>>#<<. CLOS BROWSER

One of the key components of CLOS in inheritance. The CLOS Browser provides functionality for displaying this structure and for extending it. It also provides functions for displaying and changing the class definitions and method definitions which make up a system written in CLOS.

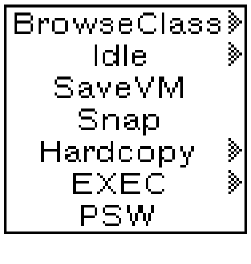
Creating a Browser

A browser can be createded in two ways:

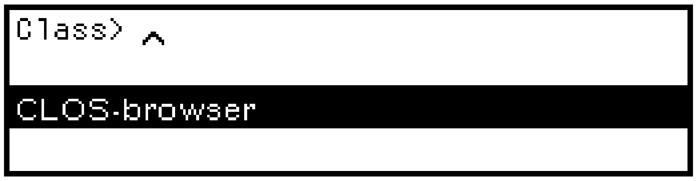
- Via a menu option from the Background Menu
- By calling the function CLOS-BROWSER:BROWSE-CLASS on a class

Creating a browser via the Background Menu

When the CLOS-BROWSER module is loaded, an enty is added to the Background Menu, as shown below:



Selecting the menu item BrowseClass brings up a window, with a prompt for the name of the class to use as the root of the browser as shown below.


Class>

CLOS-browser

Type in the name of the class you wish to browse at the flashing cursor, and the class graph will be drawn in the window.

Creating a browser programmatically

Browsers can also be created by calling the function BROWSE-CLASS:

(BROWSE-CLASS *&OPTIONAL CLASS-NAME-OR-LIST &KEY :WINDOW-OR-TITLE :GOOD-CLASSES :POSITION***)** **[Function]**

This function brings up a browser on the class named or the list of classes named. If a window is supplied for the *:WINDOW-OR-TITLE* argument, then the browser is created in that window, else an appropriately sized window is created. The window is positioned at the *:POSITION* argument or, if not supplied, then the position is set via the mouse. If a text string is supplied for the *:WINDOW-OR-TITLE* argument, then that string is used for the window title, else the string "CLOS-browse" is used. If *:GOOD-CLASSES* is supplied, then only those classes in the list are displayed.

Using the Class browser

Instances of CLOS-BROWSER are operated on through a mouse-based interface.

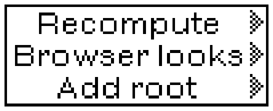
Buttoning on the browser will cause one of the following menus to be popped up:

- One menu appears when the left or middle button is pressed while the mouse is in the title bar. This menu has operations that apply to the browser itself.
- The other menu appears when the middle button is pressed when the mouse is on one of the nodes in the browser.

If the left button is pressed when the mouse is on a node, that node is boxed. This marks the node for some operations.

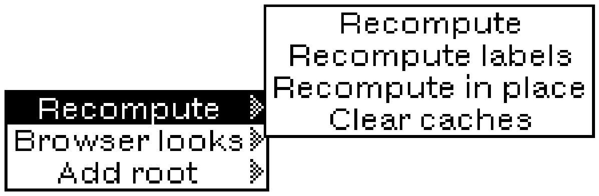
Options in the title bar menu

The following menu appears when you left- or middle-button in the title bar.



Recompute and it’s suboptions

Selecting the Recompute option and dragging the mouse to the right causes the following submenu to appear:

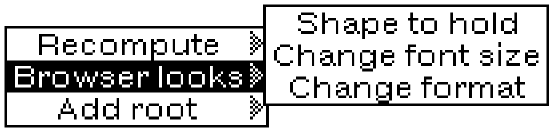


Most of these items change the appearance of the browser, not the contents.

Recompute	Recomputes the browser from the starting objects. It does not recompute the labels for each node if those labels are cached in the Label-Cache slot of the browser.
Recompute Labels	Recompute the browser from the starting objects, including the labels.
Recompute inPlace	Recompute the browser without affecting the scrolled location of the lattice within the window.
Clear caches	Clear the caches of the nodes.

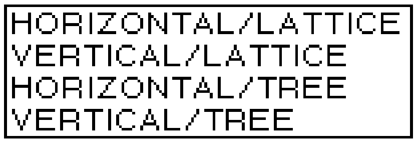
Browser looks and it's suboptions

Selecting the Browser looks menu item and sliding to the right causes the following submenu to appear:



Selecting one of these options changes the looks of the browser.

Shape to hold	Make the window for the browser just large enough to contain the browser.
Change font size	Causes a menu of alternative font sizes to pop up. Selecting one of these causes the browser to be redrawn with the nodes at that font size.
Change format	Causes the following menu to appear:

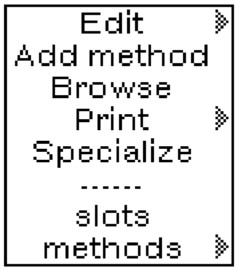


Horizontal/Lattice	Lays out the grapher as an horizontal lattice.

Vertical/Lattice	Lays out the grapher as a vertical lattice.
Horizontal/Tree	Lays out the grapher as a horizontal tree.
Vertical/Tree	lays out the grapher as a vertical tree.

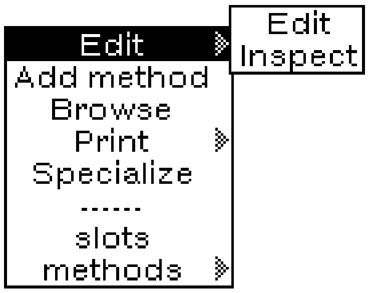
Options in the Middle-button menu

The following menu appears when you middle-button over a node in the graph:



Edit and it’s suboptions

Selecting Edit causes an editor on the class definition to be brought up. Sliding the mouse to the right causes the following menu to appear:



Edit	Edits the class named by the node
Inspect	Inspects the class object named by the node.

Add Method

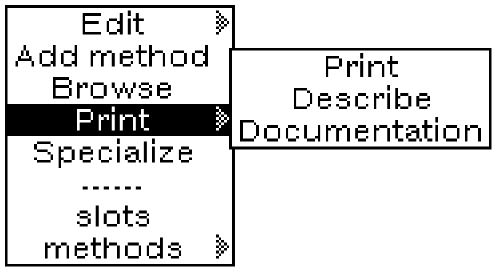
Selecting the Add Method option brings up an editor window with a template for a method to be added to that class. When the editor is done the method is installed for that class and the menu updated.

Browse

Selecting the Browse option causes a browser to be created starting with that class as the root.

Print and it’s suboptions

Selecting Print prints out the class definition. Sliding the mouse to the right causes the following menu to appear:



Print	Print’s the class definition
Describe	Describes the class, listing it’s metaclass, it’s supers classes, it’s subclasses, it’s CPL, and the number of methods specialized to it.
Documentation	Print’s the documentation string for the class

Specialize

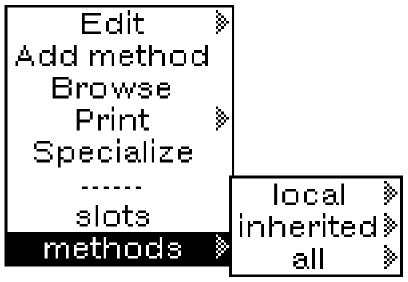
Selecting the Specialize option brings up an editor window with a template for a subclass to be added to that class. When the editor is done the class is installed and the browser updated.

Slots

Selecting the Slots option is the same as selecting the Edit option, it brings up an editor on the class definition.

Methods

The Methods option allows you to edit one of the methods defined for that class. Selecting it and sliding to the right brings up the following sub-menu:



Local	Bring up a menu of the local methods, ie methods directly defined for this class

Inherited	Bring up a menu of the methods this class inherits from it's superclasses.
All	Bring up a menu of all the methods defined for this class, both local and inherited.

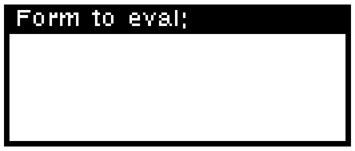
Selecting an item with the left button from the resulting menu brings up an editor on that method. If there are multiple methods that apply, a gray triangle appears in the right edge of the menu next to that item. Sliding to the right brings up a menu of method specializers to select the appropriate method.

== IMINDEX ==

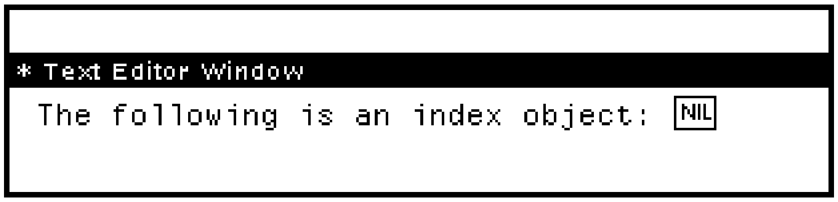
The file IMINDEX contains the functions used for creating and editing index image objects, and inserting them into a Tedit document. When a Tedit document containing index objects is formatted for printing, the index objects do not appear, but information about the index objects is put into an auxilliary "IMPTR" file. The functions in IMTOOLS can be used to take a set of IMPTR files, and generate an index.

Adding an Index Object to a Tedit Document

The simplest way of adding an index object to a Tedit document is to type ctrl-O while typing at Tedit, which will cause a window to pop up asking you to type a form to eval:



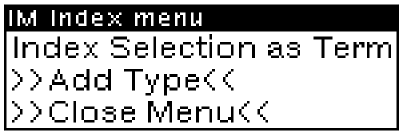
Typing (IM.INDEX.CREATEOBJ) will create an empty index object, indexing the term NIL, inserted in the Tedit document at the caret. Index objects appear in the Tedit window as words with boxes around them:



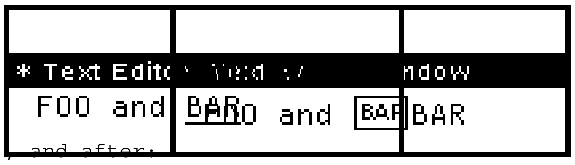
The following is an index object:

Using the IM Index Menu

An easier way to put many index objects into a Tedit document is to type (IM.INDEX.MENU) at the lisp exec, which will prompt you to position a menu that looks like this:



If the caret is blinking in a Tedit window, selecting [Index Selection as Term] with the left button will create and insert an index object that indexes the selected string. For example:

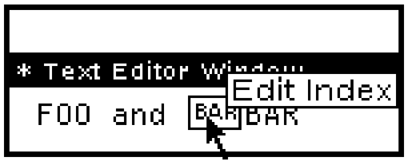


Before: , and after:

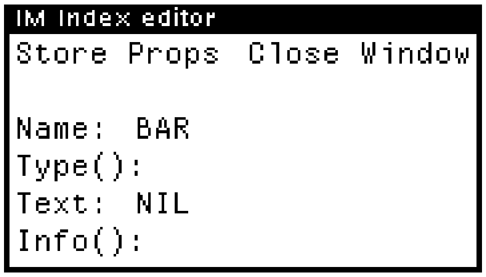
Selecting [Index Selection as Term] with the MIDDLE button will insert an index object that indexes the selected string, and then create an index object editing window, that can be used to edit the values of the fields in an index image object.

Editing an Index Object in a Tedit Document

If you select an index object, a menu will appear asking whether you want to edit the contents of the index object:



If you select [Edit Index], you will be prompted to position an index object editing window, which looks like the following:


IM Index editor
Store Props Close Window

Name: BAR
Type():
Text: NIL
Info():

This is a freemenu that allows you to edit the values of the fields in an index image object. Selecting [Store Props] stores the values in the editor into the object itself -- if the "Name:" field is changed, the Tedit window will change to show the new index name. Selecting [Close Window] will close the editor window. Note that any changes are lost if you close an index object editing window without storing the new property values.

Index Object Properties

There are more properties in an index object than show in the window initially; the window can be scrolled or reshaped to see and edit the other properties. The fields are interpreted as follows:

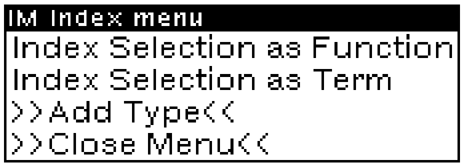
Note: The properties listed with "()", such as "Type():", interpret their value as a list of take a list of items, delimited by spaces. The other properties interpret their value as an atom, including spaces.

Name:	Value is the name used to sort and merge index entries. This should normally be all-uppercase. The [Index Selection as Term] item in the IM Index menu will automatically uppercase the selection if it is not all-uppercase, and put the real value in the "Text:" field. Examples: Name: FOO Name: BAR
Type():	Value is the type of object being indexed. If NIL, this stands for an English term. Other types are used for indexing lisp functions, variables, etc. Note that upper/lower case is important. Examples: Type(): Function Type(): Editor Command Note: The special types CHAPTER and SUBSEC (all-uppercase) are used to create entries in the table of contents, as described below. These cannot be used as index entry types.
Text:	Value is the name actually printed in the index. If NIL, the value of the "Name:" property is used. This is usually used when indexing non-uppercase items. Another place where this is useful is when you want an item containing strange characters to appear in a different place in the index. For example, if Name: = FOO and Text: = *FOO*, the item "*FOO*" will appear in the index amoung the "F" items instead of the "*" items.
Info():	Value is a list of "information" words, that mean something to the indexing programs. There are a few info words likely to be used by IMINDEX users. One is the word *PRIMARY*, which indicates that this is a primary reference. In the index, primary index page numbers are printed first, in a bold font. Another is the word *NOPAGE*, which indicates an index entry that should not be printed with a page number. This can be used to generate entries such as "FOO, see BAR". Example: Info(): *PRIMARY*
SubSec():	Probably not much use for IMINDEX users. Value is a reverse list of nested subsection and chapter numbers. For example, SubSec(): 4 2 99 indicates subsection 4 inside subsection 2 inside chapter 99. This is not used with normal index entries, except that chapter numbers, if given, are used when generating the page number in the index. For example, if the index entry specified chapter 99, and it was on page 5, it would appear in the index as 99.5. Another use for this field is if you use IM index objects to generate a table of contents, by creating index objects with type of CHAPTER or SUBSEC, as described below. In this case, the SubSec field is used to specify the subsection numbers.

Page#:	The value of this field is replaced with the page number when the Tedit textstream containing the index image object is hardcopied. Not much use to change, but it is possible.
SubName: SubType(): SubText:	Name, Type, and Text values for a sub-entry, if any, which appears under the main entry in the index.
SubSubName: SubSubType(): SubSubText:	Name, Type, and Text values for a sub-sub-entry, if any.

Adding New Types to the IM Index Menu

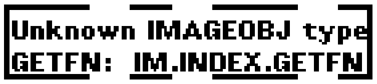
If all of the index entries in a document are terms, it is very easy to use the IM Index menu to index them. However, if there are a lot of Functions, variables, etc., you may need to edit each index entry to change the type. To make this easier, you can use the [>>Add Type<<] button in the IM Index menu to add new selections to the IM Index menu. If [>>Add Type<<] is selected, the system will prompt you to type a type name, and add a new item to the menu. For example, if you started with the initial menu, and added a new entry for the type "Function", the menu would be changed to look like the following:



If [Index Selection as Term] is selected, an index object will be created indexing the the current Tedit selection, with the type of Function.

Saving and Retrieving Tedit Documents with IM Index Objects

Tedit documents containing IM Index objects can be saved using the ordinary Tedit "Put" command, which will store all of the indexing information in the index objects. Note, however, that it is necessary to load the IMINDEX package before editing any Tedit documents containing IM Index objects -- otherwise Tedit will print the message "WARNING: Document contains unknown image objects", and all index objects will appear as the following:



If this happens, it will not be possible to use these index image objects, or re-save the Tedit document. Close the Tedit window, load IMINDEX.DCOM, and call Tedit to edit the document again.

Hardcopying a Tedit Document Containing Index Objects

A Tedit document containing index objects can be hardcopied using the normal [Hardcopy] menu button, or from the Filebrowser. Index image objects will not appear in the final hardcopy. However, while the document is being formatted, the index objects will put indexing information in an auxilliary file with the extension "IMPTR". The first time an index object is formatted, it will try to open an imptr file on the connected directory, printing a message in the prompt window: "Opening index pointer file: {DSK}<LISPFILES>FOO.IMPTR...done". When the formatting is completed, another message will appear: "Closing index pointer file: {DSK}<LISPFILES>FOO.IMPTR;1...done".

The directory used for the imptr file is the currently-connected directory (which can be changed using the CONN command). The file name used if the file name of the Tedit file, if there is one. For example, if the Tedit file is XYZ.TEDIT, the imptr file would be XYZ.IMPTR. If the Tedit text stream doesn't have a name (for instance, if the Tedit window has been brought up but never saved) the file name NONAME.IMPTR is used. The extension is always IMPTR.

For Hackers Only: If there is a need to use a different filename for the imptr file, it is possible to specify this programmatically by changing textstream properties of the Tedit textstream before the hardcopy operation. If the value of the textstream property IM.INDEX.PTRFILENAME is non-NIL, it is used as the file name for the IMPTR file. If the value of the textstream property IM.INDEX.PTRFILE is non-NIL, it is the stream used for the IMPTR information.

Creating Indicies and Tables of Contents From IMPTR Files

The file IMTOOLS.DCOM contain the routines used to create indecies and tables of contents for documents in IM format. These will also produce indicies and tables of contents from IMPTR files created by IM index objects. Loading this file will automatically load IMINDEX.DCOM, if it is not already loaded. Note, however, that you can load IMINDEX (for the purpose of editing Tedit documents with IM index objects) without loading IMTOOLS.

IMTOOLS contains the following useful functions:

(MAKE.IM.INDEX OUTFILE.FLG VOLUME.INFO IMPTR.FILES IMPTR.TYPES) [Function]

MAKE.IM.INDEX takes a number of IMPTR files, and produces an index, formatted like the one in the IRM. If OUTFILE.FLG is NIL, the output file is just sent to the default printer. If OUTFILE.FLG is T, the outfile textstream is simply returned. If OUTFILE.FLG = anything else, it is taken as a file name of an interpress file which is created <but not printed>.

VOLUME.INFO is a list of lists specifying which chapters are associated with which volumes in a large multi-volume document (like the IRM). For an example of the format, see the variable IM.MANUAL.VOLUMES in IMTOOLS. If NIL, no volume numbers are printed in the index.

IMPTR.FILES is a list of IMPTR files to be used to create this index. If NIL, any index info loaded in by explicitly calling GRAB.IMPTR is used (see IMTEDIT.TEDIT). When doing the IRM, all of the IMPTR files are loaded into global variables using GRAB.IMPTR, but users of IMINDEX probably want to specify the values specifically.

IMPTR.TYPES should be a list of IM index types to be included in the index, specified as lists. For example, if IMPTR.TYPES = ((Function)(Variable)), only function and variable entries would be listed in the index. If IMPTR.TYPES = NIL, all index entries in the IMPTR files are used.

(MAKE.IM.TOC OUTFILE.FLG CHAPTER.NUMBERS IMPTR.FILES) [Function]

MAKE.IM.TOC takes a number of IMPTR files, and produces a table of contents, formatted like the one in the IRM. All im index entries whose type is CHAPTER or SUBSEC are used as chapter and subsection pointers.

OUTFILE.FLG and IMPTR.FILES are interpreted similar to MAKE.IM.INDEX.

CHAPTER.NUMBERS is either: NIL, meaning to generate TOC of ALL data in the specified imptr files; a single number, meaning to generate a chapter TOC for that chapter; or a list of numbers, meaning to generate a TOC for those chapters.

author: Michael Sannella
file: {Phylum}<LispUsers>IMNAME. (& .DCOM)
loads file: {Phylum}<LispUsers>HASH.DCOM
loads file: {Phylum}<LispUsers>IMTRAN.DCOM [loaded by IMNAME.UPDATE.HASHFILE]

****** The IMNAME Database Package *****

IM format is the text formatting language that the Interlisp Reference Manual is represented in. It is somewhat like TEX source code, in that there are keywords, and brackets are used to delimit text. However, IM format was specifically designed for representing the Interlisp Manual, so the "text objects" used are semantically meaningful objects within the manual, such as "function definition", "lisp code", "subsection".

IMNAME is a package designed to help people who may frequently need to modify IM format files. Usually, a large document (such as the Interlisp Manual) is stored in a number of seperate files, and it is difficult to know which file contains a particular piece of information. IMNAME contains functions for analyzing a set of IM format files and building a database of "IM Names" (functions, variables, property names, etc) with pointers to the files where they are defined. Using this database, other functions allow the user to specify an IM name, and bring up a Tedit window on the appropriate file, with the cursor positioned at the right place. This tool has been very helpful to the people updating the Interlisp Reference Manual.

****** Using IMNAME to access IM format files. *****

The normal way of using an IMNAME database is by creating an "IM name inspector". This can be done using either of the following two functions:

(MAKE.IM.INSPECTOR hashFileName)
or
(IMNAME hashFileName)

hashFileName should be the name of the IMNAME database hashfile to use for this inspector. One can create multiple IM name inspectors point to the same hashfile, or to different hashfiles. Currently, I know of only two IMNAME hashfiles; the one for the Interlisp Manual and one for the Loops Manual. If hashFileName is the atom INTERLISP or LOOPS, this Inspector will point to the appropriate hashfile. If hashFileName is ommitted, it will assume that the Interlisp Manual hashfile is desired for people whose default host is Phylum, and the Loops Manual for people on Ivy. (Other people can set up their own default IMNAME hashfile by setting the global variable IM.NAME.DEFAULT.HASHFILE).

MAKE.IM.INSPECTOR sets up an "IM Inspector Window", which contains a menu. Initially, this contains the single selection "Type an IM name", which (when buttoned) prompts the user to type a name which will be looked up in the database. If an IM name is found in the hashfile, another window will appear below the first one, containing all of the "types" that the given name is known by. For example, a name may be known as both a variable and a function. If one of these types is selected, a third menu will apear below the second, listing references to the name in different files. Selecting one of the references will move the cursor in a TEDIT window (if there exists an active TEDIT window to the appropriate file), or create a new TEDIT window to the file. [Note: If a name is only known to be of one type, the "type menu" step is ommitted.]

Other functions:

(INSPECT.IM imname hashFileName)
This allows the searching for a single IM name, using several pop-up menus. In general, it is easier to use the IM name inspector.

(GET.IM.NAME.LIST hashFileName)
Returns a list of all of the IM names known within the specified hashFileName.

****** Updating an IMNAME database. *****

IMNAME works by referencing a hashfile database containing IM names and pointers to files where these names are referenced. As a set of files is modified, this information will become obsolete. In particular, simply adding a few characters to a file will invalidate all pointers to positons later in the file. This can be tolorated for awhile (it is rare that IM format files will be changed enough such that the pointers are totally off), but eventually, it is necessary to update the IMNAME database, using the following function:

(IMNAME.UPDATE.HASHFILE oldHashFileName addFileList deleteFileList flushDisappearedFlg)

This function updates an IMNAME database hashfile. First, it looks at the list of files referenced in the hashfile, and determines which of these files have been updated, by comparing version numbers. Next, every updated file is reanalyzed with IMTRAN, and updated index information is stored into an in-core hasharray. Finally, the entries in the old hashfile are read in, merged with the new info, and written out to a new hashfile.

oldHashFileName is the name of the hashfile to be updated (Note that this file must be named explicitly --- no file searches are done so that the user will not inadvertantly start updating the main manual database). The new hashfile will be created as the new version of the same file name. addFileList is a list of files that will be analyzed, and added to the database. deleteFileList is a list of files that will be deleted from the database. addFileList and deleteFileList can be used to "manage" a database, as new files are added to a document, and old ones are removed, split up,or renamed. flushDisappearedFlg determines what IMNAME.UPDATE.HASHFILE will do if it finds that some of the files in the database have disappeared (and they are not named on deleteFileList). If flushDisappearedFlg = T, the info for those files will simply be deleted. If flushDisappearedFlg = ERROR, IMNAME.UPDATE.HASHFILE will return without doing anything if files have disappeared. If flushDisappearedFlg = <anything else>, the info on the disappeared files will simply be retained.

To create a new IMNAME hashfile, pass a non-existant file name as oldHashFileName, and give a list of files as addFileList. In this case, a new hashfile will be created just from the internal hasharray info.

****** The IMTEDIT IM-to-TEDIT translation program. ******

author: Michael Sannella
file: IMTEDIT.DCOM
loads file: IMTRAN.DCOM
related files: IMTOOLS.DCOM

IM format is the text formatting language that the Interlisp Reference Manual is represented in. It is somewhat like TEX source code, in that there are keywords, and brackets are used to delimit text. However, IM format was specifically designed for representing the Interlisp Manual, so the "text objects" used are semantically meaningful objects within the manual, such as "function definition", "lisp code", "subsection". IM format is described in detail below.

IM format files are easy to edit using Tedit, but they don't look very pretty. To produce the manual, use the function IM.TEDIT, which translates IM format files to formatted Tedit text streams. These text streams can be proofread and edited by the user, or automatically printed.

A useful feature of IM.TEDIT is that is very forgiving about errors in IM format, even misplaced bracket errors! This is not to say that the output will be pretty, but at least the translation program will not bomb out on you.

***** Formatting a File with IM.TEDIT *****

To translate an IM format file into a formatted Tedit text stream, use the following function:

(IM.TEDIT *INFILE.NAME OUTFILE.FLG***)** [Function]

This function takes an IM format file, and produces a formatted Tedit text stream. *INFILE.NAME* is the name of an IM format file. *OUTFILE.FLG* determines what happens to the translated textstream. If *OUTFILE.FLG* = T, the Tedit textstream is returned by IM.TEDIT. A Tedit window showing the translated document can be created by typing **(TEDIT (IM.TEDIT** *xxx* **T))**. If *OUTFILE.FLG* = NIL, the document is immediately sent to the printer. If *OUTFILE.FLG* = anything else, it is taken as a file name for the Interpress file which is created <but not printed>.

As IM.TEDIT runs, it prints out warning messages. These messages are also saved in the file <infile>.IMERR.

Important note: It is necessary to understand that IM.TEDIT produces a totally separate document from the IM format original. Any edits to this document will NOT be reflected in the original. In general, all of the editing should be done to the IM format files, to insure that there are no inconsistancies.

Note: If IM.TEDIT is called with *OUTFILE.FLG* = T to produce a textstream, and **IM.INDEX.FILE.FLG** (below) is set to T to add index information to the textstream, the textstream cannot be stored or printed. **IM.INDEX.FILE.FLG** should only be set to T if IM.TEDIT is called with *OUTFILE.FLG* not T, so it automatically prints the document to a printer or interpress file.

***** Variables Affecting IM.TEDIT *****

The operation of IM.TEDIT affected by a number of variables:

IM.NOTE.FLG [Variable]
If T, notes will be printed out, otherwise they will be suppressed. Initially NIL. (Note: If this is T, the translation programs will print out a message to remind you that notes will be printed.)

IM.DRAFT.FLG [Variable]
If T, the output will have "--DRAFT--" and the date printed on the top and bottom of every page. Initially NIL.

IM.CHECK.DEFS [Variable]
If T, checks whether variables and functions are bound/defined in the current Interlisp environment, and prints a warning if not. For functions, will also check arg list consistancy. Initially NIL.

IM.EVEN.FLG [Variable]
If T, IM.TEDIT will add an extra page at the end of the file saying "[This page intentionally left blank]". This can be used if you need a document with an even number of pages (for double-sided copying). Initially NIL.

The following FLGs are only of interest when generating an index:

IM.INDEX.FILE.FLG [Variable]
If T, index information will be added to the formatted Tedit text stream, and the file <infile>.IMPTR will be generated containing index information when the formatted Tedit textstream is printed. Initially NIL.

IM.REF.FLG [Variable]
If T, the translation program will try to resolve cross-references by looking at various hash tables. Initially NIL.

IM.SEND.IMPLICIT [Variable]
If T, send "implicit references" for functions and variables (if not in index hash array). Initially NIL. {fn...} or {var...} objects generate "implicit references" if they are not contained in the index hash tables. This can be used to find variables and functions that are not formally defined, but only mentioned.

***** Producing an Index and Table of Contents *****

The file IMTOOLS.DCOM contains functions for gathering index information, generating an index and a table of contents, and using index information to resolve cross-references.

Currently, these tools are rather primitive. Eventually, it is hoped that they will be improved to make this process easier.

To produce an index and TOC:

(1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.

(2) Run (IM.TEDIT <file> <ipfile>) on all files in the document

(3) Evaluate (INIT.INDEX.VARS) to clear the index.

(4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).

(5) Generate an index by evaluating (MAKE.IM.INDEX OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.

(6) Generate a table of contents by evaluating (MAKE.IM.TOC OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.

To generate formatted files with cross-references resolved, do the following:

(1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.

(2) Run (IM.TEDIT <file> '{NULL}FOO.IP) on all files in the document

(3) Evaluate (INIT.INDEX.VARS) to clear the index.

(4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).

(5) Evaluate (SETQ IM.REF.FLG T), which tells IM.TEDIT to resolve cross references using the loaded index information.

(6) Format all of the files using (IM.TEDIT <file> <ipfile>).

Note that creating a formatted file with cross-references requires formatting the file twice.

The process of formatting a large set of documents such as the IRM can automated using the following function:

(DO.MANUAL *CHAPNAMES MAKE.INDEX.FLG GET.REFS NO.IP.FLG***)**   [Function]

Formats the IRM manual chapters specified by CHAPNAMES, producing Interpress files, "imptr" files (explained below), and error files. Uses the global variable **IM.MANUAL.DIRECTORY** (initially {erinyes}<lispmanual>) to indicate where the IRM files are taken from, and where the processed files should go.

Note: **DO.MANUAL** initially puts all files on {DSK}, and then copies them to the value of **IM.MANUAL.DIRECTORY**. Therefore, you should have at least 3000-4000 free pages on DSK.

"IMPTR" files are files with the extension "**IMPTR**" (such as ChapLitatoms.IMPTR), which contain index info, including the page number where each index appears. These are generated whenever a chapter is formatted. These must be read in to generate an index, or format a chapter resolving cross-references.

CHAPNAMES indicates which chapters should be processed. If *CHAPNAMES* = a list of chapter file names such as (ChapLitatoms ChapStack), only these chapters are formatted. If *CHAPNAMES*=**T**, all chapters in the IRM are formatted. If *CHAPNAMES*=**NIL**, all chapters that have been modified since the corresponding Interpress files were made are formatted. The global variable **IM.MANUAL.CHAPTERS** specifies all of the chapters, and all of the sub-files in each chapter, and the chapter numbers.

If *MAKE.INDEX.FLG* is non-**NIL**, after processing the chapters, the index, title pages, and many tables of contents are generated. Note that if this is done, all of the IMPTR files for all of the chapters are loaded again.

If *GET.REFS* is non-**NIL**, all of the IMPTR files are loaded before any of the chapters are processed, and all cross-references in the chapters are resolved.

If *NO.IP.FLG* is non-**NIL**, the chapters are formatted to the interpress file **{NULL}FOO.IP**, and thus not kept. This can be used when you simply want to generate all of the IMPTR files, but do not want the interpress files.

Ideally, all you should need to do to re-format the IRM is:

(DO.MANUAL T NIL NIL T) --- to generate the IMPTR files

(DO.MANUAL T T T) -- to generate the Interpress files, and the index.

If **DO.MANUAL** doesn’t do exactly what you want, the code is fairly self-explainatory.

***** Modifying the dimensions of the formatted pages *****

Note on modifying the formatting tools for different paper sizes: All of the lengths that IMTEDIT uses (left margin size, page height, etc.) are stored in global variables, set in the filecoms of IMTEDIT. IMTEDITCOMS also includes comments documenting the meaning of each of these global variables. Modify these variables until you get what you want.

***** Known problems with IMTEDIT *****

Currently, IM.TEDIT produces a good-looking document. However, there are some features that I was not able to provide, because Tedit did not provide the formatting capability. These are as follows:

(1) Footnotes. Tedit does not supply footnotes. Currently, I "fake" footnotes by positioning them in-line after the paragraph wherein they were created. They also will not appear within definitions, tables, or lisp code.

(2) Tables. Tedit doesn’t do table formatting. Currently, IM.TEDIT simply prints out the items in the table without formatting.

****** IM Format Description *****

This is a complete description of the syntax and vocabulary of "IM format". The general syntax is not likely to change, but the vocabulary will probably be extended as the Interlisp Manual is edited, and we discover new text objects that we need.

IM Format Syntax
****

An IM-format file consists of a linear string of visible, editable characters (and Tedit image objects, which are treated like characters). No funny control characters are allowed. "Text" is defined to be a linear string of characters, organized into paragraphs (delimited by blank lines), interspersed with any number of "Text Objects". Text Objects are used to specify the meaning of various pieces of text, which may be processed and formatted in different ways.

Text Objects can be divided into two types: those that take a single unnamed argument, and those that take a number of labeled arguments. All of the 'arguments' to Text Objects can be arbitrary text, organized in paragraphs, and including sub-text-objects nested to any level.

Text Objects within a file and arguments within a text object are specified using the characters "{" and "}". These characters are ALWAYS interpreted as Text Object delimiters or Text Object argument delimiters---there are special text objects for indicating that you really want a left or right bracket character as part of your text.

The format of Text Objects is:

```
{<TOname>  <TOarg>}        -        single argument TO
{<TOname> {<TOargname1> <TOarg1>} {<TOargname2> <TOarg2>} .... }      -        multi-argument TO
```

Between a TOname or TOargname and its TOarg, and between named TOargs, there can be any combination of spaces, tabs, and CRs. These can be used to make your file look better.

In order to help with the problem of matching brackets in large TOs (such as SubSecs, which can extend over many pages), I have introduced a new piece of syntax: Begin and End:

```
"{Begin <TOname> <tag>}"          is treated exactly like  "{<TOname> "
"{End <TOname> <tag>}"            is treated exactly like  "}"
```

except that additional checks are made when the file is processed that the "Begin"s and "End"s match up. The (optional) "<tag>"s can be used as an additional check, to distinguish between different TOs of the same type. If you use a Begin TO, you should use a matching End TO.

Note that Begin and End TOs can be used with TOs that have labeled args, for example:

```
{Begin SubSec <tag>}
{Title ----}
{Text
.
.
}
{End SubSec <tag>}
```

In order to allow the IM format translation program to recover from errors (like missing brackets), information has been included in the program about what TOs "should" appear in other TOs. In general, "complex" TOs such as LabeledLists can only appear in other complex TOs. "Simple" TOs such as Lisp can appear within anything. These rules are defined on a per-argument basis for TOs with multiple labeled arguments --- for example, a LabeledList can appear within the "Text" argument of a FnDef, but not within the "Name" argument. This feature should not cause any problems, as long as TOs are used "reasonably."

Other rules followed by the translation program: FnDef's, VarDef's, Def's, etc can only appear at the "top-level", or inside a SubSec or Chapter. (this rule is very useful, because it helps trap many bracket errors before they propogate too far) Footnotes may not be nested.

New feature <which may get changed>: If a TO name is unrecognized, it will be printed out surrounded by brackets. For example, {FOO} will actually print as "{FOO}". This allows simple expressions to be bracketed without actually using the "{bracket ...}" TO.

Text Objects currently defined:
****

(note: the order of arguments in multi-argument TOs IS significant.)

(note: No distinction is made between upper and lower case in the TO names and the argument names, or in Begin/End tags)

(note: Many TO names and argument names have synonyms. These are indicated below.)

************************* Plain Text Text objects ******

<paragraph>
All plain text is organized into paragraphs, delimited by blank lines.

{Chapter {number <number>} {title <title>} {text <text>} }
Specifies the number, title, and text of a chapter. If the number is not specified, the IM format translator will ask you to supply a number.

{subsec {title <title>} {text <text>} }
This can be used to generate sections, subsections, etc. to any depth. Heirarchical numbering is done automatically.

{Comment <text>}
Used to insert comments (which won't appear in the final formatted output).

{Note <text>}
Inserts comments that may be printed in the final formatted output, depending on the value of the variable IM.NOTE.FLG. Should be used for comments such as {Note I should write something about X here}

{foot <text>}

Generates a footnote. Footnotes may not occur within other footnotes. Currently, Tedit does not support footnotes, so these are just printed after the paragraph in which they appear.

Synonyms: footnote -> foot

{sub <text>}

Subscripts <text>.

{super <text>}

Superscripts <text>.

{it <text>}

Used to italisize pieces of text.

Synonyms: italics, emphasize -> it

{rm <text>}

Used to print text in the default, "roman" font.

{lisp <text>}

The text object for normal single-line references to lisp code. This is used for writing things like: "Obviously, {lisp (CONS 'A 'B)} evaluates to {lisp (A . B)}."

{lispcode <text>}

The text object for multiline lisp code, which do not appear in the middle of text, and have to be formatted differently. Spaces, tabs, and carriage returns are significant within <text>.

************************* TOs for Interlisp manual objects ******

{FnDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>} }

This is used to define all lisp system functions. It needs to know the name of the function, and the args, and the text of the function description. (If the function has 0 args, the {Args --} argument may be omitted.) {Type ...} is an optional argument used to specify the argument type of the function. If the keywords NLambda or NoSpread (case doesn't matter) are included in <keywords>, the function is specified to have the corresponding argument type.

Synonyms:	FnName	-> Name
	FnArgs	-> Args
	FnType	-> Type

{vardef {name <name>} {text <text>} }

Used to define system variables.

{propdef {name <name>} {text <text>} }

Used to define property names.

{MacDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>} }
Like FnDef, but for macros.

{arg <name>}
Used to talk about an abstract argument to a function, such as "x" or "y" or "number". It is used primarily within function definitions.

{fn <name>}
Used to talk about the name of a function. i.e.: "...it calls {fn CONS} to do something."

{var <name>}
Used to talk about a system variable.

{prop <name>}
Used to talk about a property name.

{Mac <name>}
Like Fn, but for macros.

{EditCom <name>}
Used to talk about an edit command.

{BreakCom <name>}
Used to talk about a break command.

{PACom <name>}
Used to talk about a programmer's assistant command.

{FileCom <name>}
Used to talk about a file package command.

************************** Indexing Text objects ******

The following TOs (for indexing, defining, and referencing) all deal with objects with specific "object-types." The Interlisp Manual contains a very large number of names (CONS, NIL, FOO, etc). It is not enough just to list the name in the index --- it is also important to indicate WHAT the name is (a function, a variable, an error message, etc.) An "object-type" is essentially the description that would be printed in the index to describe a particular name. In IM format, such a description is given by a list of words, within parenthesis. [note upper/lower case are NOT distinguished in "object-types"] For example:

(Function)
(Error Message)
(Transor Command)
(Compiler Question)

Some commonly-used object-types may be abbreviated with single words:

FN	->	(Function)
Var	->	(Variable)
Prop	->	(Property Name)
BreakCom	->	(Break Command)
EditCom	->	(Editor Command)
PACom	->	(Prog. Asst. Command)
FileCom	->	(File Package Command)
Error	->	(Error Message)
Litatom	--,	
Atom	--->	(Litatom)

[note: In {PageRef ...} and {SectionRef ...}, the word TAG can be used to refer to tags, as specified below. The word FIGURE is a synonym for TAG; this can be used to refer to a figure tag. The word TERM can be used in {index ...} to index an English term.]

{index <text>}

Creates an index reference. Some of the text objects (such as function definitions) will automatically create an index reference, but it is useful to be able to create one explicitly. This should have the format:

{index <keywords> <object> <object type>}

<keywords> (optional) can be any combination of the words
BEGIN
END
PRIMARY
<object> can be any number of words.
<object type> should be an object-type as specified above.
If this is omitted (<text> does not end with ")" and
the last word in <text> is not one of the special
object-type words), then this is the index of a term.
This can also be specified by using the word TERM.

Examples: {index *PRIMARY* BLOBBY (Transor Command)}
{index SELF-DESTRUCT SEQUENCE INITIATED Error)
{index *BEGIN* *PRIMARY* file names)

{indexX {name <name>} {type <type>} {info <info words>} {text <text>}
{subname <name>} {subtype <type>} {subtext <text>}
{subsubname <name>} {subsubtype <type>} {subsubtext <text>} }

Used for creating special index references whose printname is different from the "index name", or who have sub-index entries, or should not have page numbers in the index. <name> is the index name, used for referencing the object, and alphabetizing the index entry. <type> is the object type. <info words> (optional) can contain one or more of the keywords *BEGIN*, *END*, *PRIMARY*. <text> is the text printed in the index for this index entry.
Example: {indexX {name +} {type (Infix Operator)} {text {lisp {arg X}+{arg Y}}}} could be used to index "+" as an infix operator so that it would appear in the index alphabetized near other "+"s, but printing as "A+B".

The arguments {subname <name>} {subtype <type>} {subtext <text>} {subsubname <name>} {subsubtype <type>} {subsubtext <text>} are all optional. These are used to specify 1st and 2nd level subentries in the index. The subentry can have their own type and text definition, although these are mostly terms.

If <info terms> includes *NOPAGE*, this indicates that the index entry should not have a page number associated with it. This can be used with subentries to create "See also..." notes. Within a list of subentries, the *NOPAGE* entries will always be listed last.

Example: The following set of indexx commands:

```
{indexX {name FOO} {type (Big Command)} {text {lisp /FOO/}}}
{indexX {name FOO} {type (Big Command)} {subname ZZZ}}
{indexX {name FOO} {type (Big Command)} {subname BAZ}}
{indexX {index *NOPAGE*} {name FOO} {type (Big Command)} {subname See also BAR}}
{indexX {name FOO} {type (Big Command)} {subname BAZ} {subsubname QWERTY}
{subsubtype VAR}}
```

Would produce the index entries:

```
/FOO/ (Big Command) x.xx
    BAZ x.xx
        QWERTY (Variable) x.xx
    ZZZ x.xx
    See also BAR
```

{Def {type <type>} {name <name>} {printname <pname>} {args <args>} {parens} {noparens} {text <text>} }

May be used to specify the definition of anything. <type> should be an object-type, as specified above. <name> should be the name of the object, ideally a single word. If {printname <pname>} (optional) is given, it will be used for printing the object in the top line of the definition and in the index. If {printname <pname>} is not given, the printed name is specified by {name <name>}, {args <args>}, {parens}, and {noparens}. <args> should be the "arguments" of this object. [{args <args>} is optional.] Normally, {Def will put parenthesis around the object if and only if {Args <args>} is given. {parens} and {noparens} (if given) can be used to change this default behavior. Examples:

```
{Def {Type (Blobby Command)} {Name DOBLOB} {Args A B}
        {Text ....}}
{Def {Type (CLISP Character)}  {Name +}  {PrintName {lisp {arg X}+{arg Y}}}
        {Text ....}}
```

{Tag <tag>}

This may be used to associate <tag>, a single word (upper/lower case doesn’t matter), with a particular place in the text. This can be referenced by {SectionRef and {PageRef as described below.

{SectionRef <object-type> <object>}
{PageRef <object-type> <object>}

These TOs are used to provide a cross-reference facility. {SectionRef will print "section " and the section that the "main" occurence of the object with the given type occurs. <"main" shall remain undefined for now> {PageRef prints "page " and the appropriate page number. If <object-type> is "Tag", these TOs can refer to tags generated with {Tag <tag>}. Examples:

{SectionRef Fn CONS} could print "section 3.1.1"
{PageRef (File Package Type) Expressions} could print "page 14.34"
{PageRef Tag CompilingCLISP} could print "page 20.42"

Note: {PageRef...} and {SectionRef...} can only be used to reference 1st level index references. They cannot be used to reference subentries generated with {indexx...}

{Term <text>}
Prints <text> exactly as given, and also puts it into the index. Equivalent to "<text>{index <text> Term}".

{Figure {tag <tag>} {text <text>} {caption <text>} }
Prints a simple "figure", with figure text and caption as specified by the "text" and "caption" arguments. "{tag <tag>}" is an optional argument which, if specified, generates a tag at the beginning of the figure. This tag can be used to refer to the page or section that this figure appears on (using {PageRef Figure <tag>} or {SectionRef Figure <tag>}), or to refer to the figure number (using {FigureRef <tag>})

{FigureRef <tag>}
Prints "figure" and the figure number of the figure with the specified tag.

************************** Complex Text objects ******

{numberedlist {item <text>} {item <text>} {item <text>} ...}
Used for making numbered lists of items.

Synonyms:	Text -> Item

{unnumberedlist {item <text>} {item <text>} {item <text>} ...}
Like {numberedlist ...}, except that it uses bullets to mark each item in the list.

Synonyms:	UnlabeledList -> Unnumberedlist
	Text -> Item

{labeledlist {name <text>} {item <text>} {name <text>} {item <text>} ...}
Used for making a table associating a set of labels with descriptions of their meanings.
The argument {LName <text>} can be used to add a label left-justified on the page. Either an LName or a Name, or both, can appear as labels for an item. If an LName is supposed to be on the same line as a Name, it should appear first: {Labeledlist {LName xx} {Name yy} {item zz} ...}

Synonyms:	Label -> Name
	Text -> Item

{Table {Column} ... {Column} {First <text>} {Next <text>} {Next <text>} ... {First <text>} ... }
Used for formatting multi-column tables. The number of {Column}'s before the first {First or {Next indicates how many columns the table should have. If no {Column}'s are given,

the default is three columns. There are other formatting arguments, but they will probably change. <<<Note: This is not currently supported in Tedit. Currently, this TO only prints out the values of each {First} or {Next} argument, without formatting. Do not use tables>>>

************************** Other Text objects ******

{Include <filename>}

Used to include the text of one file within another. Files may be nested arbitrarily deep.

{Lbracket}
{Rbracket}
{bracket <text>}

These are used to insert the curly-bracket symbol into the text. {Lbracket} and {Rbracket} insert left and right brackets, and {bracket <text>} puts brackets around the given piece of text.

The following TOs are used to specify certain non-typeable characters.

{CRsymbol} or {cr}	-	carriage return symbol (currently a superscripted 'CR')
{pi}	-	Greek letter pi
{plusminus}	-	Plus or Minus ("+" + "-")
{GE}	-	greater than or equal (">" + "=")
{LE}	-	less than or equal ("<" + "=")
{NE}	-	not equal ("=" + "|")
{endash}	-	short dash
{emdash}	-	long dash
{ellipsis}	-	ellipsis (...)
{bullet}	-	bullet
{anonarg}	-	long dash (should be used in the argument list of a function definition to specify the "unspecified arguments" specified in the current manual with a dash)

FORMATTING AN INDEX

Creating a properly formatted index takes a certain amount of work (about 2 hours).

- Type in your Executive `(TEDIT (MAKE.IM.INDEX T NIL '(filenames) NIL))`
- Save the resulting file in {ERIS}<Doc>LoopsBeta>Ref>RefIndex-Raw.tedit.
- Now start dithering. The task is the change all the separators in the file. For example, the index program returns pages in the form 1.2; 10.5,38; 18.3-4. The Index format we use is in the form 1-2; 10-5; 10-38; 18-3. Unfortunately, you can't make all the necessary changes in a global substitute, since many of the separators are also in the text and the comma requires that you repeat the chapter number. Here goes:

  -- Fix the commas. Do a Find on a comma (,). Everyhwere a comma occurs in a page number, change it to a semicolon (;) and repeat the chapter number and a dot before the page number. For example, 10.5,38 becomes 10-5; 10-38.

  -- Fix the dashes. Do a Find on a dash (-). Everyhwere a dash occurs in a page number, delete it and the following page number.

  -- Fix the period. Except for he form #., you can use a global substitute from a period to a dash (. to -). This takes about 20-30 minutes, as there about 700 substitutions.

  I really recommend saving the file at this point.

- Hardcopy the index. Make whatever page breaks and other changes you feel are necessary. Make a final hardcopy and save the file.

ARCLEANUP Package --- Michael Sannella
(and augmented by Susana Wessling, 26-mar-87)

The file {eris}<lispcore>internal>library>ARCLEANUP.LCOM contains a few functions used to update the AR database and to print summaries.

(AR.CLEANUP UPDATE.FLG INDEX.LOCAL.DIR SUMMARY.FLG SUMMARY.LOCAL.DIR]

This is the main function, which should be run every few days, preferably just as you go out the door (so your machine can crunch on it overnight). If UPDATE.FLG is non-NIL, all of the ARs that have been touched since the last update are scanned, and the AR database is updated. If SUMMARY.FLG is non-NIL, a set of summaries is generated on {eris}<lispars>summaries> and press-fied.

INDEX.LOCAL.DIR and SUMMARY.LOCAL.DIR are used if you want the AR index and/or the summary reports to be cached on a local disk. This speeds up the prosess considerably, along with reducing ethernet load. If non-NIL, each of these should be a host/directory pair, which will be PACKed on the front of a filename to generate the caching filename. These are not just flags, with caching defaulting to {DSK}, so that I can use multiple partitions for caching. The issue is space: Sometimes the local disk does not have enough room for two copies of the AR index, or for both the txt and press version of the grang summary.

Example:

AR.CLEANUP (T NIL T NIL)

-- does cleanup with no caching

AR.CLEANUP (T {DSK} T {DSK2})

--- caches index on {DSK}, and the summaries on {DSK2}

AR.CLEANUP (T {DSK})

--- just does index update, caching on {DSK}.

Another useful function:

AR.CLEANUP.REDO.SUMMARIES: just generates summaries, does not do a new cleanup.

The AREDIT Interlisp bug database system

author: Michael Sannella
files: {eris}<lispcore>library>AREDIT.DCOM
doc: {eris}<lispcore>library>AREDIT.TEDIT
uses: All Tedit files

The file AREDIT.DCOM contains a number of tools useful for examining, editing, and submitting ARs ("Action Requests") related to the Interlisp-D system. These tools are loosely based on the "Adobe" tools in the Tajo environment. The Interlisp-D support group uses this system to keep track of the state of outstanding bug reports. There are currently over 2000 ARs in the database.

These tools can be used from any machine running Interlisp-D which can establish a leaf connection to the Phylex: file server, where the database files are currently stored, and the ERIS file server.

After loading AREDIT.DCOM, the user can create two types of windows: AR edit forms and AR Query forms.

The AR Edit Form

An AR edit form is used to examine, edit, and submit ARs. To create an AR Edit Form, evaluate (AR.FORM). Interlisp will prompt you to specify a region for the form window -- the best size to give it is one about half the width of the screen and at least half the height of the screen. The form window which will appear contains three subwindows: (1) On the top is the message subwindow, where prompts and status messages are printed; (2) in the middle is the command subwindow, a menu of commands for editing / submitting ARs; (3) on the bottom is the form subwindow, where the information in an AR is displayed.

The command subwindow contains the following commands:

New -- Buttoning this word clears the fields of the AR in the form subwindow. Some fields (**Source, Submitter, Status**) are initialized to appropriate values for a new AR.

Get -- Buttoning this retrieves the AR whose number follows "**Number:**" in the command subwindow.

Put -- Buttoning this will either store an edited AR, or submit a new AR. Which one (submit new or store old) depends on whether the last operation was "**New**" or "**Get**". If the current AR displayed was retrieved with "**Get**", then "**Put**" will store it as the old AR. If this AR was built up from scratch after buttoning "**New**", then "**Get**" will submit is as a new AR. The title of the form subwindow gives an indication of what state the form is in: if it says "New Bug Report", then "**Put**" will submit it. If it says "Editing AR xxx", then "**Put**" will store it. [There are plans to improve this interface]

Number: -- This is a text field just like text fields in the form window (see below) used to specify the number used by "**Get**". Buttoning the word "**Number:**" will pending-delete-select the value of the field, so you can delete it and insert a new number. If the character carriage-return is typed, then a "**Get**" is automatically done on the value of this field. This is faster than typing a number, and buttoning "**Get**".

The form subwindow contains a large number of fields. The meaning of these fields is described in XXX. The value of these fields can be edited as follows:

"Enumerated fields" can only contain certain values. These are indicated in the form subwindow by field names followed by curly-braces "{}". To change the value of one of these fields, button the field name; a menu of permissable values will appear; select a value; it will be inserted between the braces.

[note: Some of the enumerated field values are dependent on other fields. For example, the values of "Subsystem:" depend on the value of "System:" -- if the "System:" value is changed, the "Subsystem:" value is automatically set to NIL. The fields with this relationship are System:/Subsystem: and Machine:/Disk:.]

"Text fields" can contain arbitrary text. These fields do not have braces after the field name. The text can be edited using normal Tedit editing. Buttoning the field name will pending-delete-select the entire field value, which allows the whole field to be easily deleted.

[note: Currently, stored ARs only contain straight text. Any tedit formatting information put into an AR will be lost when the AR is stored. Image objects (like bitmaps) are also not stored.]

[note: A few of the text fields, like "Number:", are read only --- they cannot be edited by the user.]

[note: in older versions of AREDIT, some text fields (such as "Attn:" could only contain a certain number of characters. The user could type as many characters as he wanted, but an error would occur when the "Put" command was executed. This has now been changed --- any text field can contain an arbitrary number of characters.]

The AR Query Form

An AR Query Form is used to search the AR database for all ARs with particular characteristics. One can search for all ARs with a given name in the "**Attn:**" field, all ARs which have Status: = Open, etc. These ARs can be sorted, and a summary of the selected ARs can be printed into a file.

To create an AR query form, evaluate (AR.QFORM.CREATE). Interlisp will prompt for a region (the default size is ok), and create a window with three subwindows: (1) on top, a message subwindow, for printing prompts and messages; (2) in the middle, a browser subwindow, used for displaying the ARs seleced by a query; (3) on the bottom, the AR query command subwindow, containing a number of commands and fields.

The Ar Query command subwindow contains the following fields/ commands:

Query List: -- This field is used to specify which ARs the "**Query**" command will search for. This field should be filled with an AR query spec, which has one of the following forms:

(<field> HAS <val>) searches for all ARs whose text field <field> contains <val>. <string> may either be an Interlisp string or an atom. The search is case-independent: foo matches Foo matches FOO.

(<field> IS <val>) searches for all ARs whose enumerated field <field> has the value <val>.

(AND <spec1> <spec2> ... <specN>) returns all ARs satisfying ALL of the given specs.

(OR <spec1> <spec2> ... <specN>) returns all ARs satisfying ANY of the given specs.

[note: an implicit (AND) is wrapped around the value of the "Query List:" field, so just giving a number of specs will AND them together.]

Not every AR field can be searched for in the same way: some can only be searched with HAS, some can only be searched with IS, and some (like the Description: field) cannot be searched at all. To find out the possible query specs, button the words "**Query List:**" --- this will put up a menu of all of the permitted searching options. Some of these menu items have submenues. When one of the options is selected, it is added at the end of the value of this field.

Examples:

Query List: (Subject: HAS foo)
Searches for all ARs whose subject contains the string "foo".

Query List: (Status: IS Open) (Attn: HAS sannella)
Searches for all open ARs which have "sannella" in the Attn: field.

Query List: (OR (Status: IS Declined) (Status: IS Superceded))
Searches for all ARs with Status: either Declined or Superceded.

Sort List: -- This field determines how the Query-ed ARs will be sorted. Currently, ARs can only be sorted by the values of enumerated fields. Buttoning the words "**Sort List:**" will put up a menu of the permitted field names -- selecting one will add it to the value of this field.

Example:

Sort List: Status: Priority:
This will sort first by the Status: field, and then by the Priority: field.

[note: After sorting by all given fields, if two ARs are the same, they are sorted by AR number. Therefore, if this field is left blank, the queried ARs will be in numerical order]

Query -- Buttoning this command will initiate the query specified by the "**Query List:**" field, and sort it according to the value of the "**Sort List:**" field. While the query is in progress, the AR query command subwindow is greyed-out. When the query is completed, the numbers, subjects, etc of the ARs which have been found are displayed in the AR query browser subwindow. This window can be scrolled both vertically and horizontally.

Print File: -- This field can be filled in with a file name, which is used to specify the file that the **Print** command should store a report. If left blank, a window will pop up on the screen, and the information will be displayed there.

Print -- This prints a detailed summary of all of the ARs from the last **Query** into the file given in the **Print File:** field.

To generate and print a summary of a selected group of ARs, fill in the **Qury List:** and **Sort List:** fields, select **Query** and wait for the query to complete, fill in the **Print File:** field, and select **Print**. The summery is rather wide -- it may be a good idea to use the LANDPRESS package to printit out sideways on a printer.

The AR query browser window:

This window shows a short summary (one line each) of the ARs that have been queried. Left-buttoning one of the AR lines will call AR.SHOW on that AR, to display it. Middle-buttoning an AR line will "**Get**" that AR into a specified AR edit form window.

Background menu commands:

When AREDIT is loaded, the item "AREDIT" is added to the background menu, with a number of subitems. These are interpreted as follows:

AREDIT -- Creates a new AR edit form, initially cleared.

New AR Form -- Same as AREDIT

Load AR Form -- prompts the user for a number, and creates a new AR edit form, with the specified AR number loaded initially.

AR.SHOW -- prompts the user for a number, and calls AR.SHOW, an old version of AREDIT which quickly displays the contents of a given AR. It prompts for a window region the first time it is used -- thereafter it uses the same window.

AR Query Form -- Creates a new AR query form.

Auxiliary AR edit form commands.

Pressing the left mouse button in the title bar of the AR Edit form command subwindow will bring up a menu of less-used commands:

Clear -- Clears ALL the fields of the current AR. Similar to New, except that non of the fields like Source:, etc., are filled in. This is useful when you are submitting an AR for someone else.

New
Get
Put -- the same as the Commands in the edit command subwindow.

Put&Get -- prompts the user for a number, Puts the current AR, and Gets the given numbered AR. Useful when scanning through a number of ARs.

GetFromFile -- Prompts the user for a file name, and loads the information from that file into the AR edit form subwindow. If this file is not in the right format, an error will be generated.

PutToFile -- Prompts the user for a file name, and stores the information from the AR into that file.

Locally caching the AR index.

All AR query operations use the information in the "AR Index" file. This file, which is updated every few days, is stored as {eris}<lispars>AR.INDEX. To speed up Query operations, this file can be copied to a local file server, or the local hard disk. Warning: this file is currently ~700 IFS pages long, and it will undoubtedly get larger. Also, it is the responsibility of the user to make sure that they update their local version of AR.INDEX when the master copy is updated.

To use a local version of AR.INDEX, give the file name as an argument to AR.QFORM.CREATE:

(AR.QFORM.CREATE '{DSK}AR.INDEX).

Global Variables that control AREDIT

The following are global variables that can be set to alter the operation of AREDIT. These are the only global variables that it is safe to change.

AR.ENTRY.LIST.WINDOW.FIELDS -- Controls which fields are displayed in the AR query browser window, along with the widths of the fields.

AR.ENTRY.LIST.PRINT.FIELDS -- Controls the fields displayed by the Print command of the AR query form.

AR.ENTRY.LIST.PRINT.MULTILINE.FLAG -- if non-NIL, the Print command of the AR query form will print all of the characters in each field, using multiple lines for those field values bigger than the field width allowed. If NIL, each AR will use only one line, truncating any field values that are too big.

DATEPATCH

By: Bill van Melle
(vanMelle.pa@Xerox.com)

DATEPATCH fixes some bugs and extends the functionality of the date functions `DATE`, `GDATE` and `IDATE`.

Date Parsing

The date parser (`IDATE`) now handles all dates legal in RFC822 syntax (except the silly single-digit military time zones). In addition, it handles months spelled out, months abbreviated with a period, and ignores initial strings of the form "{letter}*,", assuming these to be specifying a day, as in "Monday, May 1, 1989". In addition to the official time zone specifications, it also recognizes any in the list `TIME.ZONES`, whose format has changed slightly:

`TIME.ZONES` [Variable]

An association list whose elements are of the form (*offset regzone dstzone*), where *offset* is the number of hours west of GMT (note that this, unfortunately, is opposite in sign to the RFC822 standard, but is strictly an internal matter), *regzone* is a string specifying the time zone normally, and *dstzone* is a string specifying the zone when daylight savings time is in effect. If *dstzone* is omitted, then there is no representation for that zone in daylight savings time, and `DATE` is forced to use absolute syntax (e.g., +0400).

The initial value of `TIME.ZONES` is

```
((8 "PST" "PDT")
 (7 "MST" "MDT")
 (6 "CST" "CDT")
 (5 "EST" "EDT")
 (0 "GMT" "BST")
 (0 "UT")
 (-1 "MET" "MET DST")
 (-2 "EET" "EET DST"))
```

`IDATE` also accepts an optional argument DEFAULTTIME, which is interpreted as a number of seconds past midnight. If the date string does not contain a time, DEFAULTTIME is used; if DEFAULTTIME is `NIL`, `IDATE` returns `NIL` in this case (as it always has).

Date Output

The date printers (`DATE` and `GDATE`) now produce appropriate time zones outside of the U.S. Given a choice of time zones, they take the first entry found in `TIME.ZONES`. In addition, they support a few more `DATEFORMAT` options:

`MONTH.LONG` [DateFormat Option]

Provides for full names of months rather than the first three characters. For instance, `(DATE (DATEFORMAT MONTH.LONG SPACES NO.TIME))` might produce "20 February 87".

`MONTH.LEADING` [DateFormat Option]

Causes the month to be produced as a word before the day and the day to be followed by a comma. For instance, `(DATE (DATEFORMAT MONTH.LEADING MONTH.LONG YEAR.LONG NO.TIME))` might produce "February 20, 1987". `MONTH.LEADING` implies `SPACES` and inhibits `NUMBER.OF.MONTH`.

`CIVILIAN.TIME` [DateFormat Option]

Specifies 12-hour time instead of 24-hour (military) time. For instance, `(DATE (DATEFORMAT CIVILIAN.TIME NO.DATE NO.SECONDS))` might produce "11:34pm".

For completeness, listed below are all the `DATEFORMAT` options currently supported.

Those affecting the date portion:

`NO.DATE`	Omit the date portion (month, day, year, day of week).
`NUMBER.OF.MONTH`	Use a number for the month instead of spelling it.
`MONTH.LONG`	Spell the month out instead of abbreviating it.
`MONTH.LEADING`	Month before day, spelled out, comma after day.
`YEAR.LONG`	Use 4 digits for year instead of 2.
`DAY.OF.WEEK`	Include day of week (it appears at the end of the string, in parentheses).
`DAY.SHORT`	Use 3-letter abbreviation for day.
`SLASHES`	Separate parts of date with slashes instead of hyphens.
`SPACES`	Separate parts of date with spaces instead of hyphens.
`NO.LEADING.SPACES`	Omit leading spaces (default is a fixed format that always "lines up"). This also affects the hour when `CIVILIAN.TIME` is specified.

Those affecting the time portion:

`NO.TIME`	Omit the time portion (hour, minutes, seconds, zone).
`NO.SECONDS`	Omit seconds.
`CIVILIAN.TIME`	12-hour instead of 24-hour time.
`TIME.ZONE`	Include time zone specification.

FILEBANGER

Filed as {eris}<lispcore>internal>library>FileBanger.*
Not for customer release

Basic Functions

(DOFILEBANGER *Destination Length NoBreak*)

Starts a new file banger in its own process. The device and directory specified by *Destination* will be where the test files are created during the testing. Each file will be *Length* bytes long. The file banger performs consistency checks on all the files it creates after each operation. If those consistency checks fail, it will print a message indicating what it found wrong, and if *NoBreak* is NIL will open a break window.

To stop the file banger process, one may (KILL.PROCESS 'FILEBANGER), provided that only a single file banger is in operation.

(FILEBANGER *TestFile Destination MakeWindow NoBreak InParms OutParms*)

Enters a loop of creating, manipulating, and deleting files, making consistency checks after each pass. *TestFile* may be the name of a file (whose contents are used as the contents of the test files), or an integer, in which case a test file of thet many random bytes is created. *Destination* is the device and directory to be tested (normally this should be {DSK}<LispFiles>). *MakeWindow*, if T, causes a fresh window to be created; that window is used for progress messages as the test runs. If *MakeWindow* is NIL, the regular TTY window for the process is used (and created, if need be!). If the test's consistency checks fail, FILEBANGER fill print a message in the status window; if *NoBreak* is NIL, it'll also stop and open a break window. If you are interested in really stressing the file system, you may specify PARAMETERS arguments to be given to OPENSTREAM when files are opened. *InParms* is used when opening for input, and *OutParms* when opening for output. We don't recommend the use of these arguments.

Generally speaking, the test file should be of a good length (5000 bytes or more), to assure that the files spread out to cover the disk volume you're testing.

FILEBANGER continues to run until interrupted from the outside somehow. The easiest way to arrange that is to run it in its own process (using DOFILEBANGER) and use KILL.PROCESS.

Recommended Operation

(DOFILEBANGER '{DSK}<LispFiles> 5000 NIL)

This will open a TTY window for the FILEBANGER process, and print small ongoing status messages there; you will need to do something to prevent the scrolling from hanging things up when the TTY window is full (either type ahead some spaces in that window, or arrange for the window not to pause when it fills up). To stop the file banger when it has run long enough, do (KILL.PROCESS 'FILEBANGER) in the top-level TTY window.

Likely Error Messages

HARD DISK ERROR

The low-level disk code hit a hard error on the disk, and was able to complete its operation to the extent of recognizing the error. Indicates a bad spot on the disk.

VERIFY ERROR

The disk-page label checking code found an inconsistency. While this might be software, it is most likely to be controller trouble or a bad spot on the disk.

(infinite uninterruptible loop)

The low-level disk code hit a snag so severe that it could never complete the disk operation at all. This should never happen. If it does, suspect IOP trouble.

File1 and File2 differ at byte xxx

Two files should have been identical but weren't. Suspect software problems, unless the error follows a particular drive, processor, or IOP.

File1 has length xxx, but File2 has length yyy.

Two files should have been identical but weren't. Suspect software problems, unless the error follows a particular drive, processor, or IOP

```
Files in this directory /usr/local/lde/i ternal/
where copied from {eris}<lispcore>internal>library>
31-Jan-90
```

LISP DIAGNOSTICS

Date : 05 - 21 - 1985

This document describes Lisp Diagnostics program for Xerox 1108 and Xerox 1109 workstations.

To start it, load LISPDIAGNOSTICS.DCOM from the Lisp Library.

As soon as it finishes loading the file, a menu will appear.
You will have 2 choices, i.e. : Start Exercise and Stop Exercise.

If the file has previously been loaded, you'll only have to type :
(MAKEDIAGNOSTICSMENU) to make the menu appear.
When you select Start Exercise, the following message will appear on the Top level typescript window :

```
                      Legend

! -> Completed !DIAGNOSE of MACROTEST
@ -> Completed TANSPEED benchmark
# -> Completed BROWSE benchmark
¤ -> Found a Clearing House on the Ethernet
- -> Looked, but failed to find a Clearing House
[xxx] -> Tried 32 retrievals from CH and got xxx failures
{xxx} -> Copied and deleted xxx copies of the Disk file
(xxx) -> Finished with xxx'th run of the EMUPROC loop
GDATE on new line marks release of working set pages
```

In addition to that, a window appears on top of the menu, it describes the current activity indicated by a black rectangular cursor.
They are :
1. Swap out working set.
2. Ethernet activity.
3. Disk Activity.
4. Benchmark.

To stop the diagnostics, simply click the Stop Exercise selection.

(This package was superficially tested on Intermezzo>Full.sysout on
05 - 21 - 1985 by G. Santosa)

NSMAIL

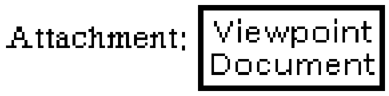
INTRODUCTION

The module NSMAIL implements the protocols to allow Lafite to be used to send and retrieve Xerox NS Mail. Load the file NSMAIL.LCOM. To run this in Lyric, you must have loaded the LispUsers module NSRANDOM as well (q.v. for important loading information). If you don't have NSRANDOM loaded, you can't use the "Put to file" command described below.

If you have both Grapevine and NSMAIL implementations loaded, you must set Lafite's mode to NS. Use the "NS Mode" subitem underneath Lafite's Quit command, or call (LAFITEMODE 'NS). You must also be a registered NS user, and have a mailbox.

ATTACHMENTS

The main difference between this and earlier versions of NSMAIL is that "attachments" are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:


Attachment: Viewpoint Document

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

View as text	This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)
Put to file	This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint.

Send to Printer	This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (must be one that accepts Interpress, of course), and sends the attachment to it for printing.
Expand folder	This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The Expand folder command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the Put to file command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, Put to file (usually) produces an ordinary (non-directory) file.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through a mail gateway).

There is currently no mechanism for creating your own attachments to end to other users.

MISCELLANY

If you prefer the old behavior of leaving the attachments behind in the mailbox, set the variable NSMAIL.LEAVE.ATTACHMENTS to T, but this use is discouraged. You must take care to regularly retrieve your mail from somewhere (such as Viewpoint) that will flush out all the mail; otherwise, the mail with attachments, whether you want them or not, accumulate on the server.

When in NS mode, Lafite will want your NS login identity. Normally, if your NS password differs from your default password, you will be prompted to login. You can also call (LOGIN '|NS::|) yourself to set your NS login.

You can freely intermix Grapevine and NS mail in the same mail folder if you like, but the Answer command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result. Also, the status window always shows you mail status only of the current mode.

SOURCELOOKUP

INTRODUCTION

This module provides a mechanism to locate the source file and floppy in which a particular system function is defined. It is designed for use with the set of Xerox LISP source files distributed on floppy with the Lyric release. This module uses the WHEREIS module distributed with Lisp Library documented in Section 23 of the ***Interlisp Reference Manual***.

LOADING THE MODULE

SOURCELOOKUP requires WHEREIS.DCOM and HASH.DCOM to be loaded. The variable WHEREIS.HASH must be set to the hash file LYRICSOURCES.WHEREIS. All these files are distributed on the floppy labeled Lyric Sources #10. The hash file must reside on a random access device. To use the hash file on FLOPPY do the following:

```
(SETQ WHEREIS.HASH '({FLOPPY}LYRICSOURCES.WHEREIS))
```

The variable LyricSourceIndex will be set to a list where each element is a list whose CAR is the name of a Lyric Source floppy and CDR is the list of files contained on the floppy.

USER FUNCTION

(LOCATE.FUNCTION *function*)

If *function* is defined in a distributed source file prints out the appropriate file name and floppy and returns the file name; else prints "*function* not found" and returns NIL.

EXAMPLES

(LOCATE.FUNCTION 'SETQ)
The function **SETQ** is defined in the file **LLINTERP** located on floppy **Lyric Source #5**.
(LLINTERP)

(LOCATE.FUNCTION 'UNDEFINED)
UNDEFINED not found.
NIL

SETUP instructions for
PC Emulation virtual hard disk and real PC floppy disk access

1. Make sure all of the normal PCE .DCOM files are loaded to run PCE:
   (PCEWINDOW, PCE, PCEERD, PCEDISPLAY, PCEKEYBOARD).

2. Select PC Emulation on the background MENU. This may take a while to load the PC fonts.

3. Put MSDOS disk in floppy drive. Select BOOT in the command window.

4. If the PCE boots ok, then select QUIT in the the command window.

5. Now we will make a virtual hard disk for the PCE.

   From LISP do the following:

```
1_(PCE.CREATE.ERDFILE 'PCE.DISK 50)

                      (* this will make a lisp file PCE.DISK which is nearly 2
                          megabytes big - make sure you have enough disk space)
```

6. Reboot the PCE (BOOT command) and make sure configuration has the right file name PCE.DISK for the ERDFILE using the CONFIG command.

7. From the PC window:

```
-FDISK
                      (* create DOS partition using all of the disk )

-FORMAT C:/V/S
                      (* format the rigid disk, put a syste on it, and label)


-COPY A:*.* C:
                      (* copies all of the system files from floppy to ERD)
```

8. Select QUIT from the COMMAND window.

9. You may now boot from the ERD (emulated rigid disk) by selecting FIXED DISK as boot device and Set-Config from the configuration window.

10. From LISP exec:

```
2_ LOAD(VPCDISK.DCOM)
                      (* loads in LISP virtual PC disk/floppy and real PC floppy access)
3_ (VPCDISK.CREATE.DEVICE 'PCDISK 'PCE.DISK T)
                      (* associates ERDFILE PCE.DISK with a device name PCDISK)
4_ DIR {PCDISK}PCDISK:*
                      (* directory listing of the PC DISK)
5_(SETQ PC.TEXTFILE.EXTENSIONS '(TXT PAS BAS 1 2 3]
                      (* list of PC file extensions which are treated as TEXT type files)
6_(VPCDISK.CREATE.DEVICE 'PCFLOPPY '{PCFLOPPY})
                      (* associates real PC floppy driver with {PCDISK} device PCFLOPPY)
                      (* make sure real floppy is in the disk drive when you execute this)
7_DIR {PCDISK}PCFLOPPY:*
                      (* directory listing of the real floppy)
```

11. At this point you are set up to do any LISP commands you want such as Filebrowser, Hardcopy, TEdit, etc. using the {PCDISK}PCDISK: for the virtual hard disk, and {PCDISK}PCFLOPPY: for real PC floppies.

12. Try using Filebrowser and then selecting a textfile that has one of the extension names you setup in PC.TEXTFILE.EXTENSIONS above, and selecting HARDCOPY.

13. Try using TEDIT on a floppy file such as {PCDISK}PCFLOPPY:README.1 and then editing the file and writing (PUT) it back. and then reselect it from filebrowser and selecting hardcopy. (Make sure floppy is not write-protected.)

14. You may also access files on sub-directories. IE, {PCDISK}PCDISK:<DIR>FILE - However, you cannot delete or create directories from LISP. You have to do this from the PC window.

MASTERSCOPE EXTENSIONS

1. More than one type of executable thing.

All possible executable thing types are stored on MSFNTYPES, which is a list of MSANALYZABLE records:
(filepkgname setname getdef-fn)
Vanilla Masterscope starts with it set to
((FNS FNS GETDEF))
and new things get added to it through MSADDANALYZE (see below).
The filepkgname must be the File Manager name of the new executable thing. MSADDANALYZE will barf if filepkgname is not on IL:FILEPKGTYPES.
The setname must be the name Masterscope uses for a set of the new executable things.
The getdef-fn has to take the same arguments as GETDEF and return NIL or a definition suitable for Masterscope munching.

MSGETDEF, MSEDITF, MSSEEKTYPE, MSMARKCHANGED, and MSADDANALYZE look at MSFNTYPES

Subject: New Lispcore>Library package: MesaTypes
To: Lispcore^, Lispsupport, Sheil, Cooper, Purcell

Announcing a new Lispcore>Library package: MesaTypes

By Tayloe Stansbury with help from Richard Burton.

This package introduces three new clisprecordtypes which allow you to describe any block of bits with an arbitrarily nested datatype. You can define multidimensional, nonstandardly indexed arrays with multi-word elements; records with multi-word fields; and multi-word types. Special accessfns cover up the necessary \BLTs and LOCFs. Appropriate create methods are automatically provided. The package also provides a number of macros for manipulating instances of such types.

Anyone who wants to use graceful datatypes to describe some arbitrary chunk of memory (e.g. ethernet, rs232, nsfiling, any file system) will probably find this package useful; most of the 1108 file system now depends on it. People translating Mesa system code into Lisp will find it particularly useful.

This message is stored as Lispcore>Library>MesaTypes.tedit. Proper external documentation of this package will follow release of the 1108 file system. Extensive examples of its use can be found in the first few pages of stansbury>newdlionfs>dlionfs.

-- Tayloe.

Common Lisp package code and symbol conversions
Ron Fischer

[This document is for developers only. It is not intended to be part of any external software release.]

The package code can behave mighty strangely if not operated "just so." Please read this document carefully.

Loading

The package code is now loaded in the full.sysout. It lives in `{Eris}<LispCore>Library>` on

```
        CMLSYMBOL.DCOM
        CMLPACKAGE.DCOM
```

Initializing

Packages, as you might expect, must be bootstrapped carefully. At its simplest, initialization is accomplished by calling just one function:

```
(package-init t)
```

simple startup

`(package-init &optional (convert? nil))` [Function]
Clear, make structures of, initialize & convert symbols (if `convert?` is `t`) to, and enable use of the symbol package system. On a DandeTiger this takes about 25 minutes.

The bootstrap actually takes place in three steps. The first one creates all the package structure needed. The second passes over all the atoms in the sysout to "tag" them with packages. The third enables the package world.

step 1: make system packages

`(package-make)` [Function]
Create, but do not fill with symbols, the base packages that need to exist. Also enables the package qualifier characters in the readtables and saves the old definitions of \READ.SYMBOL and \MKATOM.

step 2: convert existing symbols

`(package-hierarchy-init &optional (convert? nil))` [Function]
Fill all the initial system packages with their proper symbols, moving litatoms into appropriate places and such. If `convert?` is non-nil then symbols whose pnames have fake package qualifiers, like cl:length, will be converted IN PLACE to remove the qualifier. If conversion takes place you cannot fully disable the package system.

step 3: enable package system, remove old hacks

The third part of the bootstrap enables the use of packages by the reader, printer and friends. This involves redefining `\MKATOM` and `\READ.SYMBOL`. Also, if `*package*` is set to `nil` things will not go well. Please use `package-enable` and `package-disable` to start and stop, as these fellows know what they're doing. These faithful functions are availible on a menu by calling package-menu.

`(package-enable &optional (package *interlisp-package*))` [Function]
Turn on the package system, making package the current one and redefining `\READ.SYMBOL` and `\MKATOM` appropriatly.

`(package-disable)` [Function]
Turn off the package system and restore the old definitions of `\READ.SYMBOL` and `\MKATOM`. After disabling, symbols interned under the package system will not be `eq` to symbols of the same name reread.

`(package-menu)` [Function]
Make a menu that allows turning the package system on or off without using the reader.

`(package-hacks-disable)` [Function]
Eliminates package simulation hacks when loading over an old sysout. These hacks cannot be re-enabled.

`(pkgconvert-enable)` [Function]
Enables a change in the behavior of the function `\\READ.SYMBOL` (using the function `check-symbol-namestring`), which converts symbols being read based on their "looking like" a pacakge prefixed symbol name. "Looking like" is defined by a table in the global `litatom-package-conversion-table`.

`litatom-package-conversion-table` [Variable]
a global variable containing a list of clauses used by the functions package-hierarchy-init and check-symbol-namestring to determine if a symbol "looks like" is trying to be package qualified. The list contains clauses with the following structure:

(*prefix-string list-of-exception-strings package-name-string where*)

prefix-string - string containing the prefix of symbols which this clause converts, eg `"CL:"`
list-of-exception-strings - list of strings naming symbols which this clause should leave alone, eg `("CL:FLG")`
package-name-string - string containing the name of the package that symbols converted by this clause should wind up in, eg `"LISP"`
where - either `:INTERNAL` or `:EXTERNAL`, indicates whether symbols converted by this clause should be external or internal in their package.

The initial value of this variable is:
```
(      ("CL:"      ("CL:FLG" "CL:MAKE-SYMBOL" "CL:COPY-SYMBOL" "CL:INTERN"
                    "CL:MAKE-KEYWORD" "CL:GENTEMP" "CL:KEYWORDP")
                         "LISP"            :external)
       (":"        nil    "KEYWORD"    :external)
)
```

Notes & cautions

The read functions may return strings if you use package qualifier syntax and `*package*` does not contain a package. This is part of debugging code that Bill put into them. Beware especially of making `*package* NIL` without calling `package-disable`.

The list of conversion clauses is searched linearly, hence longer prefixes should come before shorter ones with the same chars in them, ie put a clause for `"CL::"` before one for `"CL:"`.

Missing features

Cannot be placed early into loadup due to dependancies on CL files.

Apropos (and other Interlisp litatom functions) are not redefined to operate with packages.

Performance

There have been some small improvements since these timings were taken.

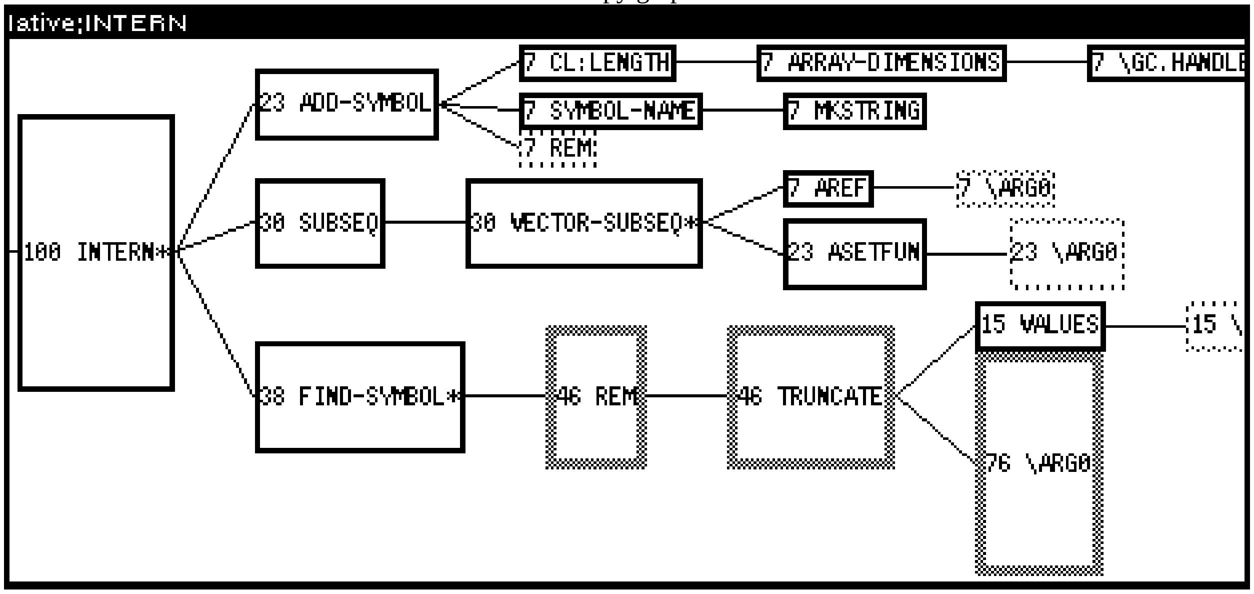
```
FILEDATES : (("21-Jul-86 14:12:17" .
{ERIS}<LISPCORE>CML>LAB>CMLPACKAGES.;129))
```

Testing symbol / litatom creation. Old array package. There are 6 random symbols made (new ones). intern is a factor of 16 slower, conses heavily, makes 3 strings for each call.

```
85#(TIMEALL (TIME.MKATOM))
Elapsed Time =       .336 seconds
SWAP time =           .32 seconds
CPU Time =           .016 seconds
PAGEFAULTS = 8
LISTP
9

86#(TIMEALL (TIME.INTERN))
Elapsed Time =       .449 seconds
SWAP time =          .188 seconds
```

```
CPU Time =           .261 seconds
PAGEFAULTS = 5
LISTP   STRINGP
423     18
```

Breakdown spy graph follows:



Peano

Maintained By: Frank Shih (Shih.envos@Xerox.com)

This document last edited on 8-Nov-88

INTRODUCTION

Peano is an ancient graphics demo.

OPERATION

(PEANODEMO *LEVEL SCALE*) [Function]

Runs a demo in PEANOWINDOW (set first time), drawing Peano curves to level *LEVEL* with lines *SCALE* units wide.

Read and Print state profile

There are a large number of special variables that control reading and printing. Taken together these comprise a "mode of operation" for the reader and printer. Because there are a fair number of such variables a method for capturing and restoring their state has been provided.

(xcl:make-read-print-profile &key (readtable *readtable*) (read-base *read-base*) (read-suppress *read-suppress*) (package *package*) (read-default-float-format *read-default-float-format*) (print-escape *print-escape*) (print-pretty *print-pretty*) (print-circle *print-circle*) (print-base *print-base*) (print-radix *print-radix*) (print-case *print-case*) (print-gensym *print-gensym*) (print-level *print-level*) (print-length *print-length*)) [Function]

Creates a read-print-profile object. The default values of its components are taken from the current special bindings of the variables shown in the argument list above.

(xcl:copy-read-print-profile profile) [Function]

Creates a new read-print-profile object and copies the values in the slots of profile into the new one.

(xcl:read-print-profile-p object) [Function]

Returns true if the object is a read-print-profile, otherwise false.

(xcl:read-print-profile-readtable profile) [Function]
(xcl:read-print-profile-read-base profile) [Function]
(xcl:read-print-profile-read-suppress profile) [Function]
(xcl:read-print-profile-package profile) [Function]
(xcl:read-print-profile-read-default-float-format profile) [Function]
(xcl:read-print-profile-print-escape profile) [Function]
(xcl:read-print-profile-print-pretty profile) [Function]
(xcl:read-print-profile-print-circle profile) [Function]
(xcl:read-print-profile-print-base profile) [Function]
(xcl:read-print-profile-print-radix profile) [Function]
(xcl:read-print-profile-print-case profile) [Function]
(xcl:read-print-profile-print-gensym profile) [Function]
(xcl:read-print-profile-print-level profile) [Function]
(xcl:read-print-profile-print-length profile) [Function]
(xcl:read-print-profile-print-array profile) [Function]
(xcl:read-print-profile-print-structure profile) [Function]

profile must be a read-print-profile object. Returns the named slot of the read-print-profile. A corresponding setf method is provided for each slot.

(xcl:save-read-print-profile profile) [Function]

Capture bindings of special read & print variables into the profile. Returns profile.

(xcl:with-read-print-profile profile-form &body forms) [Macro]

Binds all the special read & print variables to the values in the profile and executes the body forms as in a let. Returns the value of the last of the forms.

(xcl:restore-read-print-profile profile) [Function]

Restore values of special read & print bindings from profile. Sets current bindings. Returns T.

xcl:*default-read-print-profile* [Variable]

Holds a simple default read-print-profile. When possible programs should default to the current settings of the read-print variables by capturing them with save-read-print-profile instead.

(xcl:find-read-print-profile name) [Function]

Since read-print-profiles enclose readtables and packages, which are availible by name, named read-print-profiles are also availible.

This function will retrieve one of the following "standard" read-print-profiles:

LISP:	LISP readtable and USER package, others nominal.
XCL:	XCL readtable and XCL-USER package, ditto above.
INTERLISP:	INTERLISP readtable and INTERLISP package, ditto.

This function is case insensitive. It returns NIL if a profile by that name is not found.

(xcl:list-all-read-print-profile-names)   [Function]

Returns a list of strings containing the names of all availible read-print-profiles.

WHEREIS

The WHEREIS package extends the function WHEREIS such that, when asked about a given name as a function, WHEREIS will consult not only the commands of files that have been noticed by the file packages, but also one or more hash file data bases that associate function names with file names.

(WHEREIS *NAME TYPE FILES FN*) [Function]

Behaves exactly like the standard WHEREIS function (see the *Interlisp-D Reference Manual*) unless *TYPE*=FNS (or NIL) and *FILES*=T. In this case, WHEREIS will consult, in addition to the files on FILELST, the hash files that are on the value of WHEREIS.HASH. Note: normally the user will just enter the full name of a file onto WHEREIS.HASH, but as WHEREIS begins to use it, it will convert the entry into a cons of the name and the hash file handle.

Many system functions, such as the editors, call WHEREIS with *FILES*=T, so loading this package automatically makes any information contained in the WHEREIS data base files available throughout the system.

Creating a Where Is Data Base

Information may be added to an existing WHEREIS hash file, or by creating new data bases. It is often useful to have a separate data base for large user systems, explicitly calling the following function:

(WHEREISNOTICE *FILEGROUP NEWFLG DATABASEFILE*) [Function]

Inserts the information about all of the functions on the files in *FILEGROUP* into the WHEREIS data base contained on *DATABASEFILE*. If *DATABASEFILE* is NIL, the first entry on WHEREIS.HASH is used.

FILEGROUP may be simply a list of files, in which case each file thereon is handled directly; but it may also be a pattern to be given as a file group argument to DIRECTORY, so * may be used.

If *NEWFLG* is non-NIL, a new version of *DATABASEFILE* will be created containing the data base for the functions specified in *FILEGROUP*. If *NEWFLG* is a number, the hash file will be created with *NEWFLG* entries. Otherwise, it will be created to allow 20,000 entries.

Example:

The following sequence of actions will cause all of the files on the PROJECT directory to be noticed by the WHEREIS package.

```
(WHEREISNOTICE'<PROJECT>*. T '<PROJECT>PROJECTWHEREIS.HASH)
    (push WHEREIS.HASH
    (FINDFILE '<PROJECT>PROJECTWHEREIS.HASH))
```

One of the key components of CLOS in inheritance. The CLOS Browser provides functionality for displaying this structure and for extending it. It also provides functions for displaying and changing the class definitions and method definitions which make up a system written in CLOS.

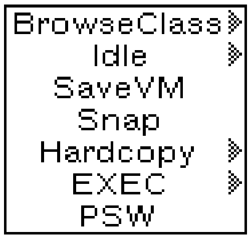
Creating a Browser

A browser can be createded in two ways:

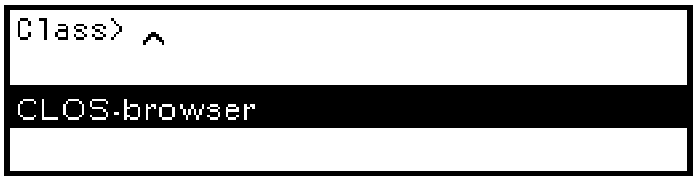
- Via a menu option from the Background Menu
- By calling the function CLOS-BROWSER:BROWSE-CLASS on a class

Creating a browser via the Background Menu

When the CLOS-BROWSER module is loaded, an enty is added to the Background Menu, as shown below:



Selecting the menu item BrowseClass brings up a window, with a prompt for the name of the class to use as the root of the browser as shown below.



Type in the name of the class you wish to browse at the flashing cursor, and the class graph will be drawn in the window.

Creating a browser programmatically

Browsers can also be created by calling the function BROWSE-CLASS:

(BROWSE-CLASS *&OPTIONAL CLASS-NAME-OR-LIST &KEY :WINDOW-OR-TITLE :GOOD-CLASSES :POSITION***)** **[Function]**

This function brings up a browser on the class named or the list of classes named. If a window is supplied for the *:WINDOW-OR-TITLE* argument, then the browser is created in that window, else an appropriately sized window is created. The window is positioned at the *:POSITION* argument or, if not supplied, then the position is set via the mouse. If a text string is supplied for the *:WINDOW-OR-TITLE* argument, then that string is used for the window title, else the string "CLOS-browse" is used. If *:GOOD-CLASSES* is supplied, then only those classes in the list are displayed.

Using the Class browser

Instances of CLOS-BROWSER are operated on through a mouse-based interface.

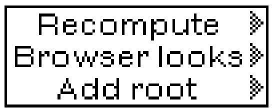
Buttoning on the browser will cause one of the following menus to be popped up:

- One menu appears when the left or middle button is pressed while the mouse is in the title bar. This menu has operations that apply to the browser itself.
- The other menu appears when the middle button is pressed when the mouse is on one of the nodes in the browser.

If the left button is pressed when the mouse is on a node, that node is boxed. This marks the node for some operations.

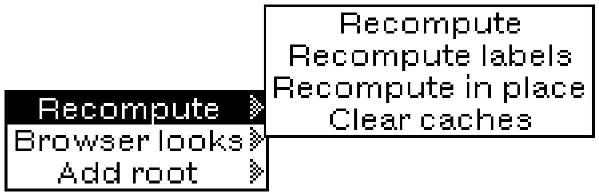
Options in the title bar menu

The following menu appears when you left- or middle-button in the title bar.



Recompute and it’s suboptions

Selecting the Recompute option and dragging the mouse to the right causes the following submenu to appear:

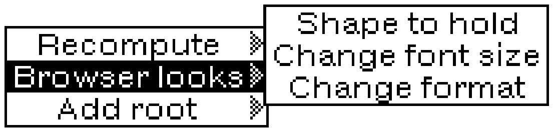


Most of these items change the appearance of the browser, not the contents.

Recompute	Recomputes the browser from the starting objects. It does not recompute the labels for each node if those labels are cached in the Label-Cache slot of the browser.
Recompute Labels	Recompute the browser from the starting objects, including the labels.
Recompute inPlace	Recompute the browser without affecting the scrolled location of the lattice within the window.
Clear caches	Clear the caches of the nodes.

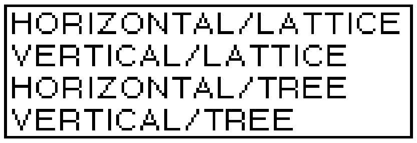
Browser looks and it’s suboptions

Selecting the Browser looks menu item and sliding to the right causes the following submenu to appear:



Selecting one of these options changes the looks of the browser.

Shape to hold	Make the window for the browser just large enough to contain the browser.
Change font size	Causes a menu of alternative font sizes to pop up. Selecting one of these causes the browser to be redrawn with the nodes at that font size.
Change format	Causes the following menu to appear:

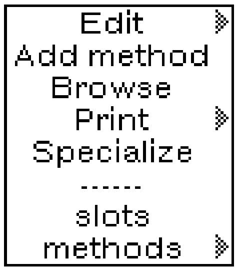


Horizontal/Lattice	Lays out the grapher as an horizontal lattice.

Vertical/Lattice	Lays out the grapher as a vertical lattice.
Horizontal/Tree	Lays out the grapher as a horizontal tree.
Vertical/Tree	lays out the grapher as a vertical tree.

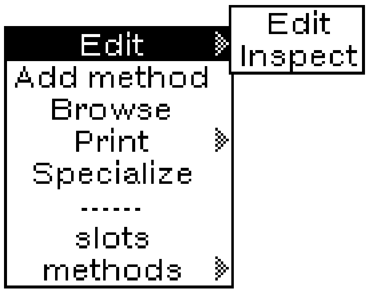
Options in the Middle-button menu

The following menu appears when you middle-button over a node in the graph:



Edit and it’s suboptions

Selecting Edit causes an editor on the class definition to be brought up. Sliding the mouse to the right causes the following menu to appear:



Edit	Edits the class named by the node
Inspect	Inspects the class object named by the node.

Add Method

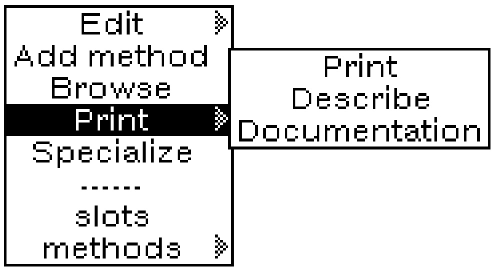
Selecting the Add Method option brings up an editor window with a template for a method to be added to that class. When the editor is done the method is installed for that class and the menu updated.

Browse

Selecting the Browse option causes a browser to be created starting with that class as the root.

Print and it’s suboptions

Selecting Print prints out the class definition. Sliding the mouse to the right causes the following menu to appear:



Print	Print’s the class definition
Describe	Describes the class, listing it’s metaclass, it’s supers classes, it’s subclasses, it’s CPL, and the number of methods specialized to it.
Documentation	Print’s the documentation string for the class

Specialize

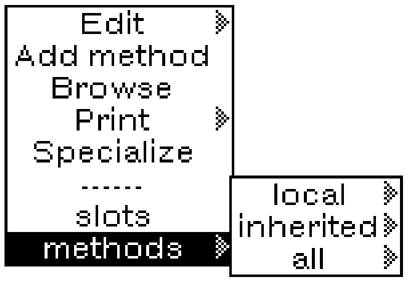
Selecting the Specialize option brings up an editor window with a template for a subclass to be added to that class. When the editor is done the class is installed and the browser updated.

Slots

Selecting the Slots option is the same as selecting the Edit option, it brings up an editor on the class definition.

Methods

The Methods option allows you to edit one of the methods defined for that class. Selecting it and sliding to the right brings up the following sub-menu:



Local	Bring up a menu of the local methods, ie methods directly defined for this class

Inherited	Bring up a menu of the methods this class inherits from it's superclasses.
All	Bring up a menu of all the methods defined for this class, both local and inherited.

Selecting an item with the left button from the resulting menu brings up an editor on that method. If there are multiple methods that apply, a gray triangle appears in the right edge of the menu next to that item. Sliding to the right brings up a menu of method specializers to select the appropriate method.

COLOROBJ

By: Greg Nuyens (Nuyens.pa@Xerox.com)

uses: COLOR

COLOROBJ provides an Interlisp-D imageobject which allows the color to be set on an imagestream. The default image looks like this:

Unknown IMAGEOBJ
GETFN: COLOROBJ.GETFN

The color of the object can be changed by middle-clicking on the object.

(COLOROBJ.CREATE Color) [Function]

Creates a single colorobj. Color is optional and defaults to COLOROBJ.DEFAULT.COLOR.

COLOROBJ.DEFAULT.COLOR [Variable]

Determines the initial color of a colorobj. Default is RED.

HostUp

By: Johannes A. G. M. Koomen
(Koomen.wbst@Xerox or Koomen@CS.Rochester.edu)

This document last edited on December 27, 1988

SUMMARY

This module provides the function HOSTUP? which attempts to find out if a given host is currently available. Contrast this to the system function HOSTNAMEP, which returns T if it has *at any time* successfully opened a connection to given host, whether or not it is currently available.

DETAILS

(HOSTUP? hostname) [Function]

Returns T if and only if the given host is currently responding. No distinction is made between dead and non-existing hosts.

HOSTUP.TIMEOUT [Global variable]

The function HOSTUP? returns NIL if no response is received from the given host within HOSTUP.TIMEOUT milliseconds. Default value is 15,000.

HOSTUP.RETRYCNT [Global variable]

This variable indicates the number of times the function HOSTUP? sends requests to the given host. Each time through the loop the function waits longer by a geometrically increasing amount if time, such that the total time does not exceed HOSTUP.TIMEOUT. Default value is 5. Hence, in the default case, a call to HOSTUP? with a dead host ends up sending a request to the host 5 times, waiting for an answer about 500, 1000. 2000. 4000 and 8000 milliseconds, respectively.

IRIS

By: Nuyens (Nuyens.pa@Xerox.Com)

Requires: COLOR and COLOROBJ

This document last edited on January 31, 1987.

INTRODUCTION

The LispUsers module IRIS is a collection of functions for using an IRIS (Integrated Raster Imaging System) with Interlisp-D. The IRIS is a sophisticated 3-d color graphics workstation produced by Silicon Graphics Incorporated. Much of its imaging system is implemented in special purpose VLSI, making it a very powerful graphics engine.

IRIS provides three separate ways to exploit the IRIS in Interlisp-D. The first is to use the IRIS as an Interlisp-D imagestream. In this capacity the IRIS serves as the output device for a program which needn't even know that it is outputting to the IRIS. Thus standard sytem utilities (such as GRAPHER, SKETCH, and TEDIT) can make use of the IRIS without change. The images produced by these system utilities can then be manipulated with the special abilities of the IRIS (3-d rotation, color updates, double buffering, object definitions, etc.)

The second mode of use for the IRIS is to directly access the IRIS graphics library. In this mode, Interlisp-D provides stubs for the functions in the IRIS graphics library. Thus, Interlisp-D provides the illusion of having the IRIS as a direct device like the standard screen. However, it is actually a separate computer accessible across the 10-MB Ethernet.

The third mode is to use the view controller to interactively view scenes created with either of the two modes. The view controller allows the user to create an object once (interactively or via program) but then view the object many times on the IRIS without recreating the object. This object downloading allows real-time rotation of the object displayed on the IRIS.

This package also implements boot service for the IRIS workstation. The D-machine supplies the boot program to the IRIS over the Ethernet, eliminating the need for a floppy-disk on the IRIS.

This package benefited substantially from comments and improvements by Stu Card, Michel Desmerais and Lennart Lovstrand.

FILE DESCRIPTION

The implementation includes the following files:

LOADIRIS	after loading this file, the user will be prompted to place the IRIS icon. Buttoning this icon will produce a menu including the choice "create loadup panel". Choosing this item will produce a panel which can load the IRIS files. Button the "Standard" entry in the loadup panel, then button on the small SGI logo next to "Load". Prior to being able to run any applications, the IRIS must be running the terminal program. This can be booted from the floppy on the IRIS or from the Lisp workstation. To use the workstation as a bootserver, the boot file for the IRIS terminal program (see msg below) must be copied to {DSK}irisbootfile (or a directory in the list IRISBOOTDIRECTORIES) on the Dandelion that is to be the bootserver. At that point, hit the reset button on the IRIS and type 'n' to the monitor prompt on the iris. That will initiate a net boot from the Dandelion.
IRISSTREAM	The file which contains the Interlisp-D imagestream definition for the IRIS. Thus, the standard Interlisp-D graphics facilities can be applied to an open IRIS imagestream. After this is loaded, images which are processed through the standard hardcopy menu can be sent to the IRIS. "Iris" will appear as a printer when the "to a printer" choice is made from the roll-off menu entry "Hardcopy".
IRISLIB	this file contains the stubs for the functions in the IRIS library. For instance, corresponding to the function circf (found in the IRIS documentation), there is a function IRIS.CIRCF . The arguments are as listed in the IRIS documentation with the addition of the argument SPPstream. This is the SPP connection opened to the IRIS. (If omitted, it defaults to the value of IRISCONN, which is set by OPEN.IRISCONN). Where an argument is a matrix, it is passed to the lisp function as a list of the rows, each row itself a list. When a library fn returns a matrix, the appropriate (usually floatp) matrix is passed in and the elements are set.
IOLIB	this file contains the communication primitives which are used by the fns on IRISLIB.
IRISNET	this file contains the network support for the iris. The IRIS must be on the same network as the D machine (the IRIS doesn't handle routing through the gateway properly). If the IRIS is being bootserved from the Lisp workstation, the boot server software will set IRISNSHOSTNUMBER automatically. (The diagnostic messages can be inhibited by setting \IRIS.VERBOSE to NIL. They are left in because they can be useful when initially getting the IRIS to boot).
IRISVIEW	This file contains the functions supporting the interactive viewing menu, together with the object definition facility it uses.
IRISDEMOFNS	This file contains an example 3-d function for the IRIS. It is a function called TETRA which draws a colored, 3-d recursive tetrahedron.
IRIS.TEDIT	This file.

GETTING STARTED

N.B. All variables and functions in this document are written as in the old Interlisp readtable. To type in the examples in this document, bring up an "Old-Interlisp" Exec. (available from the background menu by rolling off "exec" and then "Interlisp".)

-1) type (FILESLOAD LOADIRIS)

It will ask you to position the following (SGI) icon:

(it can be recreated later by typing (IRIS.CREATE.ICON) at the Interlisp executive).

0) Next type (FILESLOAD IRISSTREAM IRISVIEW)

1) Boot the IRIS

If you have a floppy drive for your IRIS, use it to boot the IRIS GL2 (XNS) Terminal program. (If your IRIS is a workstation, not a terminal, you need to run the WSIRIS program available from SGI).

If you do not etherboot, you must set the variable IRISNSHOSTNUMBER. (see variables section below.)

If you have no floppy, you are going to "etherboot" the IRIS. To begin ensure that IRISBOOTDIRECTORIES is set properly. (see Magic Variables section below).

Press a mouse button in the SGI icon. Choose "Enable bootserver". Now boot the IRIS(by pressing the black button labelled "reset" on the IRIS display). To the prompt on the IRIS, type (on the iris keyboard) N (followed by carriage return). To the eventual prompt "connect to which host?" type < (carriage return). Etherbooting will automatically set the variable IRISNSHOSTNUMBER. Status messages will be printed to the promptwindow. The IRIS will beep when it is finished booting.

2) Open an IRIS stream

Button in the IRIS stream and choose "Create IRISView panel". Confirm the command by clicking left button of the mouse. "Connected" should print on the IRIS screen, and then the screen should turn black and display axes. (**N.B.** If at any time the screen goes completely blank, touch any key on the IRIS keyboard. The IRIS has a dubious screen-saver "feature".)

Modes

The modes of use of the IRIS are as follows:

Interlisp-D display stream

In this mode, the IRIS is acting like a window on the Lisp machine. To see updates as they occur toggle the "Double Buffer" button on the IRISView Controller. It will respond "Single buffering". Then press the "2D-Home" button on the IRIS. To demonstrate this, try the use of commands like

```
(PROGN (DSPCOLOR 'BLUE \IRISSTREAM)

        (PRINTOUT \IRISSTREAM T "Let's here it for " )

     (DSPCOLOR  'RED \IRISSTREAM)

     (PRINTOUT  \IRISSTREAM  "COLOR." T)

     (DSPCOLOR  'BLUE \IRISSTREAM))
```

NOTE that the first time a font family is defined it will download the font after loading the spline font definitions. This will take a few minutes.

```
(DRAWLINE 0 0 400 200 2 NIL  \IRISSTREAM 'RED)

(FORCEOUTPUT IRISCONN)
```

Sinc e the SPP stream buffers, sometimes it will be necessary to (FORCEOUTPUT IRISCONN). (There is a function simply called F that will do this for you.)

These simple instructions indicate the use of the IRIS as an Interlisp Device Independent Graphics (DIG) stream. See the IRM for further examples common to all DIG devices. For interactive graphics, the following sections contain other uses of the IRIS.

Output stream for Sketch and TEdit

TEdit and Sketch can make use of the above display stream facilities. To see an example color document, do the following. (If you don't want to see the sketch in the document, you may omit loading sketch and sketchcolor, since they are slow to load.)

```
(FILESLOAD COLOROBJ SKETCH SKETCHCOLOR)

(TEDIT 'IRIS-EG.TEDIT)
```

To print a TEdit (or sketch) to the IRIS

First time: Press "2D-Home". In the title bar of the window, use the right button to bring up the default window menu. Pick the roll-off item Hardcopy. Rolloff into "hardcopy to printer", then choose "Iris". Choose "yes" to the menu saying "Make this the default printer".

Later times: The IRIS will now be the default printer, so just press "2D-Home"and choose Hardcopy from the default window.

To use Sketch

From the background menu, choose "SKetch". Build a closed polygon (with the entry in the sketch menu). Choose "change" in the sketch menu. Pick a control point of the polygon you just built. Pick "filling color" from the menu that appears. From the menu that appears, pick a filling color.

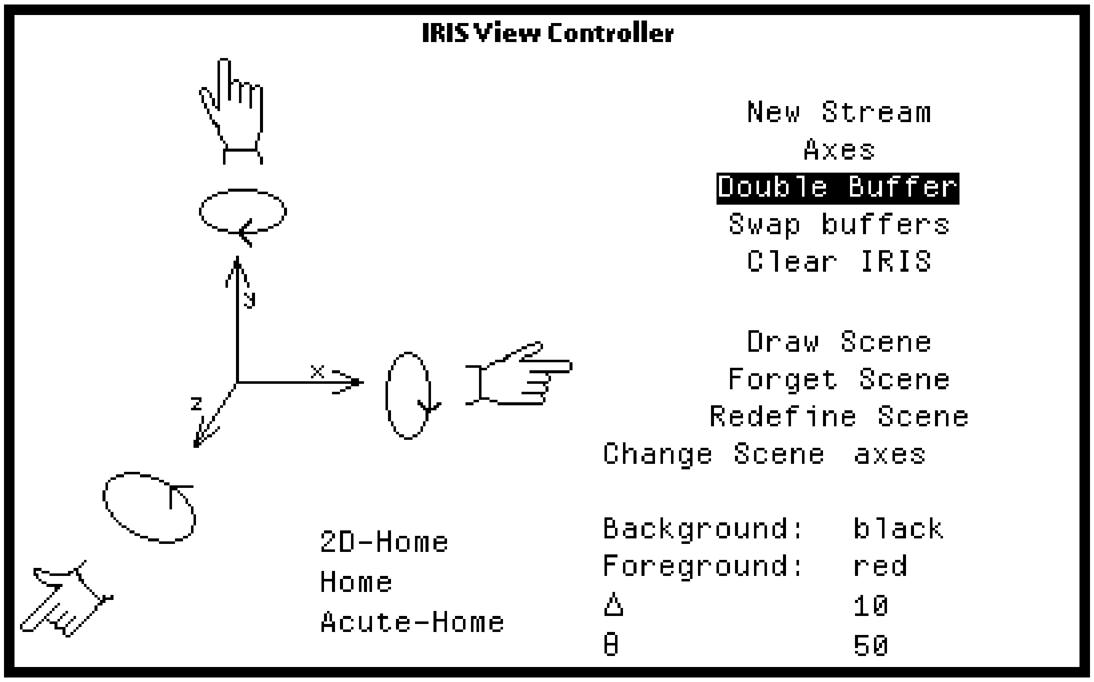
Clearing the IRIS

In the IRIS icon is an entry "clear IRIS". It will return the IRIS to the original state with the white screen, etc.

Defining new colors

At any time the color menu is presented, "new color" may be selected. RGB will provide color sliders and CNS will provide an english description of colors. A window will pop up asking for a name for the color. Type in a name, and that name will appear in the color menu from then on.

Using the IRISVIEW controller



```
Background:   black
Foreground:   red
Δ             10
θ             50
```

N.B. When an IRISview controller is initially opened, the IRIS will be double buffering. This means that any drawing on the iris occurs on the back of two buffers. "Swap Buffers" must be chosen to bring the back buffer to the front. This includes "Clear IRIS" and other drawing commands. Double buffering can be turned off with the "Double Buffer" button, but moving images are much smoother with it turned on.

This window allows you to define objects on the IRIS and control your view of them.

Changing the scene:

First , select "change scene". You will be prompted with the predefined scenes as examples. Choose "axes". After a moment (during with "defining object for axes" is printed), an XYZ axes picture will appear on the screen. Try choosing each of the 3 home buttons to see the effect.

Press the "2d Home" button. (This actually means to move the view such that the IRIS simulates an 1108 screen in size and view). Now change the scene to "SKULL". Again after a delay, a Grateful Dead logo will be displayed.

Defining a scene:

We are going to define a scene that calls a function on the file IRISDEMOFNS so type

(FILESLOAD IRISDEMOFNS)

Button "Change Scene". Choose "New Scene" from the Menu. In response to "Scene Name", type "TETRA". In response to "Form to eval" click in the window the message is printed in and then type

(TETRA) [carriage return]

When the object is displayed, press each of the "Home" buttons to see different home positions. (2D-Home is perhaps best).

Viewing the scene:

Before viewing the scene, choose the "Double Buffer" option. Now hold the left button down on the item marked "Background:" in the view controller. Choose the background color for the object. White is often a good choice. (Note that sliding off the menu at this point will display "none" as the background. This means that each time the scene is drawn will be on top of the previous result. This can give very pleasing, flashy results. Experiment a bit. It's great fun (and you are probably being payed for it!))

Adjusting the view:

Bug the part of the free menu marked ☞ This will move the image in the positive x axis. Similarly with the other hand icons. Holding down the shift key will reverse the direction in the same axis. Bugging the Δentry in the view controller will let you change the "delta" for translation.

Now bug the part of the free menu marked . This will rotate the image in that axis. Holding down the shift key will reverse the direction of rotation.

Bugging the θ entry will let you choose a new "theta" for rotation (in degrees).

Home:

Bugging "home" will return the scene to the original view. It is often a good idea to choose "Home" before changing scenes.

Axes:

Bugging the "axes" entry in the menu will superimpose a set of three dimensional axes on the view for reference purposes. It defines the axes of rotation controlled by the view controller and the directions of translation. (The axes look best against a black background, I think)

Forgetting a Scene:

Reclaims the space inside the IRIS and removes scene names from the scene menu. All but one scene may be deleted.

Defining a scene containing a sketch (or Tedit):

Button "Change Scene". Choose "New Scene" from the Menu. In response to "Scene Name", type "IRIS-EG". In response to "Form to eval", click in the window the message is printed in and then type a carriage return.

The view controller will prompt with "Make object then type RETURN". Now, hardcopy the sketch exactly as above. (periodically the view controller will flash the screen to remind you that you are still making the object). When you are finished, type carriage return as before. The current scene name will be printed in the view controller.

Troubleshooting:

"Iris Terminall SPP not responding" printing in prompt window

This may occasionally print when the Dlion is too busy to service the SPP connection to the IRIS. If it repeats, however, this means that a previous IRIS stream has been lost. Bring up a process status window by choosing PSW in the background, then choose the process (Iris Terminal SPP#2 for instance) and bug "kill" in the PSW. Be sure to kill the correctly numbered process. "Iris Terminal SPP" is not the same as "Iris Terminal SPP#3".

Magic Variables

IRISNSHOSTNUMBER [Variable]

contains the 48 bit etherhostnumber of the IRIS. If you are etherbooting, this will be set automatically. Otherwise, set it to a string like "0#4000.12000.41504#0" as described on page 31.9 of the IRM.

\IRIS.DEBUG [Variable]

Defaults to NIL. If T, when fonts are created, only the first lowercase letters will be defined. This is much faster than loading the whole font.

IRISBOOTDIRECTORIES [Variable]

For users still running with the R1B version of the IRIS terminal program, the file concerned is called xiris. This is the file that should be copied to {core}irisbootfile. However, for users running R1C (also referred to as GL2), it is necessary to obtain an updated version of the terminal program from SGI. (It was named simply "iris" on the tape we received.) This variable must contain the list of directories where the boot files are to be found. (For instance, very fast booting may be obtained by copying the boot files to the {core} device and putting {core} at the front of IRIS.BOOTDIRECTORIES.)

IRIS.VERBOSE [Variable]

Defaults to T. Says whether or not status messages are printed during font creation etc.

\IRISSTREAM [Variable]

contains the IRIS stream that is current.

IRISCONN [Variable]

contains the SPP connection that is current.

KOTOLOGO

By: Masinter (Masinter.PA@Xerox.COM)

Uses: none

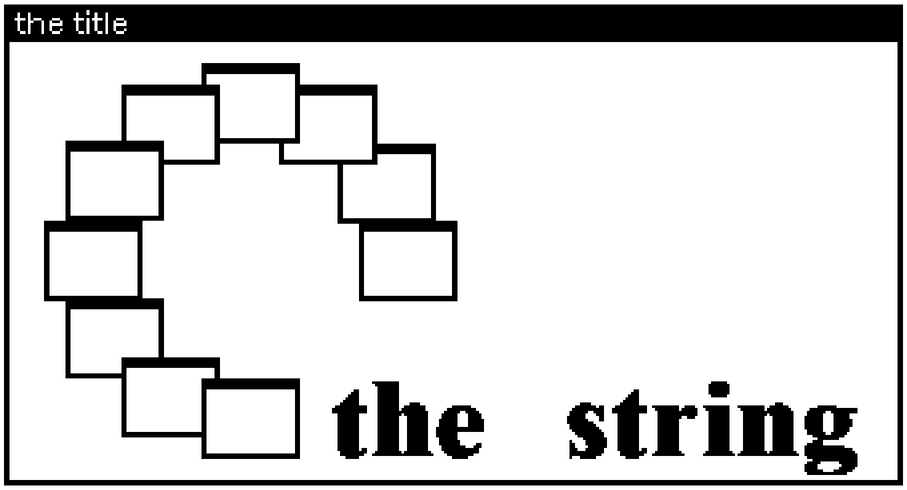
This document last edited on August 17, 1988.

INTRODUCTION

Makes a Koto-style logo window.

(KOTOLOGOW *string where title angledelta*) [Function]

Works like LOGOW did in Koto. Put *string* as the main logo name, with *title* in the window title. *angledelta* is the angle at which the little windows go. *where* is either a position or an old window. For example (KOTOLOGOW "the string" NIL "the title" 30) produces:



LispNerd

By: Maxwell (Maxwell.pa)

DICTCLIENT, DINFO

INTERNAL

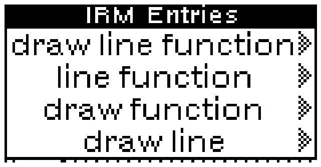
INTRODUCTION

The LispNerd provides a menu-based interface to the Interlisp Reference Manual. The data for the LispNerd is stored on the Dictionary Server, and is accessed via DICTCLIENT. Because of certain licensing agreements we have with Houghton-Miflin, the Dictionary Server should only be used by people within PARC. Hence LispNerd has not been released as a LispUsers package.

HOW TO USE THE LISPNERD

When you load the LISPNERD, it adds a new menu item called "Search IRM" to the background menu. Bugging "Search IRM" causes the LispNerd to prompt the user for keywords in the prompt window. It then produces a menu of items that have at least two of the keywords in their definitions (perhaps with submenus, if there is more than one class of items). Bugging one of these items will cause the LispNerd to fetch the documentation for that item using DInfo.

For example, if you type the input "draw line function", you will get the following menu:



where each of the entries has a sub-menu of the items that have the keywords listed in the entry.

If you type the input "date", you will get a menu all of the items in the Interlisp Reference Manual that have the word "date" in them. Putting parenthesis around items means that the items should all be treated as one keyword for the purposes of sorting them into groups. For instance, the input "(draw draws)(line lines) function" will look for all of the items that have "draw" OR "draws" in their definition.

LispNerd only fetches 50 entries at a time, so sometimes you will see an entry in the menu that says something like ". . . + 103 more". Clicking this item will cause the LispNerd to fetch the next 50 entries. Also, when there is more than one class of entries, sometimes a sub-menu will only list the number of entries it has, and not the entries themselves. To get the entries, click the menu item with the keywords in it, and LispNerd will recompute the menu with just those keywords.

Please send all bug reports to Maxwell.pa.

MATHTONS

By: Tad Hogg (Hogg.pa@Xerox.com)

INTRODUCTION

This file defines the translation array needed to convert from the Press MATH font to corresponding NS characters. This allows documents containing the MATH font to be printed on Interpress printers.

The array \MATHTONSARRAY contains the translations for most of the characters. Some may not be available on particular printers, causing them to appear as black boxes.

MICROTEK

By: Ron Clarke (RClarke.pa@Xerox.COM)

This document last edited October 31, 1988

INTRODUCTION

MICROTEK is an image processing software package that enables you to operate Microtek Models-300 and 300A Intelligent Image Scanners with the Xerox 1108 and 1186 workstations. The Microtek MS-300, 300A, and MSF-300C are high-resolution optical scanners that can convert text, artwork, photographs, etc, into digital form for processing by computer. The digitized images that are output to the computer contain up to 300 black and white dots for every linear inch of the original document. Page size can be as large as 8.5 by 14 inches. Sophisticated firmware in this scanner enables the user to set the scanning area and control brightness, contrast, scaling, shading and other characteristics of the scanned images through simple commands transmitted from the 1108 or 1186. Two basic scanning modes are supported: Line Art mode for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading. Mixed-mode scanning is also available.

With the MICROTEK software package you will be able to: Set the scanner to capture images of all kinds, with desired visual effects, and transmit them to the 1108/1186, save scanned images to disk, floppy or file server for later reloading to recreate images and print scanned images to a Xerox 4045 or 8044 laser printer.

SOFTWARE REQUIRED

MICROTEK.DFASL

MICROTEKPRINT.DFASL (if you have a Xerox 4045 or 8044 laser printer)

DLRS232C.LCOM

EDITBITMAP.LCOM

READNUMBER.LCOM

4045XLPSTREAM.DFASL (if you have a Xerox 4045 laser printer)

FONTS USED

MODERN 10, 12 BOLD

Other useful software for manipulating the scanned image:

Lispuser’s Packages:

ACTIVEREGIONS, ACTIVEREGIONS2, AIREGIONS, FILLREGION

HARDWARE REQUIRED

Xerox 1108 with RS232C port (E-30 upgrade kit). It is also recommended that the 1108 have 3.5 meg of memory and a floating point processor (CPE board) to enable faster scanning and creation of bitmaps.

Xerox 1186. It is also recommended that the 1186 have 3.7 meg of memory.

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner with optional serial port.

CABLE CONFIGURATION

Note that the cable configuration is DIFFERENT for the MSF-300C scanner. Plugging a standard RS232C cable into the MSF-300C DB25 connector may result in damage to the equipment.

RS232C Port (DTE)		**MICROTEK MS-300, MS-300A - DB25 Connector**	
Signal	Pin	Pin	Signal
FGround	1	1	FGround
TD	2	3	RD
RD	3	2	TD
SGround	7	7	Sground

Pins 5, 6, 8 and 20 are jumpered together on the RS232C port end of the cable.

RS232C Port (DTE)		**MICROTEK MSF-300C- DB25 Connector**	
Signal	Pin	Pin	Signal
TD	21	3	RD
RD	9	2	TD
Ground	5,7	7	Ground

DOCUMENTATION REQUIRED

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner Operation Manual

LOADING MICROTEK

Make sure that DIRECTORIES contains the directory where the required software is located. When the file MICROTEK.DFASL is loaded, the item "MicrotekScanner" will be added to the Background menu. If you have a Xerox 4045 or 8044 laser printer load MICROTEKPRINT.DFASL. If you have a Xerox 4045 laser printer load 4045XLPSTREAM.DFASL. Your 4045 laser printer should be connected to the TTY/DCE port.

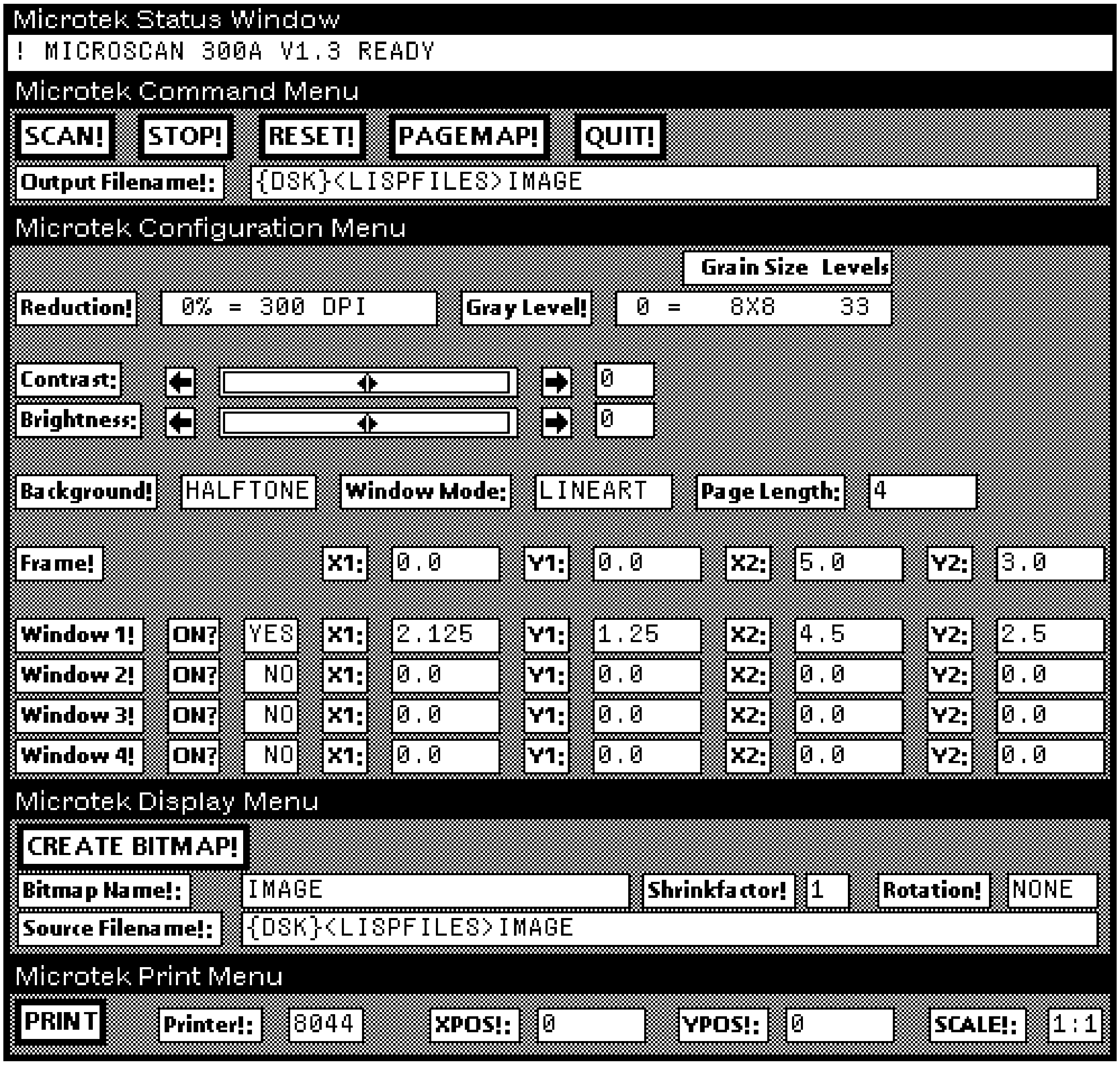
RUNNING MICROTEK

The process of running the Microtek scanner software consists of three phases: Scanner initialization, scanning, and creating a bitmap of the scanned image that can eventually be printed. Each of these are controlled by different menus within the Microtek Scanner Control Window.

SCANNER INITIALIZATION

Set the Microtek scanner so that it is operating at 19200 baud by setting its internal DIP switches (See Microtek Operating Manual for details). Turn on the Microtek scanner. Select "MicrotekScanner" from the background menu and the Microtek Scanner Control window (figure 1) and Microtek Scanner Pagemap window (figure 2) will be created. (Note you may have do a control-E and retry if cursor flashes while trying to create the control window).The scanner pagemap window is used to select the area of the image to be scanned and to select the page length. The scanner control window is used to set all other scanner parameters, start and stop scanning as well as to initiate creation and printing of scanned image bitmaps. After these windows have been created, the RS232 port will be initialized to

19200 baud and an attention command will be sent to the scanner. If all cables are connected properly and the scanner is on, the message "MICROSCAN 300(A) V# is ready" will be displayed in the Microtek Status Window. If the cable is configured incorrectly or the scanner is not on or ready the messsage "Microtek Not Responding ... Check scanner and cable" will appear instead.



Microtek Status Window

```
! MICROSCAN 300A V1.3 READY
```

Microtek Command Menu

SCAN! STOP! RESET! PAGEMAP! QUIT!

Output Filename!: {DSK}<LISPFILES>IMAGE

Microtek Configuration Menu

Grain Size Levels
Reduction!: 0% = 300 DPI
Gray Level!: 0 = 8X8 33
Contrast: 0
Brightness: 0
Background!: HALFTONE
Window Mode: LINEART
Page Length: 4

Frame!			X1:	0.0	Y1:	0.0	X2:	5.0	Y2:	3.0
Window 1!	ON?	YES	X1:	2.125	Y1:	1.25	X2:	4.5	Y2:	2.5
Window 2!	ON?	NO	X1:	0.0	Y1:	0.0	X2:	0.0	Y2:	0.0
Window 3!	ON?	NO	X1:	0.0	Y1:	0.0	X2:	0.0	Y2:	0.0
Window 4!	ON?	NO	X1:	0.0	Y1:	0.0	X2:	0.0	Y2:	0.0

Microtek Display Menu

CREATE BITMAP!

Bitmap Name!: IMAGE
Shrinkfactor!: 1
Rotation!: NONE
Source Filename!: {DSK}<LISPFILES>IMAGE

Microtek Print Menu

PRINT

Printer!: 8044
XPOS!: 0
YPOS!: 0
SCALE!: 1:1

FIGURE 1 - MICROTEK SCANNER CONTROL WINDOW

Before scanning can be initiated, a number of parameters have to be set by the user via the Microtek Command Menu and Microtek Configuration Menu as follows:

Microtek Command Menu:

Output FileName Left buttoning on this item allows you enter the name of the file on disk, floppy or fileserver where the scanned data is to be saved. Be sure to type a carriage return to terminate this entry.

Microtek Configuration Menu:

Reduction Left button on the number next to the item Reduction and a menu will appear. Reduction can be changed from 0%, which corresponds to scanning at 300 dots per inch (DPI) to 75%, which corresponds to 75 DPI.

GrayLevel Left button on the number next to the item GrayLevel and a menu will appear allowing you to choose from a selection of gray levels based on grain size and number of gray levels within that grain size.

Contrast Left button on either the the left or right arrow to either decrease or increase the contrast setting.

Brightness Left button on either the the left or right arrow to either decrease or increase the brightness setting.

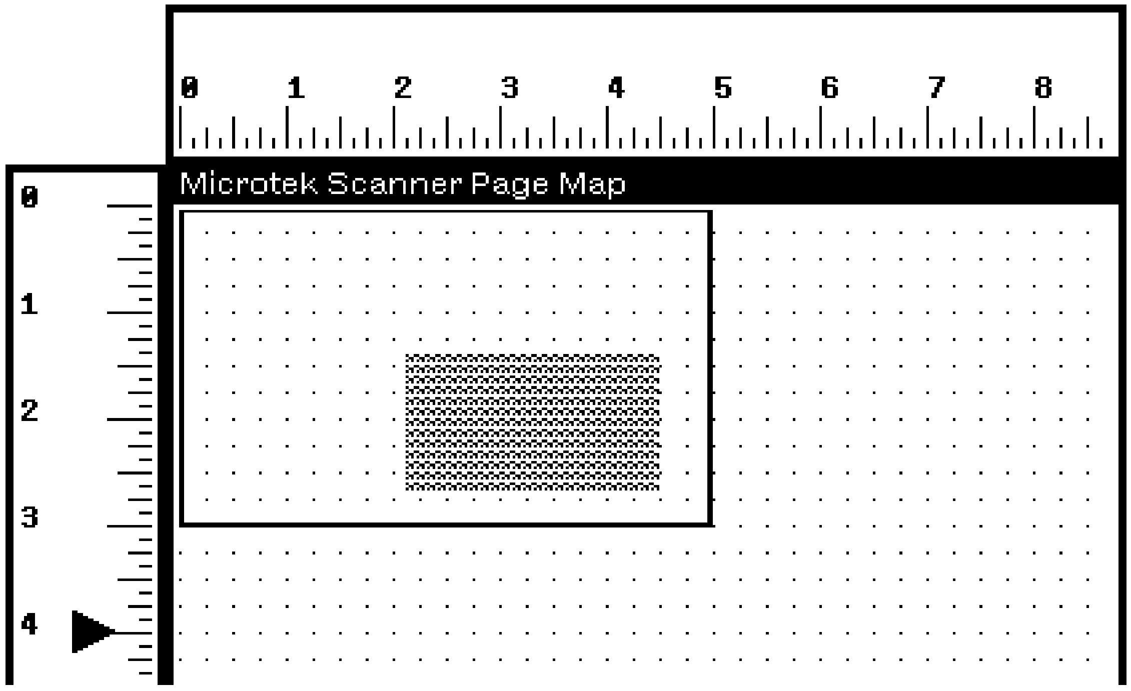


FIGURE 2 - MICROTEK SCANNER PAGEMAP WINDOW

BackGround Select either HALFTONE or LINEART as the primary scanning mode for the image. Line Art mode is for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading.

Pagelength Move the cursor to the vertical ruler of the page map (figure 2). The cursor will change to a right pointing triangle. Position this triangle and left-button to select the pagelength. The page length will also show up in the configuration menu. The page length should be set so that it is longer than the actual page length of the document to be scanned. Otherwise you will get a paper jam message at the completion of scanning. The minimum page length is 3 inches and the maximum page length is 14 inches.

Frame The scanning frame is an area within the document that will be scanned. The maximum scanning frame is 8.5" by 14". Left button on the item Frame and you will be prompted to sweep out an area on the scanner page map to select the primary area to be scanned. The horizontal and vertical rulers and the page map grid dots can be used as a guide in determining the dimensions of the scanning frame. When the the scanning frame has been swept out, a box of the scanned area will be drawn on the page map and the actual X and Y coordinates of the top lefthand corner and lower righthand corner will appear next to corresponding items on the configuration menu (See Figure 2).

Windows 1- 4 Windows are areas within the scanning frame that are scanned in a different mode from the rest of the frame. If LINEART mode is selected as the background, all material in any windows you set will be scanned in Halftone mode, and vice versa.

The method used to set the windows is similar to that used to set the scanning frame except that you first need to specify whether the window is to be selected or not. This is done by left buttoning on the YES/NO indicator next to each window. A menu will pop-up and will allow you select "yes" or "no". After making your selection, left buttoning on the appropriate Window # will cause you to be prompted to sweep out an area within the scanning frame. Each selected window will be displayed and have a unique shade to it (See Figure 2). The only restriction is that the scanning mode must not change more than twice in one 8.5" horizontal scan line. Thus, if two windows lie across the same scan line they must extend to the edges of the page setting area. (Note that material to the left and right of the frame is scanned but not transmitted to the 1108.) You can select different windows for halftone vs lineart mode by switching between backgrounds. The item above WINDOW1 indicates which window mode is selected. An illegal window setting will result in an error message when you attempt to scan. Also note that the windows will be displayed on the scanner pagemap only if there is a "yes" next to the window number.

SCANNING

After the Microtek scanning parameters have been initialized, the document to be scanned should be placed in the scanner top-first with the image to be scanned facing away from the user. Scanning is initiated by left-buttoning SCAN on the Microtek Command Menu. The software first creates a scratch file in {CORE} for storage of the incoming data. It then sends the scanning parameters to the scanner and if all are valid the scanning process starts as indicated by movement of the rollers. You have up to 5 minutes to insert a document before the scanner automatically stops. After scanning has been completed you will be notified in the status window that it is saving the core file to the file specified in Output Filename. It takes approxiamtely 20 minutes to scan an 8.5" x 11" document at 300 DPI.

You may stop the scanning at any time by selecting STOP. The document will be ejected and the scanner reset. You can also explicitly reset the scanner by selecting RESET. This closes the scanner scratch file if it is open, sends a reset command to the scanner and then sends the attention command. If everything is reset properly, you will get the message "MICROSCAN 300(A) V# is ready" in the status window.

CREATING BITMAPS OF SCANNED IMAGES

The Microtek Display Menu is used to create bitmaps from a file that contains scanned data. Select SOURCE FILENAME and enter the name of the file that contains the scanned data. Select BITMAP NAME and enter the name of a variable that you would like the bitmap bound to. **Be sure to type a carraige return to terminate the entry of each of these items.** Left button on the number next to

SHRINKFACTOR and choose a factor by which you want the bitmap shrunk. The default value is 1. Left button on the item next to ROTATION and choose how you want the scanned image to be rotated. The default is "none." After these items have been set, you can then select CREATE BITMAP to start the bitmap creation process. The status window will be updated as it proceeds to create the bitmap and finally, you will be prompted to sweep out a scrollable window to display the bitmap. NOTE: Depending on the size of the bitmap, rotation may take a "very" long time and will look like your machine has frozen...be patient, it will come back. If you desire to save the bitmap(s) on a file you can do the following:

```
(SETQ filenameCOMS '((VARS bitmapname1 bitmapname2 etc))).
(MAKEFILE '{device}<directory>filename)
```

PRINTING BITMAPS OF SCANNED IMAGES TO A XEROX LASER PRINTER

If you have the package MICOTEKPRINT loaded you will have a MicrotekPrint Menu under your display menu (See Figure 1). Select BITMAPNAME on the display menu and enter the name of the bitmap that you would like to print. To select where on the page the bitmap is printed, left button XPOS and YPOS and enter a number. For the 4045 laser printer the values of XPOS can be between 0 - 2550 and YPOS, between 0 - 3300. 1" = 300 print units o 4045 . For an 8044 Interpress laser printer the values of XPOS can be between 0 - 21590 and YPOS, between 0 - 27940. 1" = 2540 Interpress units. The scale that an image is printed at is dependent upon its initial scanned reduction/DPI. You can increase or decrease the scale at which the bitmap is printed by buttoning on the number next to the item SCALE and selecting a scaling factor. On an 8044 Interpress printer a scale of 8:1 will magnify an image by 8 times on printing , 1:1 will print at true size and 1:8 reduce the image by 8 times. Values in between are also available. On a 4045 laser printer only a limited number of scale factor are availble and is dependent upon the original scan reduction as shown in the table below.

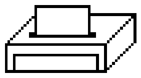
REDUCTION (%)	RESOLUTION (DPI)	SCALES ALLOWED
0	300	1:1, 2:1, 4:1
5	285	1:1, 2:1, 4:1
10	270	1:1, 2:1, 4:1
15	255	1:1, 2:1, 4:1
20	240	1:1, 2:1, 4:1
25	225	1:1, 2:1, 4:1
33	200	1:1, 2:1, 4:1
35	195	1:1, 2:1, 4:1

REDUCTION (%)	RESOLUTION (DPI)	SCALES ALLOWED
40	180	2:1, 1:1, 1:2
45	165	2:1, 1:1, 1:2
50	150	2:1, 1:1, 1:2
55	135	2:1, 1:1, 1:2
60	120	2:1, 1:1, 1:2
67	100	1:1, 1:2, 1:4
70	90	1:1, 1:2, 1:4
75	75	1:1, 1:2 , 1:4

Select PRINT to initiate the printing process. NOTE: The amount of reduction that you will be able to do is dependent upon the number bits that were originally scanned in. If you make the scale too small nothing will be printed out.

OTHER ITEMS AND GENERAL COMMENTS

On the Microtek Command Menu, left buttoning the item PAGEMAP will alternately open and close the scanner pagemap window. Left buttoning on the item QUIT will close the input and output streams to the scanner, shutdown the RS232C port and close the scanner pagemap and control windows. The following icon will be displayed if you shrink the Microtek Scanner Control window.



The Microtek Pagemap window will close when you shrink the Microtek Scanner Control window and has to be expicitely opened when the Microtek Scanner Control window is expanded again. This is done by buttoning on PAGEMAP in the Microtek Command Menu window.

Within Interlisp you normally cannot create bitmaps larger than approximately 2.1 million pixels (about 1400 x 1400). The Microtek scanner software allows you to create bitmaps much larger than this but at the cost of using a correspondingly large amount of virtual memory. If you are near your maximum vmemsize, as determined by comparing (VMEMSIZE) to (VOLUMESIZE ’volumename) , there is a good chance you could crash your system if you create a very large bitmap...caveat emptor. In addition you will not be able to call the function EDITBM to edit bitmaps larger than 2.1 million pixels

The reduction % used to scan the original image is stored on the property list of the atom that the bitmap is bound to. It is saved as the property "Resolution" and is in %. This is used to determine the appropriate values that will make an image 1:1 when printed. If you attempt to print a bitmap to an Interpress printer that was not created by use of the Microtek scanner software you will be prompted to enter a scale explicitely. The following table indicates the 8044 laser printer scale used for scanned images and can be used as a guide when attempting to print bitmaps not created by the Microtek software.

REDUCTION (%)	RESOLUTION (DPI)	SCALE
0	300	.240
5	285	.252
10	270	.266
15	255	.282
20	240	.300
25	225	.320
33	200	.360
35	195	.369
40	180	.400
45	165	.439
50	150	.480
55	135	.533
60	120	.600
67	100	.720
70	90	.800
75	75	.960

Further information about the Microtek scanner can be obtained from:

Microtek Lab Inc
16901 South Western Avenue
Gardena, California 90247
Tel: 213-321-2121, 800-654-4160

NSCOPYFILE

By: Bill van Melle (vanMelle@Xerox.com)

The module NSCOPYFILE modifies COPYFILE so that if both the source and destination files are on NS file servers, the copying is done by an NSFiling-specific copy routine. This routine copies *all* attributes of the source, including non-standard ones, such as those used by Viewpoint. Thus, you can safely copy Viewpoint files from inside Lisp without losing information. In addition, if the copy is from an NS file server to itself, the copy is performed by the server itself, which is considerably faster than shipping the file over the Ethernet.

In addition, you can also copy entire directories, by specifying directory names as the source and destination, e.g.,

```
        (COPYFILE "{FS:}<Carstairs>Lisp>" "{FS:}<Calvin>Lisp>Current>")
```

The destination directory must not already exist, since this operation creates an entirely new directory, whose contents are a copy of all the source directory's offspring, to all levels. If the destination directory happens to exist but has no children, it is considered vestigial and is quietly deleted first (Lisp usually suppresses such directories when performing a directory enumeration).

You can also use RENAMEFILE in the same manner to either rename a directory, or to move an entire directory and its descendents to a new node in the file server's hierarchy, or to a new server altogether. You must, of course, have access rights to delete the source directory *and* all its children, and the destination must be on an NS file server.

A word about protection: when a file is copied or moved, the new file is given "defaulted" access rights, i.e., its protection is set as specified by its new parent (sub)directory, just as if you had created the file afresh by any other means. Thus, if the original file happened to have its own explicit protection, that protection is ignored. When copying or moving an entire directory, only the top-level directory receives default protection, so if any individual descendent file had non-default protection, that protection is copied verbatim. This can lead to confusion—you may want to use the NSPROTECTION module to change the new directory's descendents to default protection. See the documentation of NSPROTECTION for more discussion about protection issues. Note that if a file/directory is renamed within the same parent directory, the operation is considered merely "changing the name", and its protection is left unchanged.

Note: If you are using the FILEWATCH module, be aware that files being copied between NS servers do not appear (because the files are not opened by the normal Lisp file system).

POSTSCRIPT

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

INTRODUCTION

The PostScript package defines a set of imageops for printers which understand the PostScript page description language by Adobe. At Beckman we have successfully used TEdit, Sketch, LISTFILES, and HARDCOPYW to an Apple LaserWriter and an AST TurboLaser PS. The PostScript imagestream driver installs itself when it is loaded. All symbols in the PostScript driver are located in the INTERLISP: package.

VARIABLES

POSTSCRIPT.FONT.ALIST [InitVariable]

POSTSCRIPT.FONT.ALIST is an ALIST mapping Xerox Lisp font names into the root names of PostScript font files. It is also used for font family coercions. The default value should be acceptable for any of the fonts which are built into the Apple Laserwriter.

POSTSCRIPTFONTDIRECTORIES [InitVariable]

POSTSCRIPTFONTDIRECTORIES is the list of directories where the PostScript .PSCFONT font files can be found. The default value is: ("{DSK}<LISPFILES>FONTS>PSC>").

\POSTSCRIPT.SHORTEDGE.SHIFT [InitVariable]

\POSTSCRIPT.SHORTEDGE.SHIFT is the distance (in points) to shift the image perpendicular to the short edge of the paper. A positive value gives a shift upward in portrait mode, and to the right in landscape mode. The default value is: 0.

\POSTSCRIPT.LONGEDGE.SHIFT [InitVariable]

\POSTSCRIPT.LONGEDGE.SHIFT is the corresponding variable for shifts perpendicular to the long edge of the paper. A positive value here gives a shift to the right in portrait mode and downward in landscape mode. The default value is: 0.

\POSTSCRIPT.SHORTEDGE.PTS [InitVariable]

\POSTSCRIPT.SHORTEDGE.PTS indicates the printable region of the page (in points) along the short edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 576 (= 8 inches).

\POSTSCRIPT.LONGEDGE.PTS [InitVariable]

\POSTSCRIPT.LONGEDGE.PTS indicates the printable region of the page (in points) along the long edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 786.24 (= 10.92 inches).

HINT

The AST TurboLaser PS has an imageable area on the page which is a different size than that of the Apple LaserWriter. The values of \POSTSCRIPT.SHORTEDGE.PTS and \POSTSCRIPT.LONGEDGE.PTS for the AST are 575.76 and 767.76, respectively.

\POSTSCRIPT.MAX.WILD.FONTSIZE [InitVariable]

\POSTSCRIPT.MAX.WILD.FONTSIZE indicates the maximum point size that should be returned from FONTSAVAILABLE when the SIZE argument is wild (i.e. *). All integer pointsizes from 1 to \POSTSCRIPT.MAX.WILD.FONTSIZE will be indicated as available. The default value is: 72.

POSTSCRIPT.PREFER.LANDSCAPE [InitVariable]

POSTSCRIPT.PREFER.LANDSCAPE indicates if the OPENIMAGESTREAM method should default the orientation of output files to LANDSCAPE. It can have one of three values: NIL, T, or ASK. NIL means prefer portrait orientation output, T means prefer landscape, and ASK says to bring up a menu to ask the preferred orientation if it wasn't explicitly indicated in the OPENIMAGESTREAM call (with the ROTATION option). The default value is: NIL.

POSTSCRIPT.TEXTFILE.LANDSCAPE [InitVariable]

POSTSCRIPT.TEXTFILE.LANDSCAPE indicates if the printing of TEXT files (e.g. LISTFILES, ...) should force the orientation of output files to LANDSCAPE. The default value is: NIL.

POSTSCRIPT.BITMAP.SCALE [InitVariable]

POSTSCRIPT.BITMAP.SCALE specifies an independent scale factor for display of bitmap images (e.g. window hardcopies). Values less than 1 will reduce the image size. (I.e. a value of 0.5 will give a half size bitmap image.) The position of the scaled bitmap will still have the SAME lower-left corner (i.e. the scaled bitmap is not centered in the region of the full size bitmap image). The default value is: 1.

HINT

Setting POSTSCRIPT.BITMAP.SCALE to 0.96, instead of 1, will give cleaner BITMAP images on a 300 dpi printer. (This corrects for the 72 ppi imagestream *vs*. the 75 dpi printer, using 4x4 device dots per bitmap pixel.) Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

POSTSCRIPT.TEXTURE.SCALE [InitVariable]

POSTSCRIPT.TEXTURE.SCALE specifies an independent scale for the display of bitmap textures. The value represents the number of device space units per texture unit (bitmap bit). The default value is 4, which represents each bit of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen (for 300 dpi output devices, such as the Apple Laserwriter).

The PostScript package extends the allowed representations of a texture, beyond 16-bit FIXP and 16x16 bitmap, to ANY square bitmap. (If the bitmap is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bitmap textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter (in the printer controller), and can cause limitcheck errors when actually printing.

Anywhere that a texture or color can be used on an imagestream or in the specification of a BRUSH, you can instead give a FLOATP between 0.0 and 1.0 (inclusive) to represent a PostScript halftone gray shade. (0.0 is black and 1.0 is white. Specifically, the value sets the brightness of the shade.) The value you specify will not be range checked, and will be passed directly through to the PostScript setgray operator. (E.g. you can pass 0.33 as the color to DRAWLINE to get a dark gray line with approximately 67% of the pixels in the line black.)

POSTSCRIPT.IMAGESIZEFACTOR [InitVariable]

POSTSCRIPT.IMAGESIZEFACTOR specifies an independent factor to change the overall size of the printed image. This re-sizing affects the entire printed output (specifically, it superimposes its effects upon those of POSTSCRIPT.BITMAP.SCALE and POSTSCRIPT.TEXTURE.SCALE). Values greater than 1 enlarge the printed image, and values less than 1 reduce it. An invalid POSTSCRIPT.IMAGESIZEFACTOR (i.e. not a positive, non-zero number) will use a value of 1. The BITMAPSCALE function for the POSTSCRIPT printer type does NOT consider the POSTSCRIPT.IMAGESIZEFACTOR when determining the scale factor for a bitmap.

MISCELLANEOUS

The SCALE of a PostScript imagestream is 100. This is to allow enough resolution in the width information for fonts to enable TEdit to correctly fill and justify text.

The first time any PostScript imagestream is created (even if only to hardcopy a bitmap or window) the DEFAULTFONT is instantiated (unless a FONTS option was given to the OPENIMAGESTREAM, in which case the initial font for the imagestream will be set to that font, or to the CAR if a list).

The PostScript imagestream method for FILLPOLYGON uses the global variable FILL.WRULE as the default value for the WINDINGNUMBER argument. (This is the same variable which is used by the DISPLAY imagestream method for FILLPOLYGON.)

The PostScript imagestream method for OPENIMAGESTREAM (and, therefore, SEND.FILE.TO.PRINTER), supports an IMAGESIZEFACTOR option to change the size of the printed image. The IMAGESIZEFACTOR re-sizing is combined with the POSTSCRIPT.IMAGESIZEFACTOR to produce an overall re-sizing of the printed image. A HEADING option is also supported to give a running header on each page of output. The value of the HEADING option is printed at the top left of the page, followed by "Page " and the appropriate page number. They are printed in the

DEFAULTFONT (unless a FONTS option was given to the OPENIMAGESTREAM, in which case it will be that font, or to the CAR if a list).

The PostScript package is contained in the files: POSTSCRIPT.LCOM & PS-SEND.LCOM, with the source in the files: POSTSCRIPT & PS-SEND. The module PS-SEND.LCOM is required and will be loaded automatically when POSTSCRIPT.LCOM is loaded. It contains the function which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple. System record declarations required to compile POSTSCRIPT can be found in EXPORTS.ALL.

I'm pretty sure that the output generated by the PostScript imageops fully conforms to the Adobe Systems Document Structuring Conventions, Version 2.0, January 31, 1987.

Including Other PostScript Operations

If you wish to insert your own specific PostScript operations into a PostScript imagestream, you can do so with the following functions:

(POSTSCRIPT.OUTSTR *STREAM STRING*) [Function]

POSTSCRIPT.OUTSTR outputs a string or value to the imagestream. *STREAM* must be an open PostScript imagestream. *STRING* is the value to output (STRINGP and LITATOM are most efficient, but any value can be output (its PRIN1 pname is used)).

(POSTSCRIPT.PUTCOMMAND *STREAM STRING ... STRING*) [NoSpread Function]

POSTSCRIPT.PUTCOMMAND is more general for sequences of commands and values. It calls POSTSCRIPT.OUTSTR repeatedly to output each of the *STRING* arguments to *STREAM*.

(\POSTSCRIPT.OUTCHARFN *STREAM CHAR*) [Function]

\POSTSCRIPT.OUTCHARFN is used to output the characters forming the text of a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream to output to, and *CHAR* is the CHARCODE of the character to output. The ***/*** (slash), **(** and **)** (parenthesis) characters will be quoted with ***/***, and characters with ASCII values less than 32 (space) or greater than 126 (tilde) will be output as **/nnn** (in octal). \POSTSCRIPT.OUTCHARFN will output the **(** character to open the string, if necessary. Use POSTSCRIPT.CLOSESTRING (below) to close the string.

(POSTSCRIPT.CLOSESTRING *STREAM*) [Function]

POSTSCRIPT.CLOSESTRING closes a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream. It is important to use POSTSCRIPT.CLOSESTRING to output the **)** character to close the string, because it also clears the stream state flag that indicates that a string is in progress (otherwise, the next POSTSCRIPT.PUTCOMMAND would output the commands to close the string and show it).

Warning

Do not attempt to create a PostScript font larger than about 600 points, as much of Interlisp's font information is stored in SMALLP integers, and too large a font would overflow the font's height, or the width for any of the wider characters. (I know that 600 points is a ridiculously large limit (about 8.3 inches), but I thought I'd better mention it, or someone might try it!)

Changes from the Lyric Release

The Medley release of this PostScript imagestream driver changed the default value of POSTSCRIPT.TEXTFILE.LANDSCAPE from T to NIL. It also added the support for the HEADING option.

Known Problems/Limitations

The output generated for a PostScript imagestream is rather brute force. It isn't particularly careful to generate the smallest output file for a given sequence of operations. Specifically, it often generates extra end-of-lines between PostScript operator sequences (this has no effect on the printed output, only on the file size).

Using BITMAPs or Functions as BRUSH arguments to the curve drawing functions is not supported, nor is using a non-ROUND BRUSH with DRAWCIRCLE or DRAWELLIPSE.

There is no support for NS character sets other than 0, and there is no translation of the character code values from NS encoding to PostScript encoding.

There is no support for color.

\POSTSCRIPT.OUTCHARFN is pretty wimpy in its handling of TAB characters. It just outputs 8 SPACEs for the TAB.

I haven't yet documented how to build the .PSCFONT files for any new fonts that become available, I'll do that eventually.

PS-PATCH

By: Will Snow (snow.envos@xerox.com)

Requires: POSTSCRIPT

This module fixes some bugs in software other than postscript. If you are going to load sketch, load it BEFORE you load this patch. If you load sketch after loading this patch, evaluate the following form in an Interlisp Executive:

```
(MOVD (QUOTE NEW-SK-PICK-FONT) (QUOTE SK.PICK.FONT) NIL T)
```

This will make the printing of sketches with text in them reasonable.

PS-RS232

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, PS-SEND, DLRS232C

The module PS-RS232 defines a printing host named PS-RS232 which sends PostScript output to a printer over the {RS232} port of the 11xx. It also puts a function onto AROUNDEXITFNS which reinitializes the {RS232} port after returning from LOGOUT. The BaudRate and other parameters of the {RS232} port are controlled by the following variables.

VARIABLES

PS-RS232-BAUD [InitVariable]

This is the BaudRate for the {RS232} port output stream. Defaults to: 9600.

PS-RS232-DATABITS [InitVariable]

This is the BitsPerSerialChar for the {RS232} port output stream. Defaults to: 8.

PS-RS232-PARITY [InitVariable]

This is the Parity for the {RS232} port output stream. Defaults to: NONE.

PS-RS232-STOPBITS [InitVariable]

This is the NoOfStopBits for the {RS232} port output stream. Defaults to: 1.

PS-RS232-FLOWCONTROL [InitVariable]

This is the FlowControl for the {RS232} port output stream. Defaults to: XOnXOff.

UPCSTATS

By: Larry Masinter (Masinter.pa@Xerox.com)

INTERNAL

This document last edited on 11 October 84

UPCSTATS is for gathering statistics about where Dorado microcode is spending its time. (It only works on Dorados.) It samples the microcode's PC while running something, and then plots it in a histogram. It really doesn't help much unless you are familiar with the organization of the Dorado Interlisp-D microcode, and want to analyze it.

(UPCSTATS *form dolistflg*) [Function]

will EVAL *form* while gathering statistics, and then print out a histogram. If *dolistflg* is NIL, the output will go to the current output file (NIL). The first time you run UPCSTATS, it will ask you for the name of a ".MB" file. This is a Dorado Microcode Binary, and you need to get the version of "DoradoLisp.MB" that corresponds to the "DoradoLispMC.EB" that is on your local disk. Normally this is on the "Basics" release subdirectory.

Once you've done a UPCSTATS, you can print the output again, merely by calling (PLOTPCS).

SAMPLE OUTPUT:

```
Microcode PC Sample:  Each * = 11 count, or      .03%
AEMUNOTREADY                                                                   |***************
(.5154639)
NOSKIP                                                                                       |*
(.5500377)
EFFADRPCREL+1     |*
                                                               +2                            |*
(.6254715)
LDA3                                                                                         |*
(.6569022)
LDA23             |**
                            +1                                                               |*
(.7669097)
LDAIX                                                                                     |****
(.9146342)
LDAX                                                                                      |****
(1.059215)
STACKGETSMD                                                                       |************
(1.480387)
```

```
STA2                 |*
```

XORcursorPatch

By: Christopher Lane (Lane@sumex-aim.stanford.edu)

This document last edited on July 22, 1987

INTRODUCTION

This module allows the 1186/Daybreak (only) users to twiddle the hardware bits so that they can have an inverting cursor (white on black and black on white instead of black on everything) and provides a patch to keep the system from undoing the effect when calling VIDEOCOLOR to reset the screen.

USE

(DOVE.XOR.CURSOR *FLG*) [Function]

The argument *FLG*, if T, will switch to the inverting cursor mode. If *FLG* is NIL it will switch back to normal mode. If *FLG* is a number between 0 and 15 then it is used as the 'mix-in rule' and has an effect according to the table below.

Mix-in rules

	Screen	Source	Cursor Mode
0	All Black	None	
1	Normal	Normal	Paint
2	Normal	Inverted	Paint
3	Normal	None	mode.
4	Inverted	Normal	Erase
5	All Black	Inverted	Paint
6	Inverted	Normal	Invert
7	Normal	Inverted	Erase
8	Inverted	Inverted	Erase
9	Normal	Normal	Invert
10	All White	Inverted	Paint
11	Normal	Normal	Erase
12	Inverted	None	
13	Inverted	Inverted	Paint
14	Inverted	Normal	Paint
15	All White	None	

This table is relative to the normal mode of the display (1), normal screen, normal cursor in paint mode. The inverted display, (VIDEOCOLOR T), would be 13, inverted screen and inverted cursor in paint mode. There is probably a more precise or logical way to notate these modes, but this should give you a rough idea ofwhat's available

Note

The function is set up such that when trying different modes, you must do a (DOVE.XOR.CURSOR) (no argument) between calls.

No warranty expressed or implied, but we have been using it locally without problem (at least as far as I know). Enjoy.

The following notes explain my (woz) idea of what the edit-interface should be responsible for, and what constitutes and "editor".

An editor is a symbol who's function takes the args (structure props options) and starts an interactive editor on the structure. PROPS is a property list, and OPTIONS is a list of keywords, both affecting the behavior of the editor. The property :completion-fn specifies the function to be called when the user completes the edit. The completion-fn will be called with the arguments (structure props options changed?), where structure is the edited structure (even on abort, so that the edit interface could implement "undo abort"), props and options are as specified on the call to the editor and will be used to figure out what to do with this completion, and changed? is NIL (no changes made), T (changes made), or :ABORT (user wants to abort changes). It is then up to the completion function to do the right thing with the result of the edit.

The goal is that the editor doesn't know anything about who started it, where the structure came from, or what to do with it when it's done. And the edit-interface doesn't know anything about the editor's data structures.

In the case of open edit sessions (open or shrunk):
  If the editor is told to start an edit, the editor must look for one already existing that matches (this can't be the responsibility of the edit interface, because it doesn't know about existence of open edits). the editor should restart the existing edit, processing any new props or options appropriately.

The markaschanged issue:
  Since the editor knows it may have open edits, it needs to provide a hook for when the world gets changed underneath the editor. in this world this means markaschanged. in this case the editor should try to restart itself with the new structure. in other words, sedit::markaschangedfn should call edit-definition to start a new edit. the editor will then notice it has a matching edit open and restart itself with the new info.
  Thismodel is complicated by the fact that markaschanged gets called as a result of completion. Presently *ingore-changes-on-completion* controls the behavior in this case. Ideally, the editor would just say "i know it got changed, i just changed it!", and it would ignore the call.

The new version of SEdit (1/25/91) is very close to this definition, with the following exceptions, which can be fixed upon implementation of this edit inteface design:
- the completion-fn is called with (context structure changed?) since all of SEDITE's completion-fns expect these args. this should be fixed in handle-completion.
- in the abort case, undo is run until there are no more changes to undo, since sedit is sometimes handed structures "in place", destructive edits need to be undone, and thus the completion-fn never gets to see the edits. this can be fixed in the function complete.
-

To make this editor active, call (il:editmode editor-name), where editor-name is the symbol defined above. The function xcl::edit will then start the active editor.

xcl::edit-expression provides an example of starting the editor on an unnamed structure, where eqness is expected upon completion.

xcl::edit-definition provides a replacement for il:editdef, if il:getdef and il:putdef work correctly on all types.

:dontwait
  the editor (sedit) should not wait. don't wait is part of the editinterface features. if its going to wait, the editinterface should create the event and bind it in the completionfn to the notified by the completionfn.

need published edit-expression command. same as ed, but takes and expression, waits for once-only completion, and returns the expression. eg for FIX

sedit should provide a published interface to creating gaps, for use by make-prototype-defn.

edit interface may need a way to ask editor if it has an open edit for name/type

deal with editdates

an editor used to install itelf by replacing EDITL and EDITE, then editmode was created to look for Definition-for-EDITE property on the symbol returned by editmode. now an editor is defined as a function which takes (expr props options) and implements at least the props :completion-fn and :root-changed-fn. EDITMODE then returns the name of this function so the current editor can be applied.

HOW TO MAKE SYSOUT ON SUN

Osamu Nakamura:KSPA:Fuji Xerox
February 20, 1990

SUN [redacted]SYSOUT [redacted].

[redacted]SYSOUT[redacted]Dorado[redacted].
Dorado[redacted]FX[redacted]SYSOUT[redacted],
[redacted], Venue [redacted]Dorado[redacted]SUN [redacted]
SYSOUT[redacted].

[redacted], Venue[redacted]John D. Sybalsky [redacted],
SUN[redacted]SYSOUT[redacted].

[redacted]

1.MAKEINIT/LOADUP [redacted]
[redacted].
[redacted] [redacted]~/SUNLOADUP [redacted].
• runloadup.
• FILESETS.
• INIT.MAKEINIT
• XREM.CM;1
• LOADUP-REM.CM;
• LOADUP.LISP;

2. Medley [redacted]
- [redacted]([redacted] LDE)
- INIT[redacted]([redacted] INITLDE)

MAKEINIT [redacted]INIT.DLINIT[redacted]

3. LISP.SYSOUT [redacted]
(Lispcore/sources [redacted].)

4. Medley [redacted](makefile [redacted])
([redacted]INITLDE[redacted])

[redacted]

1. INITLDE[redacted]

INITLDE[redacted]makefile [redacted] makeinitlde [redacted],
INITLDE[redacted].
INITLDE[redacted], [redacted],
[redacted] $YOURWORKDIR/init.$ARCH/ [redacted].

[redacted].

```
prompt% mkdir $YOURWORKDIR/init.$ARCH
    -- $ARCH [redacted] sun4 [redacted]sparc , sun3 [redacted] mc68020 [redacted].
prompt% cd $YOURWORKDIR/bin
rompt% makeinitlde -e
```

-- [redacted] "-DINIT" [redacted] [redacted].

INITLDE '$YOURWORKDIR/init.$ARCH/ lde' [redacted].

[redacted]: [redacted]INITLDE[redacted] [redacted]

cg3,cg6 [redacted].
cg3,cg6 [redacted]INITLDE[redacted],
makeinitlde [redacted]OPTFLAGS [redacted] -DDISPLAYBUFFER [redacted] [redacted] $YOURWORKDIR/init.$ARCH/ [redacted] [redacted],
makeinitlde -e [redacted].

2. MAKEINIT/LOADUP [redacted] [redacted]

MAKEINIT/LOADUP[redacted]Venue [redacted] [redacted].
[redacted]Venue [redacted] [redacted].
[redacted].

FILE: runloadup

```
set LDE              = ¤HOME/maiko/sunos4.sparc/lde
set LDEPATH          = ¤HOME/maiko/sunos4.sparc
set INITLDE          = ¤HOME/maiko/init.sparc/lde
set INITLDEPATH      = ¤HOME/maiko/init.sparc
set FULL_SYSOUT      = /usr/local/sysouts/FULL.SYSOUT
set FIRST_REM_CM     = ¤HOME/SUNLOADUP/XREM.CM
set SECOND_REM_CM    = ¤HOME/SUNLOADUP/LOADUP-REM.CM
```

FILE: INIT.MAKEINIT

```
DIRECTORIES [redacted]
FILESETS [redacted]LOAD [redacted]

FASTINIT.DFASL [redacted]LOAD [redacted]
MAKEINIT[redacted]4[redacted]
DLFIXINIT[redacted]3[redacted]
```

FILE: LOADUP-REM.CM

```
LOADUP.LISP[redacted]LOAD [redacted]
```

MAKEINIT/LOADUP [redacted]

1. [redacted]

MAKEINIT/LOADUP [redacted].

- ~/I-NEW, ~/I-NEW.LCOM
- ~/INIT.SYSOUT

- ~/INIT.DLINIT
- ~/lisp.virtualmem
- ~/REM.CM
- ~/SUNLOADUP/LOADUP.LOG

2. [redacted]

[redacted] SUNLOADUP [redacted] runloadup [redacted].

```
prompt% cd ~/SUNLOADUP
prompt% runloadup
```

[redacted] SUNLOADUP [redacted] LISP.SYSOUT [redacted].

SYSOUT [redacted] [redacted]

LOADUP.LISP [redacted]FILESETS [redacted]
SYSOUT [redacted].

Ethernet [redacted]SYSOUT

LISP.SYSOUT [redacted]Ethernet [redacted]SYSOUT [redacted]
[redacted].

[redacted]:

[1] FILESETS[redacted]
1LISPSET [redacted] LLETHER [redacted]

[2] LOADUP.LISP[redacted]
DPUPFTP LLNS TRSERVER SPP COURIER NSPRINT CLEARINGHOUSE NSFILING INTERPRESS [redacted]

[3] XREM.CM[redacted]
[redacted]2[redacted]
(MOVD (QUOTE \ETHEREVENTFN) (QUOTE \ETHEREVENTFN-))
(MOVD (QUOTE NILL) (QUOTE \ETHEREVENTFN))

[4] runloadup [redacted]

[redacted]: SYSOUT [redacted].

D-Machine [redacted]SYSOUT

LIS.SYSOUT [redacted]D-Machine [redacted](D-Machine [redacted]

[redacted])[redacted]SYSOUT [redacted].

[redacted]:

[1] LOADUP.LISP[redacted]
DPUPFT DISKDLION DOVEINPUTOUTPUT DOVEDISK DOVEDISPLAY DOVEMISC DOVEETHER DOVEFLOPPY DSKDISPLAY FLOPPY [redacted]

[2] runloadup [redacted]

[redacted]: SYSOUT [redacted].

[redacted]SYSOUT

LISP.SYSOUT [redacted]Interlisp BYTE Compiler, XCL compiler [redacted]SYSOUT [redacted].

[redacted]:

[1] LOADUP.LISP[redacted]
DLAP BYTECOMPILER COMPILE FASDUMP XCL-COMPILER
DPUPFT DISKDLION DOVEINPUTOUTPUT DOVEDISK
DOVEDISPLAY DOVEMISC DOVEETHER DOVEFLOPPY
DSKDISPLAY FLOPPY [redacted]

[2] runloadup [redacted]

[redacted]: [redacted]
Undefined function SPECVARS [redacted] Break [redacted].
SPECVARS [redacted] [redacted]COMPILE [redacted].

TCP-IP

The Transport Control Protocol - Internet Protocol (TCP-IP) family of networking protocols was developed under the auspices of the Department of Defense to standardize communication mechanisms within Department of Defense networks such as the ARPANET.

The protocols are documented in a collection of working papers known as Requests for Comments (RFCs). Appropriate RFC numbers appear throughout this document as new protocols are introduced.

Requirements

TCP-IP has both hardware and software requirements.

Hardware

- Ethernet
- Cooperating host (yours or theirs)
- 110X/118X with an Ethernet controller (usually co-resident on an otherwise inhabited module)
- XCVR interface cable
- XCVR installed on an Ethernet with a logical (direct or internet) connection to the cooperating host.

Software

You need the files enumerated in the section titled "Interlisp Files." Files loaded by the high-level modules TCPFTP, TCPFTPSRV, TCPCHAT, and TCPTFTP automatically load their dependencies. If you load files from floppy, you must load their dependencies first:

File	Dependencies
TCP	TCPLLIP
TCPCHAT	TCP, CHAT
TCPCONFIG	None
TCPDEBUG	TCP
TCPDOMAIN	TCPUDP
TCPFTP	TCPNAMES, TCP
TCPFTPSRV	TCPFTP
TCPHTE	None
TCPLLAR	None
TCPLLICMP	None
TCPLLIP	TCPHTE, TCPLLICMP, TCPLLAR
TCPNAMES	None
TCPTFTP	TCPUDP
TCPUDP	TCPLLIP

User Interface

TCP does not have a user interface module of its own. Its functions and variables are accessible via an Interlisp Executive, and you can direct some of its debugging information to a window.

As a network protocol module, it extends capability to other programs which may have their own window interfaces, for example, Chat and FileBrowser.

Installation

The first step in installing TCP-IP is to add your workstation to a network supporting TCP-IP and communications with others on the net. The rest of this section contains a step-by-step set of directions for this installation.

After you are on the network, load the required `.LCOM` modules for the type of service you want. For a full description of these modules, see the Interlisp Files section.

Module	**Implementation**
TCPFTP	TCP-based file transfer protocol
TCPFTPSRV	TCP-based FTP server
TCPCHAT	TELNET protocol for the Chat system
TCPTFTP	TFTP protocol

Obtaining Network Addresses

The first thing you need to do is to get a TCP-IP address assigned to each of your workstations from your network administrator. If your site supports Domains, get the name of your local domain and the addresses of your domain server(s) from your network administrator. You will also need to know the network addresses and operating system of the hosts you want to communicate with and the addresses of any network gateways you have.

Note: The maximum length of the domain and organization fields is 20 characters each.

Be sure to find out whether your net is a true Class A, B or C network and is not broken up into subnets. If it is broken up into subnets, be sure to read the discussion on SUBNETMASKs in the Primer on IP Networks section.

Differences between TCP and the Medley File Systems

When running TCP-IP to a Sun from an 11xx, directory enumeration on an unmatched directory path returns a listing for the top-level directory of the logged-in user. The TCPFTP protocol does not support directory creation.

Warning: When running TCP-IP to a Sun, a file which is still open on the Sun can be deleted.

Creating HOST.TXT File

Create a `HOSTS.TXT` file containing entries for the TCP-IP hosts needed by the user community and place a copy of the file on either a directory contained in the `DIRECTORIES` search path of each workstation on the net or the local disk of each Interlisp workstation.

The following is a sample HOSTS.TXT file:

```
; Hosts.txt,
; Internet Hosts Table for Networks 192.20.10.0 and 174.23.0.0
; 12-Dec-86
;
; The format of this file is documented in RFC 810, "DoD Internet
; Host Table Specification", which is available online at SRI-NIC
; as the file
;              [SRI-NIC]<RFC>RFC952.TXT
;
; It may be retrieved via FTP using username ANONYMOUS with
; any password.
;
; or as the file
;                                        [INDIGO]<RFC>RFC952.TXT
; Read access to GV World. Valid GV credentials required.
;
; The format for entries is:
;
; GATEWAY: ADDR, ADDR : NAME : CPUTYPE : OPSYS : PROTOCOLS :
; HOST: ADDR, ALTERNATE-ADDR (if any): HOSTNAME,NICKNAME : CPUTYPE :
;   OPSYS : PROTOCOLS :
;
; Where:
;;  ADDR = internet address in decimal, e.g., 26.0.0.73
;;  CPUTYPE = machine type (Xerox-11xx, VAX-11/780, SUN, etc.)
;;  OPSYS = operating system (UNIX, TOPS20, TENEX, VMS, Interlisp, etc.)
;;  PROTOCOLS = transport/service (TCP/TELNET, TCP/FTP, etc.)
;;  : (colon) = field delimiter
;;  :: (2 colons, NO space between) = null field
;
HOST : 192.20.10.1 : Bach : Xerox-1108 : Interlisp : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.3 : PARC-VAXC : VAX-11/780 : UNIX : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.15 : Oberon : VAX-11/780 : VMS : TCP/TELNET, TCP/FTP :
HOST : 192.20.10.71 : Explorer : TI-EXPLORER : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.77.22 : Sunrise : SUN : UNIX : TCP/TELNET, TCP/FTP :
HOST : 174.23.30.21 : Rutgers : VAX-11/780 : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.76.21 : Simba : SYMBOLICS : SYMBOLICS-3600 : TCP/TELNET,
TFP/FTP :
GATEWAY : 192.20.10.240, 174.23.77.250 : Hellsgate : VMS : IP/GW :
```

This example shows a host table that indicates that there are four hosts (Bach, PARC-VAXC, Oberon, and Explorer) on net 192.20.10.0, three hosts (Sunrise, Rutgers, and Simba) on net 174.23.0.0 and a gateway (Hellsgate) that connects the two.

In regard to the OPSYS field in the HOSTS.TXT file, it is preferable to use values recognized by the Lisp variable NETWORKOSTYPES. Interlisp is the default value if a host’s OSType is not declared.

Note that if any host is accessible via another network protocol (for example, PUP or NS), you may desire to call the host by an unambiguous name when it is accessed via TCP. You can do this by giving it an unambiguous name in the `HOSTS.TXT` file.

If you ever modify the `HOSTS.TXT` table after `TCP.LCOM` has been loaded, use the function `(\HTE.READ.FILE` *'HOSTTABLE*`)` to reread the file.

For example,

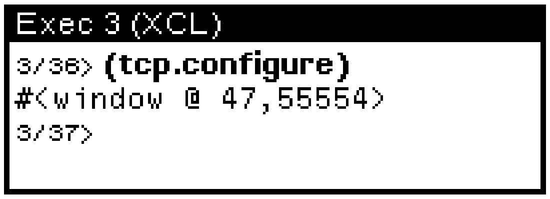
```
(\HTE.READ.FILE '{DSK}<LISPFILES>HOSTS.TXT)
```

`TCP.ALWAYS.READ.HOSTS.FILE` [Variable]

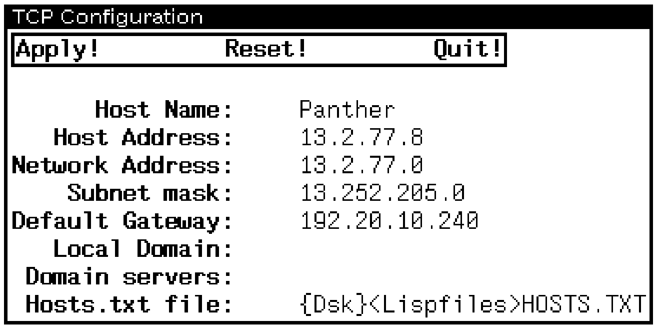
Initially set to `T`. Setting it to `NIL` causes the system to parse the `HOSTS.TXT` file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The `HOSTS.TXT` file will always be read at least once when loading the software into a clean sysout.

Creating the Local IP.INIT File

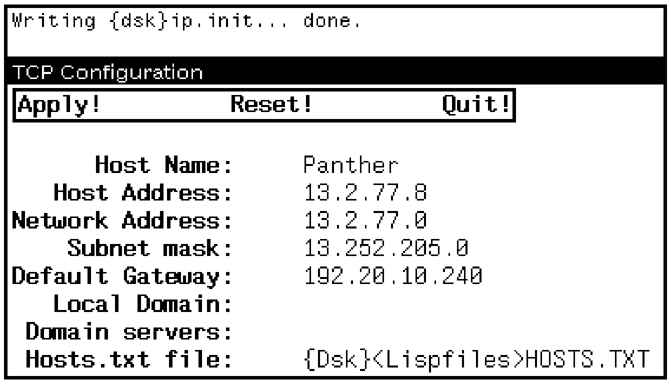
`TCP.CONFIGURE` brings up a menu that you complete.



```
Exec 3 (XCL)
3/36> (tcp.configure)
#<window @ 47,55554>
3/37>
```



```
TCP Configuration
Apply!          Reset!          Quit!

      Host Name:     Panther
   Host Address:     13.2.77.8
Network Address:     13.2.77.0
    Subnet mask:     13.252.205.0
Default Gateway:     192.20.10.240
   Local Domain:
 Domain servers:
 Hosts.txt file:     {Dsk}<Lispfiles>HOSTS.TXT
```



```
Writing {dsk}ip.init... done.

TCP Configuration
Apply!          Reset!          Quit!

      Host Name:     Panther
   Host Address:     13.2.77.8
Network Address:     13.2.77.0
    Subnet mask:     13.252.205.0
Default Gateway:     192.20.10.240
   Local Domain:
 Domain servers:
 Hosts.txt file:     {Dsk}<Lispfiles>HOSTS.TXT
```

If any field does not apply to your site, leave it blank.

Selecting **Apply!** writes the file `{DSK}<LISPFILES>IP.INIT` to the local disk.

Note: The file `{DSK}<LISPFILES>IP.INIT` must exist on each Interlisp machine before `TCP.LCOM` is loaded. And this file *must* remain on the workstation and must not be copied to other workstations. Also, the font GACHA 12 MRR must be available.

Selecting **Reset!** resets the menu to the original state.

Selecting **Quit!** closes the window.

You must perform the `TCP.CONFIGURE` step individually on each workstation, but you need to perform it only once. As long as there is an IP.INIT file on the workstation, the TCP-IP module will be configured automatically whenever it is loaded or initialized.

If you change your `IP.INIT` file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

Adding Host and Operating System Names to NETWORKOSTYPES

The variable `NETWORKOSTYPES` is used during Chat to determine the sequence of characters to send when performing auto-login. There should be an entry in `NETWORKOSTYPES` for each TCP host that you want to communicate with in the form `(TCPHOSTNAME . OSTYPE)`.

For example:

```
((SUNRISE . UNIX)(RUTGERS . TOPS-20) etc)
```

End-of-Line Conventions

To ensure correct line spacing, set the `TCPFTP.EOL.CONVENTION` switch. The associated function takes one argument, `TYPE`, which sets it correctly. `TYPE` can be one of the following:

`CR`	Set EOL to CR
`LF`	Set EOL to LF
`CRLF`	Set EOL to CRLF
`OS`	Set it to something based on the OS
`Other`	Set it to the default (CRLF)

Loading TCP

Make sure the variables `DIRECTORIES` and `LISPUSERSDIRECTORIES` point to the location of the `.LCOM` files of TCP-IP, or that they are in the connected directory.

You can then load `TCP.LCOM` which in turn loads its dependent files.

If you plan to do TCP file transfers, load `TCPFTP`.

If you plan to use the 11xx Lisp workstation as a TCPFTP Server host, load `TCPFTPSRV`. To start the server, evaluate `(TCPFTP.SERVER)`. An Interlisp machine running the TCPFTP server should be identified as a TOPS-20 machine in the other Interlisp machines' `HOSTS.TXT` table. It will thus masquesrade as a TOPS-20 server.

The rest is automatic. You can treat an Interlisp host running the server just like any other TCPFTP server. The default path for resolving filenames is `{DSK}<LISPFILES>`, but you can change or override it.

For example, assume {ERIC} is a machine running the FTP server. From another machine which has TCPFTP loaded, you can do `SEE {ERIC}{FLOPPY}FOO`, which will type out the file FOO located on the floppy drive of {ERIC}.

If you plan to Chat to a TCP host, load `CHAT`, `CHATTERMINAL`, `DMCHAT` and then `TCPCHAT.LCOM`. Be sure that hosts with which you wish to chat have their `NETWORKOSTYPES` set.

If you plan to use the TCP Trivial File Transfer Protocol, load TCPTFTP.

Interlisp's TCPTFTP also provides a TCPTFTP server. Load `TCPTFTP.LCOM` and evaluate `(TFTP.SERVER)`. You can then use the appropriate TFTP commands to copy files from the Interlisp machine; for example `TFTP.PUT` and `TFTP.GET`.

Verifying TCP Connections

Load `TCPDEBUG`. Execute `(TCPTRACE T)` and you will be prompted to open a window to show TCP packets. Select INCOMING, OUTGOING and CONTENTS from the window's menu. If the host that you are communicating with has a TCP echoserver process you can then try `(TCP.ECHOTEST 'HOSTNAME 3)`. For example, using the above `HOSTS.TXT` file this would be `(TCP.ECHOTEST 'SIMBA 3)`.

You will be prompted to open a window for the echo test and should see text, for example:

```
This is byte number 21
This is byte number 45
This is byte number 69
```

You should also see packets being sent and received in the TCPTRACE window.

Note: If a remote host is not running a TCP echo server process you will not get this response.

Connecting, Transferring Files, and Chatting to a Host

First log in to the host by typing **`(LOGIN 'HOST)`**; for example, **`(LOGIN 'SUNRISE)`**. You will then be prompted for a user name and password. This will be sent to the host when you attempt to CONNect or Chat.

Use the command CONN {HOST}<DIRECTORY>SUBDIR> to connect to a particular host. The local directory delimiters < and > can be used when connecting or file transferring. When communicating with a remote host you can specify the directory path as <DIRECTORY> SUBDIRECTORY> SUBDIRECTORY. . . , and the appropriate delimiters are presented to the remote host. Determination of what delimiter is presented depends upon the value of the OSTYPE field in the HOSTS.TXT file. If the field is empty, OSTYPE = 'Interlisp' is the default.

Using the above HOST.TXT file as an example you can do the following:

```
          UNIX  CONN {SUNRISE}<DIR>SUBDIR>SUBDIR>
           VMS  CONN {OBERON}<DIR>SUBDIR>SUBDIR>
      TOPS-20   CONN {RUTGERS}<DIR>SUBDIR>
SYMBOLICS-3600  CONN {SIMBA}<DIR>SUBDIR>
```

You can then do a DIR of the remote host, COPYFILE files to and from the host, assuming TCPFTP is loaded, MAKEFILE, etc.

Since the TCPFTP specification does not specify file type conventions, the variable TCP.DEFAULT.FILETYPES is used to associate a file's extension with the type of file it is. It is a list in the form (extension . type); for example,

```
((LCOM . BINARY)  (TXT . TEXT) etc)
```

Since UNIX systems are case-sensitive, you should also have the lower case version of the file extensions on this list. If a file extension is not found on this list, the variable TCP.DEFAULTFILETYPE is used as the default file type during file transfers.

To Chat to a remote host, select Chat from the background menu and enter the host name when prompted. You will be prompted for a Chat window and should then be able to chat to the host. If you have problems opening the Chat connection, try (CHAT 'HOST 'NONE). This will suppress the automatic login.

Warning: If no username or password has been provided for the TCP host, the default ID and password will be sent. This may create a security hazard.

Making a Sysout that Contains TCP-IP

1. Load Medley sysout.
2. Create TCP host table.
3. Load TCPCONFIG.LCOM and run TCP.CONFIGURE if there is no IP.INIT file on local disk.
4. Load TCP.LCOM.TCPFTP.LCOM, TCPCHAT.LCOM.
5. Load TCPDEBUG.LCOM if you always want to have trace and echo facilities available.
6. Evaluate (TCP.STOP)
7. Evaluate (STOPIP)
8. Evaluate

   ```
   (SETQ RESTARTETHERFNS (LIST '(LAMBDA NIL (AND \IPFLG
   (\IPINIT)))))
   ```

9. Load any other files that you want in this sysout.

10. Evaluate SYSOUT to the device of your choice. Evaluate `(\TCP.INIT)` to re-enable TCP.
11. Load TCP sysout on other machine.
12. Create TCP host table.
13. Evaluate `(TCP.CONFIGURE)` and identify the new machine.
14. Evaluate `(\TCP.INIT)` to re-enable TCP.
15. Evaluate (\IPINIT) to restart the IPLISTENER process.

TCP-IP Protocol Layers

The TCP-IP family consists of four principal protocol layers: the link layer, the internet layer, the transport layer, and the application layer.

Link Layer

The physical link layer, the medium for transferring packets between hosts, is assumed to be any medium capable of transporting packets of data between hosts. Common link layers in this family include the Ethernet and the ARPANET.

The Address Resolution (AR) Protocol enables hosts to map between internet addresses and link layer addresses.

For example, the internet layer protocol IP (see below) uses a 32-bit combined unique host and network address; the host address field is of variable size and depends on the pattern encoded in the high-order bits of the address. On the other hand, the 10 MB Ethernet uses a fixed-size 48-bit unique host address. The Address Resolution protocol, documented in RFC826, allows hosts to discover dynamically the link layer address equivalents of other internet hosts.

Internet Layer

The internet layer is responsible for routing packets between hosts. Unlike the link layer, the internet layer is capable of moving packets between hosts that are not connected to the same network. The term IP in TCP-IP refers to the Internet Protocol, the protocol that performs this task in the TCP-IP family. IP is documented in RFC791. IP is not assumed to be error-free; packets may be lost or duplicated while moving through the internet. It is the responsibility of the transport layer (see below) to guarantee perfect delivery, should the client require it.

IP also depends on an associated protocol called the Internet Control Message Protocol (ICMP). ICMP is responsible for handling exception conditions that arise between hosts using IP. Such conditions include the inability to deliver packets, errors in packet formats, etc. ICMP is documented in RFC792.

Transport Layer

The transport layer is responsible for assuring error-free, duplicate-free, sequenced delivery of packets between communicating processes. The most common transport layer is TCP, the Transport Control Protocol. TCP maintains the appearance of a perfect byte stream between processes. TCP is documented in RFC793.

An unreliable transport layer called the User Datagram Protocol (UDP) allows for packet exchange between communicating processes, but makes no attempt to guarantee delivery, suppress duplication, etc. Clients of UDP must provide their own error-recovery mechanisms if necessary. UDP is documented in RFC768.

Application Layer

Many applications exist in the TCP-IP family. The most common applications are file transfer, virtual terminal interaction, and mail delivery.

File Transfer

Two principal file transfer applications are in use: FTP, based on TCP and documented in RFC765; and TFTP (the Trivial File Transfer Protocol), based on UDP and documented in RFC783. Both are implemented in Interlisp, and are discussed at greater length below.

Virtual Terminal Interaction

The TELNET protocol, documented in RFC854, specifies the protocol for virtual terminal interaction between a user and a remote system. The Chat module will use the TELNET protocol to connect to TCP-only hosts.

Mail Delivery

The Simple Mail Transfer Protocol (SMTP) enables the delivery of mail between system elements using TCP. It is not currently implemented in Interlisp. SMTP is documented in RFC821. The format of messages is described in RFC822.

Primer on IP Networks

The Internet Protocol internetwork is a collection of IP networks, a subset of which may communicate with each other. Each network is assigned an IP address, which is composed of a network number and a host number. No two hosts in the internetwork have the same network and host number combination; the composition of the network and host number for a particular host unambiguously identifies that host within the internetwork.

Network Addresses

The address space of the internetwork is formed of the concatenated network and host numbers of its constituent hosts, and is 32 bits long. This 32-bit address space is currently partitioned into three classes of network addresses, known as class-A, class-B, and class-C:

Class-A addresses consist of 7 bits of network number and 24 bits of host number.

Class-B addresses consist of 14 bits of network number and 16 bits of host number.

Class-C addresses consist of 21 bits of network number and 8 bits of host number.

Thus, there may be 128 class-A networks, 16,384 class-B networks, and over two million class-C networks. In addition, a single class-A network has the capacity to address over 16 million hosts, while a class-C network can address only 255 hosts. The class to which a particular IP network belongs may be determined by examining the most significant bits of its address.

Network number assignments are strictly controlled by a central authority. Institutions requesting network assignments are given class-A, -B, or -C networks depending on their estimated eventual size (numbers of hosts). Sites without assigned network numbers may request an assigned number by contacting:

Joyce Reynolds
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695
Phone: (213) 822-1511
ARPANET: JKREYNOLDS@USC-ISIF.ARPA

IP addresses are normally stored or exchanged as single 32-bit numbers. The printed representation of an IP address takes the form W.X.Y.Z, where W through Z are the decimal equivalents of each of the 8--bit bytes that constitute the address. Class-A addresses are of the form N.H.H.H; class-B addresses are of the form N.N.H.H; and class-C addresses are of the form N.N.N.H, where N indicates a byte of the network number, and H indicates a byte of the host number.

Class-A: N.H.H.H The first number is between 0-127 (for example, 122.0.2.1)

Class-B: N.N.H.H The first number is between 128-191 (for example, 153.4.23.5)

Class-C: N.N.N.H The first number is between 192-255 (for example, 194.5.67.3)

For example, 36.47.0.12 is an address on network 36, a class-A network; and 192.10.200.1 is an address on network 192.10.200, a class-C address.

Broadcast Address

The Internet Protocol defines an address in which the host field contains all ones to be a broadcast address for its network. Thus, the address 36.255.255.255 is the broadcast address on network 36, and 192.10.200.255 is the broadcast address on network 192.10.200.

Subnets

It is quite common for class-B networks to be partitioned into a set of smaller subnetworks, which are really class-C networks, but have the wrong network number to be recognized as class-C networks. This is just as common is partitioning a class-A network into many class-B subnetworks. An implementation of TCP-IP that is not prepared to handle this violation of the IP standard will not be able to communicate with hosts on the same network but different subnetworks. Fortunately, extending an IP implementation to support subnetworks is straightforward.

SUBNETMASK is a 32-bit parameter that resembles an IP address. The purpose of the mask is to enable a host to determine when a destination IP address is or is not on the same subnet as the sending host itself.

The SUBNETMASK has the following properties:

- The bitwise-AND of a source host's address (for example, this machine) and the SUBNETMASK must be equal to the bitwise-AND of a destination host's address and the SUBNETMASK if and only if the two hosts are on the same subnetwork.
- The bitwise-AND of a source host's address and the SUBNETMASK must not be equal to the bitwise-AND of a destination host's address and the SUBNETMASK if and only if the two hosts are on different subnetworks.

As an example, consider network 39.0.0.0. This is a class-A network. Suppose this network consists of a number of subnetworks; for example, subnetworks with numbers like 39.47.*.* and 39.9.*.*. According to the IP specification, these subnetworks should really be one monolithic network, such that a host desiring to communicate with any other host whose address begins with 39... should have to take no special action with regard to routing packets to that host. Let us assume that this is not the case. The only way a machine has of telling which hosts are on different networks is to compare the masked version of the address with the masked version of its own address.

To continue the example further, assume the following:

Host A has address 39.9.0.6. Host A's SUBNETMASK is 39.255.0.0.

Host B has address 39.9.0.7. Host B's SUBNETMASK is also 39.255.0.0.

Host C has address 39.47.0.6. Host C's SUBNETMASK is also 39.255.0.0.

When host A sends to host B, it compares its masked address with host B's masked address, and finds them equal:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0;   39.9.0.7 AND 39.255.0.0 = 39.9.0.0

However, when host A sends to host C, it finds the masked comparison does not match:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0; 39.47.0.6 AND 39.255.0.0 = 36.47.0.0

Class-A networks that are subdivided into class-B subnetworks have SUBNETMASKs that look like X.255.0.0, where X is the class-A network number. Likewise, class-B networks subdivided into class-C subnetworks have SUBNETMASKs that look like X.Y.255.0, where X.Y is the class-B network number. Finally, networks in which subnet routing is not in use have SUBNETMASKs identical to their network addresses. For example, if network 36 did not use subnet routing, its SUBNETMASK would be 36.0.0.0.

The definitive document on this approach to subnetwork routing is RFC940.

Interlisp Files

The files that implement the TCP-IP protocol suite are divided into two classes: those that implement low-level functionality, normally not of interest to general users, and those that implement higher-level functionality for user programs (either application or transport layer protocols).

The higher-level functions reside in the files TCP, TCPDEBUG, TCPFTP, TCPCHAT, TCPNAMES, TCPPUDP, and TCPFTP.

TCP	The TCP layer. Implements TCP streams, based on the buffered TCP device (for example , BIN runs in microcode).
TCPDEBUG	Contains routines to help debug TCP and TCP-based applications.
TCPFTP	Contains the TCP-based file transfer protocol. Creates a new virtual I/O device, allowing transparent filing operations with TCP-only hosts.
TCPFTPSRV	Contains the TCP-based FTP server program. When the server program is running on a Xerox 1100-series workstation, other TCP-based hosts may transfer files to and from the workstation.
TCPNAMES	Implements translation of file name formats between operating system types.
TCPCHAT	Implements the TELNET protocol for the Chat system.
TCPUDP	Contains the UDP layer.

`TCPTFTP`	Implements the TFTP protocol. Creates a buffered TFTP device to allow efficient bulk transfer between hosts.

The low-level functions reside in the files `TCPLLIP`, `TCPLLICMP`, `TCPLLAR`, `TCPHTE`, and `TCPCONFIG`.

`TCPLLIP`	Implements the IP layer.
`TCPLLICMP`	Implements ICMP for IP.
`TCPLLAR`	Implements ARP for the 3- and 10-megabyte Ethernets.
`TCPHTE`	Implements the functionality necessary to parse RFC810-style `HOSTS.TXT` files. This allows name-to-address translation within the Interlisp host.
`TCPCONFIG`	Provides a function to carry on a configuration dialog when TCP-IP is first installed on a machine. This file needs to be loaded only once, to produce the file `{DSK}IP.INIT`. Thereafter, `TCPCONFIG` is needed only to reestablish or modify IP parameters.
`TCPDOMAIN`	Implements a domain host address lookup client.

TCP

TCP implements the transport control protocol for Interlisp. After TCP is loaded, Interlisp supports a TCP stream capable of bidirectional I/O to a remote system element. The following functions are intended for use by applications programs.

`(TCP.OPEN` *DST.HOST DST.PORT SRC.PORT MODE ACCESS NOERRORFLG OPTIONS*`)` [Function]

Opens a TCP stream to *DST.PORT* on *DST.HOST* from *SRC.PORT*.

DST.HOST can be a host name, an IP host address in text format (such as 192.10.200.1), or the 32-bit integer representation of an IP host address as returned by the function `DODIP.HOSTP` (which is documented under TCPLLIP).

DST.PORT is a 16-bit number representing a TCP port open in `LISTENING` mode on the remote system.

SRC.PORT is also a 16-bit number, but may be supplied as `NIL` to obtain a defaulted unique local port number.

MODE is either `ACTIVE`, meaning to act as initiator of the connection, or `PASSIVE`, meaning to wait for a remote system element to initiate the connection.

ACCESS is either `INPUT`, `OUTPUT`, or `APPEND` (`OUTPUT` and `APPEND` are treated in the same manner).

If *NOERRORFLG* is non-`NIL`, `TCP.OPEN` will return `NIL` if the connection fails; otherwise, `TCP.OPEN` will call `ERROR` to signal failure.

OPTIONS is an optional parameter which allows the application program to control some of the characteristics of the TCP connection. *OPTIONS* is supplied in property-list format. Currently, the only recognized option is MAXSEG, whose value should be the number of data bytes the remote TCP sender is allowed to place into a single TCP segment (Ethernet packet). The maximum value of MAXSEG is 536.

If TCP.OPEN succeeds, it returns a STREAM open as specified by *ACCESS*. The generic operations BIN, BOUT, PEEKBIN, BINS, BOUTS, READP, EOFP, OPENP, GETFILEPTR, FORCEOUTPUT, and CLOSEF may be performed on streams opened for suitable access.

(TCP.OTHER.STREAM *STREAM*) [Function]

Returns the *STREAM* open in the other direction with respect to *STREAM* (for example, if *STREAM* is open for INPUT, TCP.OTHER.STREAM returns a *STREAM* open for OUTPUT, and vice versa).

(TCP.URGENT.EVENT *STREAM*) [Function]

Returns an event upon which a user process may wait for URGENT data to arrive on *STREAM*.

(TCP.URGENTP *STREAM*) [Function]

Returns T if *STREAM* is currently reading URGENT data.

(TCP.URGENT.MARK *STREAM*) [Function]

Marks the current point in *STREAM* as the end of URGENT data. *STREAM* must be open for OUTPUT.

(TCP.CLOSE.SENDER *STREAM*) [Function]

Closes the output side of *STREAM*, which may be either the INPUT or OUTPUT stream for the connection. This function differs from CLOSEF in that the INPUT side of the connection is not closed (although the remote system element may close the connection once the local output side of the connection is closed).

(TCP.STOP) [Function]

Disables the TCP protocol, closing all open TCP streams.

(\TCP.INIT) [Function]

(Re)initializes the TCP module.

\TCP.DEFAULT.RECEIVE.WINDOW [Variable]

Is the default number of bytes allowed outstanding from the remote system. It is initially 4,096.

\TCP.DEFAULT.USER.TIMEOUT [Variable]

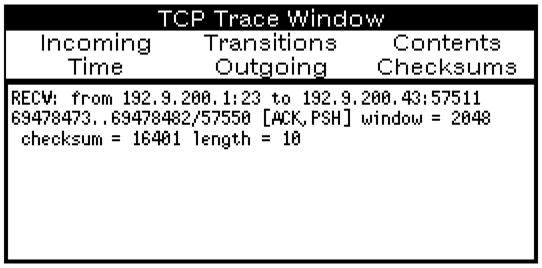
Is the default number of milliseconds a remote system element is allowed to remain silent before the TCP connection is declared broken. It is initially 60,000.

TCPDEBUG

TCPDEBUG implements tracing and test functions used to debug TCP and TCP-based applications.

(TCPTRACE) [Function]

Opens a trace window and attaches a menu to the window's top.



```
RECV: from 192.9.200.1:23 to 192.9.200.43:57511
69478473..69478482/57550 [ACK,PSH] window = 2048
 checksum = 16401 length = 10
```

The menu entries represent state changes or data elements to be traced; each entry is a toggle. Clicking on the toggle once will activate the trace of the particular element and will gray-over the entry; clicking a second time will deactivate the tracing and ungray the menu item. The following data elements/transitions may be displayed:

Contents	Displays a line's worth of packet contents. The Incoming or Outgoing switch must be on.
Incoming	Displays incoming data.
Outgoing	Displays outgoing data.
Checksums	Displays checksums for each TCP segment.
Time	Displays the time interval since the last action on the connection.
Transitions	Displays state transitions on the TCP state machine.

(PPTCB *TCB FILE*) [Function]

Prints the state of a TCP connection. PPTCB is normally the INFO function for the process that monitors a connection; thus, selecting INFO in the process status window will cause a window to pop up containing a report on the status of the associated connection.

(TCP.ECHOTEST *HOST NLINES*) [Function]

Opens a TCP connection to the TCP echo port on *HOST* and sends *NLINES* of random text. The echo responses are displayed in a window. If *NLINES* is NIL, the echo test will run forever.

(TCP.ECHO.SERVER *PORT*) [Function]

Starts a TCP echo server on *PORT* (defaults to the TCP echo port). It is usually more useful to start the echo server as a process by doing (ADD.PROCESS '(TCP.ECHO.SERVER *PORT*)).

(TCP.SINK.SERVER *PORT*) [Function]

Starts a TCP sink server on *PORT* (defaults to the TCP sink port). Any data sent to this port will be acknowledged and discarded. As with the TCP echo server, it is usually more useful to start this server as an independent process.

(TCP.FAUCET *HOST PORT NLINES*) [Function]

If *HOST* is non-NIL, this function opens a connection to *PORT* on *HOST* and sends *NLINES* of text (the default is to send lines of text forever). *PORT* defaults to the TCP sink port. If *HOST* is NIL, this function waits for a remote system to connect to the TCP faucet port and then sends out *NLINES* of random text.

TCPFTP

TCPFTP implements a virtual I/O device that performs Lisp filing operations transparently using the RFC765 FTP protocol. The standard filing operations of reading, writing, renaming, deleting, and directory enumeration are supported by the TCPFTP device. However, neither random access filing nor GETFILEINFO are supported, as there is no protocol specification for performing these operations on files. Interlisp operations such as RECOMPILE will not work when files are stored on TCPFTP file servers.

Once TCPFTP is loaded, filing operations should be transparent to users; no additional initialization need be performed. There are, however, three important global variables:

TCPFTP.EOL.CONVENTION [Variable]

This variable controls the end–of–line convention used with files accessed via TCP. Generally, you should set it to match the convention on the system where the files reside. The value can be one of those shown below.

(TCPFTP.EOL.CONVENTION *TYPE*) [Function]

Sets the variable TCP.EOL.CONVENTION to *TYPE*. *TYPE* can be one of the following:

CR	Set EOL to CR
LF	Set EOL to LF
CRLF	Set EOL to CRLF
OS	Set it to something based on the OS
Other	Set it to the default (CRLF)

TCPFTP.DEFAULT.FILETYPES [Variable]

This variable is an association list, keyed by common extensions of file names, and contains appropriate file types (for example, TEXT or BINARY) for such files. The TCPFTP protocol provides no mechanism for determining the type of a file about to be retrieved. The file type is usually known in the case of output operations (for example, COPYFILE or MAKEFILE to a file server). However, in the case of COPYFILE from a file server, the TCPFTP module has to infer the file type from other knowledge. The module tries to match the extension of the file name with an entry on the list TCPFTP.DEFAULT.FILETYPES. If it finds a match, it uses the value of the entry in the list as the file type of the file; if it doesn't find a match, it uses the value of TCP.DEFAULTFILETYPE for the file type of the file.

TCP.DEFAULTFILETYPE [Variable]

If no matching extension is found for the file being opened, the TCPFTP module uses the value of TCP.DEFAULTFILETYPE as the file type of the remote file. The initial value of TCP.DEFAULTFILETYPE is BINARY.

The following functions are available for debugging broken file server connections.

`(FTPDEBUG` *FLG*`)` [Function]

If *FLG* is `T`, this function opens a scrolling trace window that displays FTP commands as they are issued. PUPFTP commands will also be displayed in this window (the window is the value of FTPDEBUGLOG).

`(\TCP.BYE` *HOST*`)` [Function]

Breaks an FTP connection to *HOST*.

`(\TCPFTP.INIT)` [Function]

(Re)initializes the TCPFTP module.

TCPFTPSRV

The TCPFTPSRV module contains a program which implements an FTP service for Interlisp. When this program is running on a workstation, other hosts are able to store and retrieve files from the workstation.

`(TCPFTP.SERVER` *PORT DEFAULT.FILE.PATH*`)` [Function]

To start the server program, evaluate the form (TCPFTP.SERVER). If *PORT* is supplied, the FTP server program will listen for connections on the TCP port specified by *PORT*; otherwise, the server will listen on the default FTP server port, port 21.

If *DEFAULT.FILE.PATH* is supplied, the initial path for resolving file names will be relative to *DEFAULT.FILE.PATH*; the default value of this variable is `{DSK}<LISPFILES>`.

`TCPFTP.SERVER.USE.TOPS20.SYNTAX` [Variable]

This variable controls whether file names sent back to FTP client programs are formatted in Tops-20 or Interlisp syntax. If the variable is true (the default), all file names will be formatted in Tops-20 syntax. This permits an Interlisp workstation to masquerade as a Tops-20 mainframe for the purposes of file transfer to and from other vendors' machines.

TCPNAMES

The TCPNAMES module provides a set of functions for translating among the file-naming conventions of different operating systems. This is needed by the TCPFTP module in order for it to convert between Interlisp format file names and the file name formats of other operating systems.

`(REPACKFILENAME.STRING` *NAME FOROSTYPE*`)` [Function]

NAME is a file name in some operating system's format. *FOROSTYPE* is the name of an operating system. `REPACKFILENAME.STRING` attempts to translate *NAME* into a format acceptable to the operating system named by *FOROSTYPE*. *NAME* may be a string or atom; the function always returns a string.

Currently acceptable operating system types are:

```
IFS
INTERLISP
MS-DOS
SYMBOLICS-3600
TENEX
TOPS-20 (also TOPS20)
UNIX
VMS
```

(TI-Explorers should use TOPS-20 as their operating system.)

The correspondence between the target operating system type and the file name translation function is maintained in an extensible hash table.

(`\REPACKFILENAME.NEW.TRANSLATION` *OSTYPE FUNCTION*) [Function]

This function adds a new file name translation function for a new operating system type. The function must be a LAMBDA-NOSPREAD function, and must be prepared to receive either a single property-list format argument, such as would be returned by `UNPACKFILENAME`, or an arbitrary number of arguments in property-list format.

File names in the above format will be passed to the translation function adhering to the conventions of many operating systems; the function must recognize the operating system type and produce the desired output format, which must be a string.

`\REPACKFILENAME.OSTYPE.TABLE` [Variable]

This variable is the hash table that stores the correspondence between operating system types and translation functions.

TCPCHAT

TCPCHAT implements the TELNET protocol for virtual terminal I/O between Interlisp and a remote system. Once loaded into Interlisp, the standard Chat system will use TCP TELNET to communicate with hosts that are believed to support the protocol.

No user-callable functions reside in this module, although the following variables may be of interest.

`TCPCHAT.TELNET.TTY.TYPES` [Variable]

This variable is an association list that maps internal names of Chat terminal emulators to official terminal names as specified in RFC884, the TELNET Terminal Type Option. This allows TCPCHAT to set the user's terminal type automatically when a connection is established.

`TCPCHAT.TRACEFLG` [Variable]

If this variable is non-`NIL`, TELNET negotiations will be printed to `TCPCHAT.TRACEFILE` (see below). This is sometimes useful in debugging negotiation problems.

`TCPCHAT.TRACEFILE` [Variable]

TELNET negotiations are printed to this file if `TCPCHAT.TRACEFLG` is non-`NIL`.

TCPUDP

UDP implements the user datagram protocol. The following functions are meant to be called by client applications.

`(UDP.INIT)` [Function]

Initializes the UDP module. This function is normally called when UDP is loaded and should not need to be called again under normal circumstances.

`(UDP.STOP)` [Function]

Disables the UDP module, closing any open UDP sockets.

(`UDP.OPEN.SOCKET` *SKT# IFCLASH*) [Function]

Opens a socket for UDP operations.

SKT#, if supplied, is a 16-bit number and will default to a number between 1,000 and 65,535.

IFCLASH specifies what to do if the requested socket is already open and is handled as in OPENPUPSOCKET and OPENNSOCKET (see the *IRM*).

It returns an instance of an IPSOCKET.

(`UDP.CLOSE.SOCKET` *IPSOCKET NOERRORFLG*) [Function]

Closes an open *IPSOCKET*. If *IPSOCKET* is not an open socket and *NOERRORFLG* is `NIL`, an error will occur; otherwise, `NIL` is returned if the socket is not active, and `T` is returned if the socket is active.

Any remaining packets on the socket's input queue are discarded when this function is called.

(`UDP.SOCKET.EVENT` *IPSOCKET*) [Function]

Returns an event that a process may use to wait for packet arrival on *IPSOCKET*.

(`UDP.SOCKET.NUMBER` *IPSOCKET*) [Function]

Returns the socket number of *IPSOCKET*.

(`UDP.GET` *IPSOCKET WAIT*) [Function]

Returns the next packet waiting on *IPSOCKET*. If no packets are waiting, does one of the following based on the value of *WAIT*.

`NIL`	Returns immediately.
`T`	Waits forever for a packet to arrive.
a number	*FIXP* waits up to *WAIT* milliseconds for a packet to arrive and returns `NIL` if none arrived during that time.

Thus, this function is like GETPUP and GETXIP.

(`UDP.SETUP` *UDP DESTHOST DESTSOCKET ID IPSOCKET REQUEUE*) [Function]

Initializes a fresh packet (as returned from `\ALLOCATE.ETHERPACKET`). The packet will be sent to *DESTSOCKET* on *DESTHOST*.

ID is a number to be placed in the IP header ID field (zero is fine).

REQUEUE specifies what to do with the packet after it is sent; `NIL` (the default) means no special treatment; `FREE` means to release the packet and return it to the free packet queue. Any instance of a `SYSQUEUE` will cause the packet to be queued on the tail of the specified queue.

`UDP.SETUP` initializes all IP and UDP fields and sets the packet up as a minimum-length UDP packet.

(`UDP.SEND` *IPSOCKET UDP*) [Function]

Sends *UDP*, a UDP-formatted packet, out from *IPSOCKET*.

(`UDP.EXCHANGE` *IPSOCKET OUTUDP TIMEOUT*) [Function]

Sends *OUTUDP* out from *IPSOCKET* and waits *TIMEOUT* milliseconds for a response; returns NIL if no response came in during the specified interval, or the packet that did come in during that time.

Clears the socket's input packet queue before waiting for a packet to arrive.

(`UDP.APPEND.BYTE` *UDP BYTE*) [Function]

Appends *BYTE* to the UDP data portion of *UDP* and increments the UDP and IP length fields by one.

(`UDP.APPEND.WORD` *UDP WORD*) [Function]

Appends *WORD* to the UDP data portion of *UDP* and increments the UDP and IP length fields by two.

(`UDP.APPEND.CELL` *UDP CELL*) [Function]

Appends *CELL* to the UDP data portion of *UDP* and increments the UDP and IP length fields by four.

(`UDP.APPEND.STRING` *UDP STRING*) [Function]

Appends *STRING* to the UDP data portion of *UDP* and increments the UDP and IP length fields by the length *STRING*.

TCPTFTP

TFTP implements the trivial file transfer protocol. This protocol is useful for transferring unimportant files rapidly (for example, between workstations and printers). The following user-callable functions exist.

(`TFTP.PUT` *FROM TO PARAMETERS*) [Function]

Sends a file to a TFTP host.

FROM may refer to any accessible file; *TO* must refer to a file accessible via TFTP.

No attempt is currently made to translate between Interlisp file name syntax and remote system file name syntax for *TO*.

For example, if *TO* resides on a UNIX host, it would take a syntax like `{HOST}/DIRECTORY/SUBDIRECTORY/FILENAME`.

PARAMETERS is currently a list of parameters in the same format used by OPENFILE in `.PARAMETERS`; for example `((EOLCONVENTION 1) (TYPE TEXT))`.

Note: TFTP transfers between Xerox Lisp and UNIX hosts initiated from Xerox Lisp should have the PARAMETERS argument be `'((EOLCONVENTION 10))`.

`(TFTP.GET` *FROM TO PARAMETERS*`)` [Function]

Gets a file from a TFTP host. *FROM* must be a file accessible by TFTP; *TO* may be any file.

The file name syntax caveats for *FROM* are the same as for *TO* in TFTP.PUT. *PARAMETERS* is also as in `TFTP.PUT`.

`(TFTP.SERVER` *LOGSTREAM*`)` [Function]

Starts a TFTP server process.

LOGSTREAM may be left `NIL`, causing a new window to appear when the TFTP server is first invoked. Remote systems that support TFTP clients may store or retrieve files through any Interlisp workstation running the TFTP server.

The full Interlisp syntax for file names is supported; thus, requests to store files whose names include hosts will result in the Interlisp workstation's transparently storing the files on the designated hosts.

`(\TFTP.OPENFILE` *FILENAME ACCESS RECOG PARAMETERS*`)` [Function]

Returns a STREAM to open for *ACCESS* on *FILENAME*.

PARAMETERS is the usual format; `TYPE` is the only recognized parameter (`BINARY` opens a stream in *octet* format; `TEXT`, the default, opens a stream in NETASCII format; see RFC783).

`BIN`, `BOUT`, `READP`, `EOFP`, etc., may be used on this stream.

The stream is not RANDACCESSP.

`(\TFTP.CLOSEFILE` *STREAM*`)` [Function]

Closes the open stream. This is normally useful for streams open for OUTPUT; for INPUT streams, end-of-file will occur eventually.

TCPLLIP

For users planning implementations on top of IP, the following low-level TCP functions are available.

IP Socket Access

`(\IPINIT)` [Function]

Reinitializes the IP world; for example, after some catastrophe.

`(STOPIP)` [Function]

Disables IP.

(DODIP.HOSTP *NAME*) [Function]

If *NAME* is an integer, *NAME* is returned unaltered. If *NAME* is a text format IP host address (such as 192.10.200.1), DODIP.HOSTP returns its integer representation.

If *NAME* is a string or atom name, DODIP.HOSTP attempts to convert *NAME* to its IP host address integer value, using information supplied in the HOSTS.TXT file (see TCPHTE, below), followed by doing a domain query if TCPDOMAIN is loaded.

If *NAME* is unknown, DODIP.HOSTP returns NIL.

If *NAME* is known, it is cached with its corresponding address so that the function IPHOSTNAME may be used later to convert the address back to a name.

(IPHOSTNAME *IPADDRESS*) [Function]

Tries to convert *IPADDRESS* to a host name.

If *IPADDRESS* has no known name, it is converted to the text representation of an IP address (for example, 192.10.200.1).

(IPTRACE *MODE*) [Function]

Turns on tracing of IP activity. This function is like PUPTRACE and XIPTRACE, which are documented in the *IRM*.

If *MODE* is NIL, IP tracing is disabled.

If *MODE* is T, verbose IP tracing is enabled.

If *MODE* is PEEK, concise IP tracing is enabled. If *MODE* is either T or PEEK, the user is prompted for a window into which trace output will be printed.

(\IP.ADD.PROTOCOL *PROTOCOL SOCKETCOMPAREFN NOSOCKETFN INPUTFN ICMPFN*) [Function]

Defines a new IP-based protocol. The lowest-level IP functions maintain a list of active protocols and perform packet delivery based on the existence of open sockets for protocols of received packet types.

PROTOCOL is a protocol number, a number between 1 and 255. The following protocols are defined and should not be disturbed:

TCP	6
ICMP	1
UDP	17

SOCKETCOMPAREFN is a function with two arguments, an IP packet that has just been received and an open IPSOCKET. This function should return NIL if the packet does not belong to the supplied socket, or T if it does. The function will typically be interested in the IPSOCKET field of the IPSOCKET.

NOSOCKETFN is a function with one argument, an IP packet that has just been received. Its purpose is to handle received packets for which no socket can be found. If *NOSOCKETFN* is NIL, the default function, \IP.DEFAULT.NOSOCKETFN, will be used; this function simply returns an ICMP message indicating the socket is unreachable.

INPUTFN is a function with two arguments, a received IP packet and an open IPSOCKET. The *INPUTFN* is supposed to handle reception of packets when their destination socket has been found. If *INPUTFN* is `NIL`, the default function, `\IP.DEFAULT.INPUTFN`, will be supplied.

INPUTFN enqueues the received packet on the IPSQUEUE field of the IPSOCKET if the current queue length (stored in the IPSQUEUELENGTH field) is less than the allocated length (stored in the IPSQUEUEALLOC field).

INPUTFN also increments the IPSQUEUELENGTH field, and notifies the event stored in the IPSEVENT field.

ICMPFN is a function with two arguments and is called when an ICMP packet referring to the protocol is received. The first argument is a pointer to the received ICMP packet. The second argument is a pointer that may be used as if pointed to the original outgoing packet included in the ICMP data. This allows the protocol functions to parse the data in the ICMP packet to determine which socket sent the offending packet. The *ICMPFN* must never attempt to deallocate the packet identified by the second argument; however, it is quite permissible (and expected) that the *ICMPFN* will release the packet identified by the first argument. The default *ICMPFN* simply releases the packet identified by the first argument.

`\IP.ADD.PROTOCOL` returns an IPSOCKET datum, which represents the active protocol; it is not in fact a useful IPSOCKET and may be safely ignored.

(`\IP.DELETE.PROTOCOL` *PROTOCOL*) [Function]

Deactivates a protocol with protocol number *PROTOCOL*. Any open sockets are closed.

(`\IP.OPEN.SOCKET` *PROTOCOL SOCKET NOERRORFLG SOCKETCOMPAREFN NOSOCKETFN INPUTFN*) [Function]

Attempts to open an IPSOCKET for protocol *PROTOCOL*.

SOCKET is the identifying information for this socket; this quantity will be EQUAL-compared with other sockets open on *PROTOCOL*. Should a match be found, an error will occur unless *NOERRORFLG* is `T`, in which case the existing socket will be returned.

SOCKETCOMPAREFN, *NOSOCKETFN*, and *INPUTFN* may be supplied to override the functions specified when the protocol was defined; they are not normally useful, however.

(`\IP.CLOSE.SOCKET` *SOCKET PROTOCOL NOERRORFLG*) [Function]

Closes a socket open on *PROTOCOL*. *SOCKET* is the same quantity passed to `\IP.OPEN.SOCKET`; it is currently not an instance of an IPSOCKET. If *NOERRORFLG* is `T`, an error will not occur if the socket is not found.

IP Packet Building

The following functions are useful for placing bytes into IP packets (as allocated by `\ALLOCATE.ETHERPACKET`).

Most applications will probably want to define a block record to overlay the data portion of an IP packet. Here is an example of such a block record.

Note: Users who are developing new IP-based protocols will need to load `EXPORTS.ALL` from the library.

```
(ACCESSFNS UDP
    ((UDPBASE (\IPDATABASE DATUM)))
    (BLOCKRECORD UDPBASE
        ((UDPSOURCEPORT WORD)
        (UDPDESTPORT WORD)
        (UDPLENGTH WORD)
        (UDPCHECKSUM WORD)))
    (ACCESSFNS UDP
        ((UDPCONTENTS
        (\ADDBASE
            (\IPDATABASE DATUM)
            (FOLDHI \UDPOVLEN BYTESPERWORD))))))
```

(`\IP.APPEND.BYTE` *IP BYTE INHEADER*) [Function]

Appends *BYTE* to the IP data portion of *IP* and increments the IP length field by one. If *INHEADER* is `T`, the IPHEADERLENGTH field is appropriately incremented so that the bytes appear to have been appended to the options portion of the IP header. There must not be any data bytes in the data portion of the packet if this function is to work correctly.

(`\IP.APPEND.WORD` *IP WORD INHEADER*) [Function]

Appends *WORD* to the IP data portion of *IP* and increments the IP length field by two. *INHEADER* is as in `\IP.APPEND.BYTE`.

(`\IP.APPEND.CELL` *IP CELL INHEADER*) [Function]

Appends *CELL* to the IP data portion of *IP* and increments the IP length field by four. *INHEADER* is as in `\IP.APPEND.BYTE`.

(`\IP.APPEND.STRING` *IP STRING*) [Function]

Appends *STRING* to the IP data portion of *IP* and increments the IP length field by the length *STRING*.

IP Packet Sending

(`\IP.SETUPIP` *IP DESTHOST ID SOCKET REQUEUE*) [Function]

Initializes *IP*. This function should be called just after *IP* is obtained from `\ALLOCATE.ETHERPACKET`; if this is not done, the append functions above will fail.

DESTHOST is the 32-bit IP address to which this packet will be sent.

ID is an arbitrary 16-bit quantity that will become the IPID field of the packet.

SOCKET is the open IPSOCKET from which the packet will be sent.

REQUEUE defaults to FREE and controls the disposition of the packet after transmission (see the *IRM* for the documentation of SETUPPUP or FILLINXIP).

(`\IP.TRANSMIT` *IP*) [Function]

Tries to send *IP*. Performs IP checksum algorithm prior to sending. Returns NIL if successful, otherwise it returns a status indication, such as NoRouting or

AlreadyQueued. This function is like SENDPUP and SENDXIP, except that no socket argument is required.

TCPHTE

HTE provides functions for parsing `HOSTS.TXT` files as documented by RFC810. This file is loaded automatically by LLIP and is used by `\IPINIT` to read in the initial file, `HOSTS.TXT`. The following variable and function may be of interest.

`HOSTS.TEXT.DIRECTORIES` [Variable]

Is the search path for the file `HOSTS.TXT`. This variable is initialized to `NIL`; thus the search path to be used is by default `DIRECTORIES`.

`(\HTE.READ.FILE` *FILE WANTEDTYPES*`)` [Function]

Reads a `HOSTS.TXT` file.

WANTEDTYPES is a list of types drawn from the set *{HOST, NET, GATEWAY}*, to be read from the file; types not specified in *WANTEDTYPES* are ignored. *WANTEDTYPES* defaults to *(HOST*).

TCPDOMAIN

TCPDOMAIN enhances the host address lookup to do queries to a domain name server as specified by RFC883. This allows the system to only keep in memory the addresses of hosts that are actually communicated with, resulting in considerable savings of space on large networks. There is a slight delay in establishinig communications with a host the first time. Since it is faster to initiate communications with hosts defined in the `HOSTS.TEXT` file, we recommend that frequently-communicated-with hosts be defined therein.

`(DOMAIN.INIT)` [Function]

(Re)initializes the `TCPDOMAIN` package.

`(DOMAIN.TRACE)` [Function]

Opens up a window for tracing domain queries. (The window is the value of `DOMAIN.TRACE.FILE`.

`(DOMAIN.LOOKUP` *NAME TYPE SERVER*`)` [Function]

Looks up a host named *NAME* and type *TYPE* from server *SERVER*. If *SERVER* is unspecified, uses the default server.

`(DOMAIN.LOOKUP ADDRESS` *NAME SERVER DONT.GET.OSTYPE*`)` [Function]

Looks up the address of host *NAME* from server *SERVER*. If *SERVER* is unspecified, uses the default server. If `DONT.GET.OSTYPE` is `T`, will not get the *OSTYPE* of the host.

`(DOMAIN.LOOKUP.NAMESERVER` *NAME SERVER* `)` [Function]

Looks up the server serving the domain of *NAME*. If *SERVER* is unspecified, chooses the "best" server based on its knowledge of the namespace.

`(DOMAIN.GRAPH` *WINDOW* `)` [Function]

Draws a graph of the namespace in window *WINDOW*. Opens a new one if *WINDOW* is unspecified.

TCP Debugging Aids

With TCPDEBUG loaded use `(TCPTRACE T)` to open up a trace window of TCP traffic. The appropriate items need to be selected from the windows menu in order for data to be seen.

`(SETQ TCPCHAT.TRACEFLG T)` will print TELNET negotiations to a file, which is what the variable `TCPCHAT.TRACEFILE` points to.

`(FTPDEBUG T)` opens a scrolling trace window that displays FTP commands as they are issued. You will see unencrypted passwords if they are issued.

Limitations

You must use the 1186 microcode in order for TCP-IP to work on an 1186 (microcode for the 1185 will not do).

TCP-IP will not work with UNIX systems that have trailer encapsulation enabled. Connections will hang and then eventually break.

Directory enumeration on a VMS system results in `NIL`.

Known Problems in TCPFTPSRV

It does not handle error conditions in the middle of file transfers.

Doing a DIR gives you only filename and version: no author, creation date, etc. This is because the TCPFTP protocol specification doesn't support author, creation date, etc.

If there are multiple files on the system, deleting a file without specifing a specific version deletes the most recent version. The workaround is to give the specific version to delete.

The subdirectory structure is not presented back to the client host. If you have a file on both the <lispfiles> directory and a subdirectory, when you do a DIR *.* you do not see the subdirectory listed, but you do see that there are two files on the host with the same version number.

References

Users with access to the ARPANET may retrieve any RFC from host SRI-NIC.ARPA with the file transfer protocol (FTP) anonymous log-in option. RFCs are stored under `<RFC>RFC`*`nnn`*`.TXT`, where *nnn* is replaced by the number of the particular RFC.

From points on the Xerox internet, the RFC files can be retrieved from {Indigo}<RFC>. {Indigo} is an IFS host. From Lisp, you can simply (LOGIN) and supply your GV credentials if you haven't already, open a FileBrowser on that directory, and retrieve the file to the local workstation environment.

The following RFCs are mentioned in this manual:

RFC765 (superseded by RFC 959)
RFC768
RFC783
RFC791
RFC792
RFC793
RFC810 (superseded by RFC 952)
RFC814
RFC821
RFC822
RFC826
RFC854
RFC894
RFC895
RFC903
RFC904
RFC940

[This page intentionally left blank]

FREE MENU

Free Menus are powerful and flexible menus that are useful for applications that need menus with different types of items, including command items, state items, and editable items. A Free Menu lives in a window, which can be opened and closed as desired, or attached as a control menu to the application window.

Making a Free Menu

A Free Menu is built from a description of the contents and layout of the menu. As a Free Menu is simply a group of items, a Free Menu Description is simply a specification of a group of items. Each group has properties associated with it, as does each Free Menu Item. These properties specify the format of the items in the group, and the behavior of each item. The function FREEMENU takes a Free Menu Description, and returns a closed window with the Free Menu in it.

Probably the easiest way to make a Free Menu is to define your own function which calls FREEMENU with the Free Menu Description right there in your function. This function can then also set up the Free Menu window as required by the application. The Free Menu Description is then saved as part of your function when you save your application.

Alternatively, you can save the Free Menu Description as a variable in your file, and then just call FREEMENU with the name of the variable. This may be a more difficult alternative if you want to use the backquote facility to built your Free Menu Description. See the section **Free Menu Item Descriptions**.

Free Menu Formatting

A Free Menu can be formatted in one of four ways. The items in any group can be automatically layed out in rows, in columns, or in a table, or else the application can specify the exact location of each item in the group. Additionally, Free Menu keeps track of the region that a group of items occupies, and items can be justified within that region. This way an item can be automatically positioned at one of the nine justification locations, top-left, top-center, top-right, middle-left, etc.

Free Menu Description

A Free Menu Description, specifying a group of items, is a list structure. The first thing in the list is an optional list of the properties of this group of items, in the form:

```
    (PROPS <PROP> <VALUE> <PROP> <VALUE> ...)
```

The key word PROPS determines whether or not the optional group props list is specified. The section **Free Menu Group Properties** describes each group property. For now, the important property is FORMAT. The type of formatting determines the syntax of the rest of the Free Menu Description, in a very simple way.

When using EXPLICIT formatting, the rest of the description is any number of Item Descriptions, which have LEFT and BOTTOM properties specifying the position of the item in the menu. The syntax is:

```
((PROPS FORMAT EXPLICIT ...)
 <ITEM DESCRIPTION>
 <ITEM DESCRIPTION> ...)
```

When using ROW or TABLE formatting, the rest of the description is any number of item groups, each group corresponding to a row in the menu. These groups are identical in *syntax* to an EXPLICIT group description, with an optional PROPS list and then any number of Item Descriptions, except that the items need not have LEFT and BOTTOM properties, as the location of each item is figured out by the formatter. But the order of the rows and items is important. The menu is layed out top to bottom by row, and left to right within each row. The syntax is (the comments are not part of the description):

```
((PROPS FORMAT ROW ...)            ; props of this group
 (<ITEM DESCRIPTION>               ; items in first row
  <ITEM DESCRIPTION> ...)
 ((PROPS ...)                      ; props of second row
  <ITEM DESCRIPTION>               ; items in second row
  <ITEM DESCRIPTION> ...))
```

When using COLUMN formatting, the syntax is identical to that of ROW formatting. However each group of items corresponds to a column in the menu, rather than a row. The menu is layed out left to right by column, top to bottom within each column.

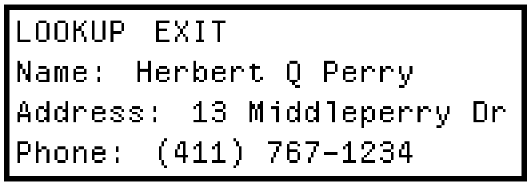
Finally, a Free Menu Description can have recursively nested groups. Anywhere the description can take an Item Description, it can take a group, marked by the key word GROUP. A nested group inherits all of the properties of its mother group, by default. However, any of these properties can be overridden in the nested groups PROPS list, including the FORMAT. The syntax is:

```
(                             ; no PROPS list, default row format
 (<ITEM DESCRIPTION>          ; first in row
  (GROUP                      ; nested group, second in row
     (PROPS FORMAT COLUMN...)     ; optional props
     (<ITEM DESCRIPTION> ...)     ; first column
     (<ITEM DESCRIPTION> ...))
  <ITEM DESCRIPTION>))        ; third in row
```

Here is an example of a simple Free Menu Description, for a menu which might provide access to a simple data base:

```
(((LABEL LOOKUP SELECTEDFN MYLOOKUPFN) (LABEL EXIT SELECTEDFN MYEXITFN))
 ((LABEL Name: TYPE DISPLAY) (LABEL "" TYPE EDIT ID NAME))
 ((LABEL Address: TYPE DISPLAY) (LABEL "" TYPE EDIT ID ADDRESS))
 ((LABEL Phone: TYPE DISPLAY)
  (LABEL "" TYPE EDIT LIMITCHARS MYPHONEP ID PHONE)))
```

This menu has two command buttons, LOOKUP and EXIT, and three edit fields, with ID's NAME, PHONE, and ADDRESS. The Edit items are initialized to the empty string, as in this example they need no other initial value. The user could click after the Name: prompt, type a person's name, and then press the LOOKUP button. This would cause the function MYLOOKUPFN to be called, which could look at the NAME Edit item, lookup that name in the data base, and then fill in the rest of the fields appropriately. Note that the PHONE item has MYPHONEP as a LIMITCHARS function. This function would be called when editing the phone number, in order to restrict input to a valid phone number. After looking up Perry, the Free Menu might look like:

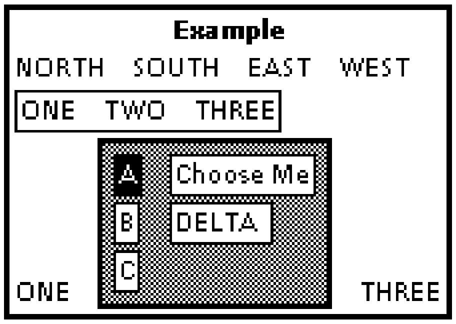


```
LOOKUP  EXIT
Name:  Herbert Q Perry
Address:  13 Middleperry Dr
Phone:  (411) 767-1234
```

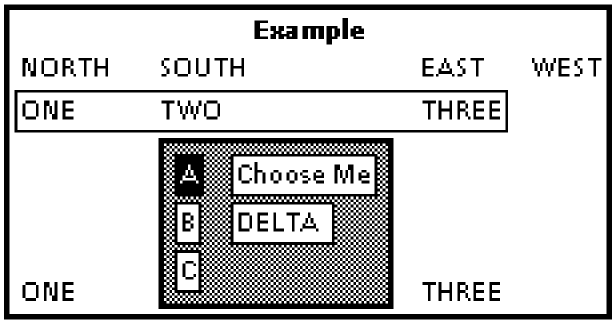
Here is a more complicated example:

```
((PROPS FONT (MODERN 10))
 ((LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
 ((LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
 ((PROPS ID ROW3 BOX 1)
  (LABEL ONE) (LABEL TWO) (LABEL THREE))
 ((PROPS ID ROW4)
  (LABEL ONE ID ALPHA)
  (GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
         ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
          (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
          (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
         ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
           INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
          (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
   (LABEL THREE)))
```

which will produce the following Free Menu:



And if the Free Menu were formatted as a Table, instead of in Rows, it would look like:



Free Menu Group Properties

Each group has properties. Most group properties are relevant, and should be set, in the group's PROPS list in the Free Menu Description. User properties can freely be included in the PROPS list. A few other properties are setup by the formatter. After the Free Menu is created, group properties can be accessed by the macro FM.GROUPPROP or FM.MENUPROP.

ID	The identifier of this group. Setting the group ID is desirable, for example, if the application needs to get handles on items in particular groups, or access group properties.
FORMAT	One of ROW, COLUMN, TABLE, or EXPLICIT. The default is ROW.
FONT	A font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type. This will be the default font for each item in this group. The default font of the top group is the value of the variable DEFAULTFONT.
COORDINATES	One of GROUP, or MENU. This property applies only to Explicit formatting. If GROUP, then the items in the explicit group are positioned in coordinates reletive to the lower left corner of the group, as determined by the mother group. If MENU, which is the default, then the items are positioned reletive to the lower left corner of the menu.
LEFT	Specifies a left offset for this group, pushing the group to the right.
BOTTOM	Specifies a bottom offset for this group, pushing the group up.
ROWSPACE	The number of bits between rows in this group.
COLUMNSPACE	The number of bits between columns in this group.
BOX	The number of bits in the box around this group of items.
BOXSHADE	The shade of the box.
BOXSPACE	The number of bits between the box and the items.
BACKGROUND	The background shade of this group. Nested groups will inherit this background shade, but items in this group and nested groups will not. This is because in general it is difficult to read text on a

background, so items appear on white background by default. This can be overridden by the BACKGROUND Item Property.

Other Group Properties

The following group properties are setup and maintained by Free Menu. The application should probably not change any of these properties.

ITEMS	A list of the items in the group.
REGION	The region that is the extent of the items in the group.
MOTHER	The ID of the group that is the mother of this group.
DAUGHTERS	A list of ID of groups which are daughters to this group.

Free Menu Items

Each Free Menu Item is stored as an instance of the Data Type FREEMENUITEM. Free Menu Items can be thought of as objects, each item having its own particular properties, such as its type, label, and mouse event functions. A number of useful item types, described in the section **Free Menu Item Types**, are predefined by Free Menu. New types of items can be defined by the application, using Display items as a base.

Each Free Menu Item is created from a Free Menu Item Description when the Free Menu is created.

Free Menu Item Descriptions

A Free Menu Item Description is a list in property list format, specifying the properties of the item. For example:

```
    (LABEL Refetch SELECTEDFN MY.REFETCHFN)
```

describes a command (Momentary) item labelled 'Refetch', with the function MY.REFETCHFN to be called when the item is selected.

None of the property values in an item description are evaluated. When constructing Free Menu descriptions that incorporate evaluated expressions, for example labels that are bitmaps, it is helpful to use the backquote facility. For example, if the value of the variable MYBITMAP is a bitmap, then

```
    (FREEMENU `(((LABEL A) (LABEL ,MYBITMAP))))
```

would create a Free Menu of one row, with two items in that row, the second of which has the value of MYBITMAP as its label.

Free Menu Item Properties

The following Free Menu Item Properties can be set in the Item Description. Any other properties given in an Item Description will be treated as user properties, and will be saved on the USERDATA property of the item.

TYPE	The type of the item. Choose from one of the Free Menu Item type keywords MOMENTARY, TOGGLE, 3STATE, STATE, NWAY, EDITSTART, EDIT, NUMBER, or DISPLAY. The default is MOMENTARY.

LABEL	An atom, string, or bit map. Bit maps are always copied, so that the original won't be changed. This property must be specified for every item.
FONT	The font that the item will appear in. The default is the font specified for the group that this item is in. Can be a font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type.
ID	May be used to specify a unique identifier for this item, but is not necessary.
LEFT and BOTTOM	When Row, Column, or Table formatting, these specify offsets, pushing the item right and up, respectively, from where the formatter would have put the item. In Explicit formatting, these are the actual coordinates of the item, in the coordinate system given by the group's COORDINATES property.
HJUSTIFY	One of LEFT, CENTER, or RIGHT. Specifies that this item is to be horizontally justified within the extent of its group. Note that the main group, as opposed to the smaller row or column group, is used.
VJUSTIFY	One of TOP, MIDDLE, or BOTTOM. Specifies that this item is to be vertically justified.
HIGHLIGHT	Specifies the highlighted looks of the item, that is, how the item changes when a mouse event occurs on it. See the section **Free Menu Item Highlighting**, below.
MESSAGE	A string that will be printed in the prompt window after a mouse button is held down over this item for MENUHELDWAIT milliseconds. Or, if an atom, treat as a function to get the message. The function is applied to ITEM WINDOW BUTTONS, and should return a string. The default is a message appropriate to the type of the item.
INITSTATE	The initial state of the item. This is only appropriate to TOGGLE, 3STATE, and STATE items.
MAXWIDTH	The width allowed for this item. The formatter will leave enough space after the item for the item to grow to this width without collisions.
MAXHEIGHT	Similar to MAXWIDTH, but in the vertical dimension.
BOX	The number of bits in the box around this item. Boxes are made around MAXWIDTH and MAXHEIGHT dimensions. If unspecified, no box is drawn.
BOXSHADE	The shade that the box is drawn in. The default is BLACKSHADE.
BOXSPACE	The number of bits between the box and the label. The default is one bit.
BACKGROUND	The background shade on which the item appears. The default is WHITESHADE, regardless of the group's background.
LINKS	Can be used to link this item to other items in the Free Menu. See the section **Free Menu Item Links**.

Mouse Properties

The following properties provide a way for application functions to be called under certain mouse events. These functions are called with the ITEM, the WINDOW, and the BUTTONS depressed as arguments. These application functions do not interfere with any Free Menu system functions that take care of handling the different item types. In each case, though, the application function is called *after* the system function. The default for all of these functions is NILL. The value of each of the following properties can be the name of a function, or a lambda expression.

SELECTEDFN	The function to be called when this item is selected. Note that Edit and EditStart items cannot have a selectedfn. See Edit items, below.
DOWNFN	The function to be called when a mouse button goes down over this item, or when the mouse moves over the item with buttons depressed.
HELDFN	The function to be called repeatedly while the mouse is held down over this item.
MOVEDFN	The function to be called when the mouse moves off this item with buttons still depressed.

System Properties

The following Free Menu Item properties are set and maintained by Free Menu. The application should probably not change these properties directly.

GROUPID	The ID of the smallest group that the item is in. For example, in a row formatted group, the item's GROUPID will be set to the ID of the row that the item is in, not the ID of the whole group.
STATE	The current state of TOGGLE, 3STATE, or STATE items. The state of an NWAY item behaves like that of a toggle item.
BITMAP	The bitmap from which the item is displayed.
REGION	The region of the item, in window coordinates. This is used for locating the display position, as well as determing the mouse sensitive region of the item.
MAXREGION	The maximum region the item may occupy, determined by the MAXWIDTH and MAXHEIGHT properties. This is used by the formatter and the display routines.
SYSDOWNFN SYSMOVEDFN SYSSELECTEDFN	These are the system mouse event functions, setup by Free Menu according to the type of the item. These functions are called before the users mouse event functions, and are used to implement highlighting, state changes, editing, etc.
USERDATA	Any other properties are stored on this list in property list format. This list should probably not need to be manipulated directly.

Predefined Item Types

Momentary

Momentary items are like command buttons. When the button is selected, its associated function is called.

Toggle

Toggle items are simple two-state buttons. When depressed the button is highlighted, and it stays that way until pressed again. The states of a toggle button are T and NIL, initially NIL.

3State

3State items rotate through NIL, T, and OFF, states each time they are pressed. The default looks of the OFF state are with a diagonal line through the button, while T is highlighted, and NIL is normal. The default initial state is NIL.

The following Item Property applies to 3State items:

OFF — Specifies the looks of a 3STATE item in its OFF state. Similar to HIGHLIGHT. The default is that the label gets a diagonal slash through it.

State

State items are general multiple state items. The following Item Property determines how the item changes state:

CHANGESTATE — This Item Property can be changed at any time to change the effect of the item. If a MENU datatype, then this menu is popped up when the item is selected, and the user can select the new state. Otherwise, if this property is given, it is treated as a function name, which is applied to ITEM WINDOW BUTTONS. This function can do whatever it wants, and is expected to return the new state (an atom, string, or bitmap), or NIL, meaning don't change state.

The state of the item can automatically be indicated in the Free Menu, by setting up a DISPLAY link to a Display item in the menu (see **Free Menu Item Links** below). If such a link exists, the label of the DISPLAY item will be changed to the new state. Note that the possible states are not restricted at all, except that if a popup menu is used, of course the possible selections are restricted. The state can be changed to any atom, string, or bitmap, manually via FM.CHANGESTATE.

The following Item Properties are relevent to State items when building a Free Menu:

MENUITEMS — If specified, should be a list of item to go in a popup menu for this item. Free Menu will build the menu and save it as the CHANGESTATE property of the item.

MENUFONT — The font of the items in the popup menu.

MENUTITLE — The title of the popup menu. The default title is the label of the State item.

Nway

NWay items provide a way to collect any number of items together, in any format within the Free Menu. Only one item from each Collection can be selected at a time, and that item is highlighted to indicate so.

The following Item Properties are particular to NWay items:

COLLECTION — An identifier that specifies which NWay Collection this item belongs to.

NWAYPROPS — A property list of information to be associated with this collection. This property is only noticed in the Free Menu Description on the first item in a Collection.

NWay Collections are formed by creating a number of NWay items with the same COLLECTION property. Each NWay item acts individually as a Toggle item, and can have its own mouse event functions.

Each NWay Collection itself has properties, its state for instance. After the Free Menu is created, these Collection properties can be accessed by the macro FM.NWAYPROPS. Note that NWay Collections are different from Free Menu Groups.

There are three NWay Collection properties that Free Menu looks at:

DESELECT	If given, specifies that the Collection can be deselected, yielding a state in which no item in the Collection is selected. When this property is set, the Collection can be deselected by pressing the Right mouse button on any item in the Collection.
STATE	The current state of the Collection, which is the actual item selected.
INITSTATE	Specifies the initial state of the Collection. The value of this property is an item Link Description (see the section **Free Menu Item Links**.)

Edit

Edit items are textual items that can be edited. The label for an Edit item cannot be a bitmap. When the item is selected an edit caret appears at that cursor position within the item, allowing inserting and deleting characters at that point. If selected with the Right mouse button, the item is cleared before editing starts. While editing, the Left mouse button moves the caret to a new position within the item. The Right mouse button deletes from the caret to the cursor. Control-W deletes the previous word.

Editing is stopped when another item is selected, when the user clicks in another tty window, or by the Free Menu function FM.ENDEDIT, which is called when the Free Menu is reset, or the window is closed. Additionally, the Free Menu editor will time out after about a minute, returning automatically. Because of the many ways in which editing can terminate, Edit items are not allowed to have a Selectedfn, as it is not clear when this function should be called.

Each Edit item should have an ID specified, which is used when getting the state of the Free Menu, since the string being edited is defined as the state of the item, and thus cannot distinguish edit items. The following Item Properties are particular to Edit items:

MAXWIDTH	Specifies the maximum string width of the item, in bits, after which input will be ignored. If MAXWIDTH is not specified, the items becomes ‘‘infinitely wide‘‘ and input is never restricted.
INFINITEWIDTH	This property is set automatically when MAXWIDTH is not specified. This tells Free Menu that the item has no right end, so that the item becomes mouse sensitive from its left edge to the right edge of the window, within the vertical space of the item.
LIMITCHARS	The input characters allowed can be restricted in two ways: If this item property is a list, it is treated as a list of legal characters; any character not in the list will be ignored. If it is an atom, it is treated as the name of a test predicate, which is applied to ITEM WINDOW CHARACTER when each character is typed. This predicate should return T if the character is legal, NIL otherwise. The LIMITCHARS function can also call FM.ENDEDIT to force the editor to terminate, or FM.SKIPNEXT, to cause the editor to jump to the next edit item in the menu.
ECHOCHAR	This item property can be set to any character. This character will be echoed in the window, regardless of what character is typed. However the item’s label contains the actual string typed. This is useful for operations like password prompting. If ECHOCHAR is used, the font of the item must be fixed pitch.

Unrestricted Edit items should not have other items to their right in the menu, as they will be edited over. If the item is boxed, input is restricted to what will fit in the box. Typing off the edge of the window will cause the window to scroll appropriately. Control characters can be edited, including CR and LF, and they are echoed as a black box. While editing, the Skip/Next key ends editing the current item, and starts editing the next Edit item in the Free Menu.

Number

Number items are Edit items that are restricted to numerals. The state of the item is coerced to the the number itself, not a string of numerals.

There is one Number specific Item Property:

NUMBERTYPE	If FLOATP (or FLOAT), then decimals are accepted. Otherwise only whole numbers can be edited.

EditStart

EditStart items serve the purpose of starting editing on another item when they are selected. The associated Edit item is linked to the EditStart item by an EDIT link (see **Free Menu Item Links** below). If the EditStart item is selected with the Right mouse button, the Edit item is cleared before editing is started. Similar to Edit items, EditStart items cannot have a Selectedfn, as it is not clear when the associated editing will terminate.

Display

Display items serve two purposes. First, they simply provide a way of putting dummy text in a Free Menu, which does nothing when selected. The item's label can be changed, though. Secondly, Display items can be used as the base for new item types. The application can create new item types by specifying DOWNFN, HELDFN, MOVEDFN, and SELECTEDFN for a Display item, making it behave as desired.

Free Menu Item Highlighting

Each Free Menu Item can specify how it wants to be highlighted. First of all, if the item doesn't specify a HIGHLIGHT property, there are two default highlights. If the item is not boxed, the label is simply inverted, as in normal menus. If the item is boxed, it is highlighted in the shade of the box.

Alternatively, the value of the HIGHLIGHT property can be a SHADE, which will be painted on top of the item when a mouse event occurs on it. Or the HIGHLIGHT property can be an alternate label, which can be an atom, string, or bitmap. If the highlight label is a different size than the item label, the formatter will leave enough space for the larger of the two.

In all of these cases, the looks of the highlighted item are determined when the Free Menu is built, and a bitmap of the item with these looks is created. This bitmap is stored on the item's HIGHLIGHT property, and simply displayed when a mouse event occurs. The value of the highlight property in the Item Description is copied to the userdata list, in case it is needed later for a label change.

Free Menu Item Links

Links between items are useful for grouping items in abstract ways. In particular, links are used for associating Editstart items with their item to edit, and State items with their state display. The Free Menu Item property LINKS is a property list, where the value of each Link Name property is a pointer to another item.

In the Item Description, the value of the LINK property should be a property list as above. The value of each Link Name property is a Link Description.

A Link Descriptions can be one of the following forms:

`<ID>`	Simply an ID of an item in the Free Menu. This is okay if items can be distinguished by ID alone.
`(<GROUPID> <ID>)`	A list whose first element is a GROUPID, and whose second element is the ID of an item in that group. This way items with similar purposes, and thus similar ID's, can be distinguished across groups.
`(GROUP <ID>)`	A list whose first element is the keyword GROUP, and whose second element is an item ID. This form describes an item with ID, in the same group that this item is in. This way you don't need to know the GROUPID, just which group you're in.

Then after the entire menu is built, the links are setup, turning the Link Descriptions into actual pointers to Free Menu Items. There is no reason why circular Item Links cannot be created, although such a link would probably not be very useful. If circular links are created, the Free Menu will not be garbage collected after it is not longer being used. The application is responsible for breaking any such links that it creates.

Free Menu Window Properties

FM.PROMPTWINDOW	Specifies the window that Free Menu should use for displaying the item's messages. If not specified, PROMPTWINDOW is used.
FM.BACKGROUND	The background shade of the entire Free Menu. This property can be set automatically by specifying a BACKGROUND argument to the function FREEMENU. The window border must be 4 or greater when a Free Menu background is used, due to the way the Window System handles window borders.
FM.DONTRESHAPE	Normally Free Menu will attempt to use empty space in a window by pushing items around to fill the space. When a Free Menu window is reshaped, the items are repositioned in the new shape. This can be disabled by setting the FM.DONTRESHAPE window property.

Free Menu Interface Functions

(FREEMENU *DESCRIPTION TITLE BACKGROUND BORDER*) [Function]

Creates a Free Menu from a Free Menu Description, returning the window. This function will return quickly unless new display fonts have to be created. See the example above.

Accessing Macros

These Accessing Macros are provided to allow the application to get and set information in the Free Menu data structures. They are implemented as macros so that the operation will compile into the actual access form, rather than figuring that out at run time.

(FM.ITEMPROP *ITEM PROP {VALUE}*) [Macro]

Similar to WINDOWPROP, this macro provides an easy access to the fields of a Free Menu Item. A handle on the item can be gotten from the Free Menu by the function FM.GETITEM, described below. *VALUE* is optional, and if not given, the current value of the *PROP* property will be returned. If *VALUE* is given, it will be used as the new value for that *PROP*, and the old value will be returned.

When a call to FM.ITEMPROP is compiled, if the *PROP* is known (quoted in the calling form), the macro figures out what field to access, and the appropriate Data Type access form is compiled. However, if the *PROP* is not known at compile time, the *function* FM.ITEMPROP, which goes through the necessary property selection at run time, is compiled.

The TYPE and USERDATA properties of a Free Menu Item are Read Only, and an error will result from trying to change the value of one of these properties.

(FM.GROUPPROP *WINDOW GROUP PROP {VALUE}*) [Macro]

Provides access to the Group Properties set up in the PROPS list for each group in the Free Menu Description. *GROUP* specifies the ID of the desired group, and *PROP* the name of the desired property. If *VALUE* is specified, it will become the new value of the property, and the old value will be returned. Otherwise, the current value is returned.

(FM.MENUPROP *WINDOW PROP {VALUE}*) [Macro]

Provides access to the group properties of the top-most group in the Free Menu, that is to say, the entire menu. This provides an easy way for the application to attach properties to the menu as a whole, as well as access the Group Properties for the entire menu.

(FM.NWAYPROP *WINDOW COLLECTION PROP {VALUE}*) [Macro]

This macro works just like FM.GROUPPROP, except it provides access to the NWay Collections.

Accessing Functions

(FM.GETITEM *ID GROUP WINDOW*) [Function]

Get a handle on item *ID* in *GROUP* of the Free Menu in *WINDOW*. This function will search the Free Menu for an item whose ID property matches, or secondly whose LABEL property matches *ID*. If *GROUP* is NIL, then the entire Free Menu is searched. If no matching item is found, NIL is returned.

(FM.GETSTATE *WINDOW*) [Function]

Return in property list format the ID and current STATE of every NWay Collection and item in the Free Menu. If an item’s or Collection’s state is NIL, then it is not included in the list. This provides an easy way of getting the state of the menu all at once. If the state of only one item or Collection is needed, the application can directly access the STATE property of that object using the Accessing Macros above. Note that this function can be called when editing is in progress, in which case it will provide the label of the item being edited at that point.

Changing Free Menus

Many of the following functions operate on Free Menu Items, and thus take the item as an argument. The *ITEM* argument to these functions can be the Free Menu Item itself, or just a reference to the item. In the second case, FM.GETITEM will be used to find the item in the Free Menu.

The reference can be in one of the following forms:

`<ID>`	Specifies the first item in the Free Menu whose ID or LABEL property matches <ID>.
`(<GROUPID> <ID>)`	Specifies the item whose ID or LABEL property matches <ID> within the group specified by <GROUPID>.

(FM.CHANGELABEL *ITEM NEWLABEL WINDOW UPDATEFLG*) [Function]

This function changes an item’s label after the Free Menu has been created. It works for any type of item, and state items will remain in their current state. If the window is open, the item will be redisplayed with its new appearance. *NEWLABEL* can be an atom, a string, or a bit map (except for Edit items), and will be restricted in size by the MAXWIDTH and MAXHEIGHT Item Properties. If these properties are unspecified, the item will be able to grow to any size. *UPDATEFLG* specifies whether or not the regions of the groups in the menu are recalculated to take into account the change of size of this item. The application should not change the label of an Edit item while it is being edited.

The following Item Property is relevant to changing labels:

CHANGELABELUPDATE	Exactly like *UPDATEFLG* except specified on the item, rather than as a function paramater.

(FM.CHANGESTATE *X NEWSTATE WINDOW*) [Function]

Programmatically changes the state of items and NWay Collections. *X* is either an item or a Collection name. For items *NEWSTATE* is a state appropriate to the type of the item. For NWay Collections, *NEWSTATE* should be the desired item in the Collection, or NIL to deselect. For Edit and Number items, this function just does a label change. If the window is open, the item will be redisplayed.

(FM.RESETSTATE *ITEM WINDOW*) [Function]

Set an item back to its initial state.

(FM.RESETMENU *WINDOW*) [Function]

Reset every item in the menu back to its initial state.

(FM.RESETSHAPE *WINDOW ALWAYSFLG*) [Function]

Reshapes the window to its full extent, leaving the lower-left corner unmoved. Unless *ALWAYSFLG* is T, the window will only be increased in size as a result of resetting the shape.

(FM.RESETGROUPS *WINDOW*) [Function]

Recalculate the extent of each group in the menu, updating group boxes and backgrounds appropriately.

(FM.HIGHLIGHTITEM *ITEM WINDOW*) [Function]

This function provides a way of programmatically forcing an item to be highlighted. This might be useful for items which have a direct effect on other items in the menu. The item will be highlighted according to its HIGHLIGHT property, as described in the section **Free Menu Item Highlighting**. Note that this highlight is temporary, and will be lost if the item is redisplayed, by scrolling for example.

Editor functions

(FM.EDITITEM *ITEM WINDOW CLEARFLG*) [Function]

Start editing an Edit or Number item at the beginning of the item, as long as the window is open. This function will most likely be useful for starting editing of an item that is currently the null string. If *CLEARFLG* is set, the item is cleared first.

(FM.SKIPNEXT *WINDOW CLEARFLG*) [Function]

This function causes the editor to jump to the beginning of the next Edit item in the Free Menu. If *CLEARFLG* is set, then the next item will be cleared first. If there is not another Edit item in the menu, this function will simply cause editing to stop. If this function is called when editing is not in progress, editing will begin on the first Edit item in the menu. This function can be called from any process, and can also be called from inside the editor, in a LIMITCHARS function.

(FM.ENDEDIT *WINDOW WAITFLG*) [Function]

Stop any editing going on in *WINDOW*. If *WAITFLG*, then block until the editor has completely finished. This function can be called from another process, or from a LIMITCHARS function.

(FM.EDITP *WINDOW*) [Function]

If an item is in the process of being edited in the Free Menu *WINDOW*, that item is returned. Otherwise, NIL is returned.

Miscelaneous

(FM.REDISPLAYMENU *WINDOW*) [Function]

Redisplays the entire Free Menu in its window, if the window is open.

(FM.REDISPLAYITEM *ITEM WINDOW*) [Function]

Redisplays a particular Free Menu Item in its window, if the window is open.

(FM.SHADE *X SHADE WINDOW*) [Function]

X can be an item, or a group ID. *SHADE* is painted on top of the item or group. Note that this is a temporary operation, and will be undone by redisplaying. For more permanent shading, the application may be able to add a REDEDISPLAYFN and SCROLLFN for the window as necessary to update the shading.

(FM.WHICHITEM *WINDOW POSorX Y*) [Function]

Gets a handle on an item from its known location within the window. If *WINDOW* is NIL, (WHICHW) is used, and if *POSorX* is NIL, the current cursor location is used.

(FM.TOPGROUPID *WINDOW*) [Function]

Return the ID of the top group of this Free Menu.

Free Menu Changes:

This document describes the incompatible changes from the old version to the new version of Free Menu. This document does not describe any of the new features of Free Menu. Some of the terminology used in these notes is introduced in the Free Menu documentation. You should read the Free Menu documentation first.

The function FREEMENU is used to create a Free Menu, replacing and combining the functions FM.MAKEMENU and FM.FORMATMENU.

In the Description of the Free Menu:

There is no longer a WINDOWPROPS list in the Free Menu Description. Instead, the window properties TITLE and BORDER that used to be set in the WINDOWPROPS list can now be passed to the function FREEMENU. Other window properties (like FM.PROMPTWINDOW) can be set directly after FREEMENU returns the window using the system function WINDOWPROP. See the section in the documentation entitled **Free Menu Window Properties**.

Setting the initial state of an item is now done with the item property INITSTATE in the item description, rather than the STATE property.

Free Menu Items:

3STATE items now have states OFF, NIL, and T (instead of a NEUTRAL state). They appear by default in the NIL state.

STATE items are general purpose items which maintain state, and replace the functionality of NCHOOSE items. To get the functionality of NCHOOSE items, specify the property MENUITEMS (a list of items to go in a popup menu), which instructs the STATE item to popup the menu when it is selected. STATE items do not display their current state by default, like NCHOOSE items used to. Instead, if you want the state displayed in the Free Menu, you have to link the STATE item to a DISPLAY item using a Free Menu Item Link named "DISPLAY". The current state of the STATE item will then automatically be displayed in the specified DISPLAY item. The item properties MENUFONT and MENUTITLE also apply to the popup menu.

NWAY items are declared slightly differently. There is now the notion of an NWay Collection, which is a collection of items acting an a single nway item. The Collection is declared by specifying any number of NWay items, each with the same COLLECTION property. NWay Collections have properties themselves, accessible by the macro FM.NWAYPROPS. These properties can be specified in property list format as the value of the NWAYPROPS Item Property of the first NWay item declared for each Collection. NWay Collections by default cannot be deselected (a state in which no item selected). Setting the Collection property DESELECT to any non-nil value changes this behavior. The state of the NWay Collection is maintained in its STATE property.

EDIT items no longer will stop at the edge of the window. Editing is either restricted by the MAXWIDTH property, or else it is not restricted at all. The EDITSTOP property is obsolete. Starting editing with the Right mouse button causes the item to be cleared first.

EDITSTART items now specify their associated edit item (there can only be one, now) by a Free Menu Item Link named "EDIT" from the EDITSTART item to the EDIT item.

TITLE items are replaced by DISPLAY items, which work the same way.

Free Menu Interface functions:

(FREEMENU DESCRIPTION TITLE BACKGROUND BORDER) replaces FM.MAKEMENU and FM.FORMATMENU.
The desired format is not specified as the value of the FORMAT property in the group's PROPS list.

(FM.GETITEM ID GROUP WINDOW) replaces FM.ITEMFROMID.
Searches within GROUP for an item whose ID property is *ID*.
ID is matched against the item ID and then the item LABEL. If *GROUP* is NIL, the entire menu is searched.

(FM.GETSTATE WINDOW) replaces FM.READSTATE.
Returns a property list of the selected item in the menu. This list now also includes the NWay Collections and their selected item.

(FM.CHANGELABEL ITEM NEWLABEL WINDOW UPDATEFLG) **new argument order**.
Now works by rebuilding the item label from scratch, taking the original specification of MAXWIDTH and MAXHEIGHT into account. NEWLABEL can be an atom, string, or bitmap. If UPDATEFLG is set, then the Free Menu Group's regions are recalculated, so that boxed groups will be redisplayed properly.

(FM.CHANGESTATE X NEWSTATE WINDOW) **new argument order**.
X is either an item or an NWay Collection ID. NEWSTATE is an appropriate state to the type of item. If an NWay collection, NEWSTATE is the actual item to be selected, or NIL to deselect. Toggle items take either T or NIL as NEWSTATE, and 3STATE items take OFF, NIL, or T, and STATE items take any atom, string, or

bitmap as their new state. For EDIT items, *NEWSTATE* is the new label, and FM.CHANGELABEL is called to change the label of the EDIT item.

(FM.RESETSHAPE WINDOW ALWAYSFLG) replaces FM.FIXSHAPE

(FM.HIGHLIGHTITEM ITEM WINDOW) replaces FM.SHADEITEM and FM.SHADEITEMBM.
FM.HIGHLIGHTITEM will programmatically highlight an item, as specified by its HIGHLGIHT property. The highlighting is temporary, and will be undone by a redisplay or scroll. To programmatically shade an item an arbitrary shade, use the new function FM.SHADE.

ICONW

This is a package of functions for building small windows of arbitrary shape, principally for use as icons for shrinking windows; i.e., these functions are likely to be invoked from within the ICONFN of a window.

An icon is specified by supplying its image (a bit map) and a mask that specifies its shape. The mask is a bit map of the same dimensions as the image whose bits are on (black) in those positions considered to be in the image, off (white) in those positions where the background should ‘‘show through.’’ By using the mask and appropriate window functions, the icon package maintains the illusion that the icon window is nonrectangular, even though the actual window itself is rectangular. The illusion is not complete, of course. For example, if you try to select what looks like the background (or an occluded window) around the icon but still within its rectangular perimeter, the icon window itself is selected. Also, if you move a window occluded by an icon, the icon never notices that the background changed behind it.

Icons created with this package can also have ‘‘titles’’; some part of the image can be filled with text computed at the time the icon is created, or even changed after creation.

Creating Icons

(ICONW *IMAGE MASK POSITION NOOPENFLG*) [Function]

Creates a window at *POSITION*, or prompts for a position if *POSITION* is NIL. The window is borderless, and filled with *IMAGE*, as cookie-cut by *MASK*. If *MASK* is NIL, the image is considered rectangular (i.e., *MASK* defaults to a black bit map of the same dimensions as *IMAGE*). If *NOOPENFLG* is T, the window is returned unopened.

(TITLEDICONW *ICON TITLE FONT POSITION NOOPENFLG JUST BREAKCHARS OPERATION*) [Function]

Creates a titled icon at *POSITION*, or prompts for a position if *POSITION* is NIL. If *NOOPENFLG* is T, the window is returned unopened. The argument *ICON* is an instance of the record TITLEDICON, which specifies the icon image and mask, as with ICONW, and a region within the image to be used for displaying the title. Thus, the *ICON* argument is usually of the form

(create TITLEDICON ICON_*someIconImage*

MASK_*iconMask* TITLEREG_*someRegionWithinICON*).

The title region is specified in icon-relative coordinates, i.e., the lower-left corner of the image bit map is (0, 0). The mask can be NIL if the icon is rectangular. The image should be white where it is covered by the title region (in any event, TITLEDICONW clears the region before printing on it).

The title is printed into the specified region in the image, using *FONT*, which if NIL defaults to the value of DEFAULTICONFONT, initially Helvetica 10. The title is broken into multiple lines if necessary; TITLEDICONW attempts to place the breaks at characters that are in the list of character codes *BREAKCHARS*. *BREAKCHARS* defaults to (CHARCODE (SPACE ÿ)). In addition, line breaks are forced by any carriage returns in *TITLE*, independent of *BREAKCHARS*. *BREAKCHARS* is ignored as needed if a long title would not otherwise fit in the specified region. For convenience, *BREAKCHARS* = FILE means the title is a file name, so break at file name field delimiters.

The argument *JUST* indicates how the text should be justified relative to the region—it is an atom or list of atoms chosen from TOP, BOTTOM, LEFT, or RIGHT, which indicate the vertical positioning (flush to top or bottom) and/or horizontal positioning (flush to left edge or right). Where not indicated, the text is centered.

The argument *OPERATION* is a display stream operation indicating how the title should be printed. If *OPERATION* is INVERT, then the title is printed white-on-black. The default *OPERATION* is REPLACE, meaning black-on-white. ERASE is the same as INVERT; PAINT is the same as REPLACE.

For convenience, TITLEDICONW can also be used to create icons that consist solely of a title, with no special image. If the argument *ICON* is NIL, TITLEDICONW creates a rectangular icon large enough to contain *TITLE*, with a border the same width as a regular window. The remaining arguments are as described above, except that a *JUST* of TOP or BOTTOM is not meaningful.

Modifying Icons

(ICONW.TITLE *ICON TITLE*) [Function]

Returns the current title of the window *ICON*, which must be a window returned by TITLEDICONW. Additionally, if *TITLE* is non-NIL, makes *TITLE* be the new title of the window and repaints it accordingly. To erase the current title, make *TITLE* be a null string.

(ICONW.SHADE *WINDOW SHADE*) [Function]

Returns the current shading of the window *ICON*, which must be a window returned by ICONW or TITLEDICONW. Additionally, if *SHADE* is non-NIL, paints the texture *SHADE* on *WINDOW*. A typical use for this function is to communicate a change of state in a window that is shrunk, without reopening the window. To remove any shading, make *SHADE* be WHITESHADE.

Default Icons

When you shrink a window that has no ICONFN, the system currently creates an icon that looks like the window’s title bar. You can make the system instead create titled icons by setting the global variable DEFAULTICONFN to the value TEXTICON.

(TEXTICON *WINDOW TEXT*) [Function]

Creates a titled icon window for the main window *WINDOW* containing the text *TEXT*, or the window’s title if *TEXT* is NIL.

DEFAULTTEXTICON [Variable]

The value that TEXTICON passes to TITLEDICONW as its ICON argument. Initially NIL, which creates an unadorned rectangular window, but you can set it to a TITLEDICON record of your choosing if you would like default icons to have a different appearance.

Sample Icons

The file <LispUsers>StockIcons contains a collection of icons and their masks usable with ICONW, including:

FOLDER, FOLDERMASKžA file folder

PAPERICON, PAPERICONMASK—A sheet of paper with the top right corner turned

FILEDRAWER, FILEDRAWERMASK—The front of a file drawer

ENVELOPEICON, ENVELOPEMASK—An envelope

TITLED.FILEDRAWER—A TitledIcon of the filedrawer front (Capacity, about three lines of 10-pt. text)

TITLED.FILEFOLDER—A TitledIcon of the file folder (Capacity, about three lines of 10-pt. text)

TITLED.ENVELOPE—A TitledIcon of the envelope (Capacity, one short line of 10-pt. text)

APPENDIX B. SEDIT—THE LISP EDITOR

SEdit is the Xerox Lisp structure editor. It allows you to edit Xerox Lisp code directly in memory. This editor replaces DEdit in Chapter 16, Structure Editor, of the *Interlisp-D Reference Manual*. First introduced in Lyric, the SEdit structure editor has been greatly enhanced in the Medley release. Medley additions are indicated with revision bars in the right margin.

All symbols referenced in this appendix are external in the package named "SEdit", unless otherwise qualified.

16.1 SEdit - The Structure Editor

As a structure editor, SEdit alters Lisp code directly in memory. The effect this has on the running system depends on what is being edited.

For Common Lisp definitions, SEdit always edits a copy of the object. For example, with functions, it edits the definition of the function. What the system actually runs is the installed function, either compiled or interpreted. The primary difference between the definition and the installed function is that comment forms are removed from the definition to produce the installed function. The changes made while editing a function will not be installed until the edit session is complete.

For Interlisp functions and macros, SEdit edits the actual structure that will be run. An exception to this is an edit of an EXPR definition of a compiled function. In this case, changes are included and the function is unsaved when the edit session is completed.

SEdit edits all other structures, such as variables and property lists, directly. SEdit installs all changes as they are made.

If an error is made during an SEdit session, abort the edit with an Abort command (see Section 16.1.7, Command Keys). This command undoes all changes from the beginning of the edit session and exits from SEdit without changing your environment.

If the definition being edited is redefined while the edit window is open, SEdit redisplays the new definition. Any edits on the old definition will be lost. If SEdit was busy when the redefinition occurred, the SEdit window will be gray. When SEdit is no longer busy, position the cursor in the SEdit window and press the left mouse button; SEdit will get the new definition and display it.

16.1.1 An Edit Session

The List Structure Editor discussion in Chapter 3, Language Integration, explains how to start an editor in Xerox Lisp.

Whenever you call SEdit, a new SEdit window is created. This SEdit window has its own process, and thus does not rely on an Exec to run in. You can make edits in the window, shrink it while you do something else, expand it and edit some more, and finally close the window when you are done.

Throughout an edit session, SEdit remembers everything that you do through a change history. All edits can be undone and redone sequentially. When an edit session ends, SEdit forgets this information and installs the changes in the system.

The session ends with an event signalling to the editor that changes are complete. Three events signal completion:

- Closing the window.

Do this to terminate the edit session when you are finished.

- Shrinking the window.

Shrink the window when you have made some edits and may want to continue the editing session at a later time.

- Typing one of the Completion Commands, listed below.

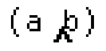
Each of these commands has the effect of installing your changes, completing the edit, and returning the TTY process to the Exec. They vary in what is done in addition to completing. Using these commands the definition that you were editing can be automatically compiled, the edit window can be closed, or both.

A new edit session begins when you come back to an SEdit after completing. The change history is discarded at this point.

If the Exec is waiting for SEdit to return before going on, complete the edit session using any of the methods above to alert the Exec that SEdit is done. The TTY process passes back to the Exec .

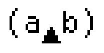
16.1.2 SEdit Carets

There are two carets in SEdit, the edit caret and the structure caret. The edit caret appears when characters are edited within a single structure, such as an atom, string, or comment. Anything typed in will appear at the edit caret as part of the structure that the caret is within. The edit caret looks like this:



(a b)

The structure caret appears when the edit point is between structures, so that anything inserted will go into a new structure. It looks like this:

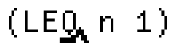


(a b)

SEdit changes the caret frequently, depending on where you are in the structure you are editing, and how the caret is positioned. The left mouse button allows an edit caret position to be set. The middle mouse button allows the structure caret position to be set .

16.1.3 The Mouse

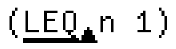
In SEdit, the mouse buttons are used as follows. The left mouse button positions the mouse cursor to point to parts of Lisp structures. The middle mouse button positions the mouse cursor to point to whole Lisp structures. Thus, selecting the Q in LEQ using the left mouse button selects that character, and sets the edit caret after the Q:



```
(LEQ n 1)
```

Any characters typed in at this point would be appended to the atom LEQ.

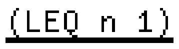
Selecting the same letter using the middle mouse button selects the whole atom (this convention matches TEdit's character/word selection convention), and sets a structure caret between the LEQ and the n:



```
(LEQ n 1)
```

At this point, any characters typed in would form a new atom between the LEQ and the n.

Larger structures can be selected in two ways. Use the middle mouse button to position the mouse cursor on the parenthesis of the desired list to select that list. Press the mouse button multiple times, without moving the mouse, extends the selection. Using the previous example, if the middle button were pressed twice, the list (LEQ ...) would be selected:

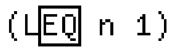


```
(LEQ n 1)
```

Pressing the button a third time would cause the list containing the (LEQ n 1) to be selected.

The right mouse button positions the mouse cursor for selecting sequences of structures or substructures. Extended selections are indicated by a box enclosing the structures selected. The selection is extended in the same mode as the original selection. That is, if the original selection were a character selection, the right button could be used to select more characters in the same atom. Extended selections also have the property of being marked for pending deletion. That is, the selection takes the place of the caret, and anything typed in is inserted in place of the selection.

For example, selecting the E by pressing the left mouse button and selecting the Q by pressing the right mouse button would produce:



```
(LEQ n 1)
```

Similarly, pressing the middle mouse button and then selecting with the right mouse button extends the selection by whole structures. Thus, in our example, pressing the middle mouse button to select LEQ and pressing the right mouse button to select the 1 would produce:

$(\boxed{\texttt{LEQ n 1}})$

This is not the same as selecting the entire list, as above. Instead, the elements in the list are collectively selected, but the list itself is not.

16.1.4 Gaps

The SEdit structure editor requires that everything edited must have an underlying Lisp structure, even if the structure is not directly displayed. For example, with quoted forms the actual structure might be (**QUOTE** GREEN), although this would be displayed as `'GREEN`. Even when the user is in the midst of typing in a form, the underlying Lisp structure must exist.

Because of this necessity, SEdit provides gaps to serve as dummy Lisp objects during typing. SEdit does not need a gap for every form typed in, but gaps are necessary for quoted objects. When something is typed that requires SEdit to build a Lisp structure and thus create a gap, as the quote character does, the gap will appear marked for pending deletion. This means it is ready to be replaced by the structure to be typed in. In this way it is possible to type special structures, like quotes, directly, while SEdit maintains the structure.

A gap looks like: –x–

A gap displayed after a quote has been typed in would look like this:

$'\boxed{–x–}$

with the gap marked for pending deletion, ready for typein of the object to be quoted.

16.1.5 Special Characters

A few characters have special meaning in Lisp, and are treated specially by SEdit. SEdit must always have a complete structure to work on at any level of the edit. This means that SEdit needs a special way to type in structures such as lists, strings, and quoted objects. In most instances these structures can be typed in just as they would be to a regular Exec, but in a few cases this is not possible.

Lists- (and)

Lists begin with an open parenthesis character **(**. Typing an open parenthesis gives a balanced list, that is, SEdit inserts both an open and a close parenthesis. The structure caret is between the two parentheses. List elements can be typed in at the structure caret. When a close parenthesis, **)** is typed, the caret will be moved outside the list (and the close parenthesis), effectively finishing the list. Square bracket characters, [and], have no special meaning in SEdit, as they have no special meaning in Common Lisp.

Quoted Structures: SEdit handles the quote keys so that it is possible to type in all quote forms directly. When typing one of the following quote keys at a structure caret, the quote character typed will appear, followed by a gap to be replaced by the object to be quoted.

Single Quote – '	Use to enter quoted structures.
Backquote –‘	Use to enter backquoted structures.
Comma – ,	Use to enter comma forms, as used with a Backquote form.
At Sign – @	Use after a comma to create a comma-at-sign gap. This allows type-in of comma-at forms, e.g. **,@list**, as used within a Backquote form.
Dot – .	Use the dot (period) after a comma to create a comma-dot gap. This allows type-in of comma-dot forms, e.g. **,.list**, as used within a Backquote form.
Hash Quote – #'	Use this two character sequence to enter the **CL:FUNCTION** abbreviation hash–quote (#').

Dotted Lists: The dot, or period, character (**.**) is used to type dotted lists in SEdit. After typing a dot, SEdit inserts a dot and a gap to fill in for the tail of the list. To dot an existing list, point the cursor between the last and second to the last element in the list, and type a dot. To undot a list, select the tail of the list before the dot while holding down the SHIFT key.

Escape- \ or % Use to escape from a specific typed in character. Use the escape key to enter characters, like parentheses, which otherwise have special meaning to the SEdit reader. Press the escape key then type in the character to escape. SEdit uses the escape key appropriate to the environment it is editing in; it depends on the readtable that was current when the editor was started. The backslash key (****) is used when editing Common Lisp, and the percent key (**%**) is used when editing Interlisp.

Multiple Escape- | Use the multiple escape key, the vertical bar character (**|**), to escape a sequence of typed in characters. SEdit always balances multiple escape characters. When one multiple escape character is typed, SEdit produces a balanced pair, with the caret between them, ready for typing in the characters to be escaped. If you type a second vertical bar, the caret moves after the second vertical bar, and is still within the same atom, so that you can add more unescaped characters to the atom.

Comments- ; The comment key, a semicolon (**;**), starts a comment. When a semicolon is typed, an empty comment is inserted with the caret in position for typing in the comment. Comments can be edited like strings. There are three levels of comments supported by SEdit: single, double, and triple. Single semicolon comments are formatted at the comment column, about three-quarters of the way across the SEdit window, towards the right margin. Double semicolon comments are formatted at the current indentation of the code that they are in. Triple semicolon comments are formatted against the left margin of the SEdit window. The level of a comment can be increased or decreased by pointing after the semicolon, and either typing another semicolon, or backspacing over the preceding semicolon. Comments can be placed anywhere in your Common Lisp code. However, in Interlisp code, they must follow the placement rules for Interlisp comments.

Strings- "

Enter strings in SEdit by typing a double quote (**"**). SEdit balances the double quotes. When one is typed, SEdit produces a second, with the caret between the two, ready for typing the characters of the string. If a double quote character is typed in the middle of a string, SEdit breaks the string into two smaller strings, leaving the caret between them.

16.1.6 Commands

SEdit commands are most easily entered through the keyboard. When possible, SEdit uses a named key on the keyboard, for example, the DELETE key. The other commands are either Meta, Control, or Meta-Contol key combinations. For the alphabetic command keys, either uppercase or lowercase will work.

There are two menus available, as an alternative means of invoking commands. They are the middle button popup menu, and the attached command menu. These menus are described in more detail below.

16.1.6 Editing Commands

Redisplay: Control-L [Editor Command]

Redisplays the structure being edited.

Delete Selection: DELETE [Editor Command]

Deletes the current selection.

Delete Word: Control-W [Editor Command]

Deletes the previous atom or whole structure. If the caret is in the middle of an atom, deletes backward to the beginning of the atom only.

16.1.7 Completion Commands

Abort: Meta-A [Editor Command]

Aborts. This command must be confirmed. All changes since the beginning of the edit session are undone, and the edit is closed.

The following commands signal completion of an edit session and install the structure you were editing.

Control-X [Editor Command]

Signals the system that this edit is complete. The window remains open, though, so the user can see the edit and start editing again directly.

Control-C [Editor Command]

Signals the system that this edit is complete and compiles the definition being edited. The variable *compile-fn* determines the function to be called to do the compilation. See the Options section below.

Meta-Control-X [Editor Command]

Signals the system that this edit is complete and closes the window.

Meta-Control-C [Editor Command]

Signals the system that this edit is complete, compiles the definition being editing, and closes the window.

16.1.8 Undo Commands

Undo: Meta-U or UNDO [Editor Command]

Undoes the last edit. All changes since the beginning of the edit session are remembered, and can be undone sequentially.

Redo: Meta-R or AGAIN [Editor Command]

Redoes the edit change that was just undone. Redo only works directly following an Undo. Any number of Undo commands can be sequentially redone.

16.1.9 Find Commands

Find: Meta-F or FIND [Editor Command]

Finds a specified structure, or sequence of structures. If there is a current selection, SEdit looks for the next occurrence of the selected structure. If there is no selection, SEdit prompts for the structure to find, and searches forward from the position of the caret. The found structure will be selected, so the Find command can be used to easily find the same structure again.

If a sequence of structures is selected, SEdit will look for the next occurrence of the same sequence. Similarly, when SEdit prompts for the structure to find, you can type a sequence of structures to look for.

The variable *wrap-search* controls whether or not SEdit wraps around from the end of the structure being edited and continues searching from the beginning.

Reverse Find: Control-Meta-F [Editor Command]

Finds a specified structure, searching in reverse from the position of the caret.

The variable *wrap-search* controls whether or not SEdit wraps around from the beginning of the structure being edited and continues searching from the end.

Find Gap: Meta-N or SKIP-NEXT [Editor Command]

Skips to the next gap in the structure, leaving it selected for pending deletion.

Substitute: Meta-S or SHIFT-FIND [Editor Command]

Substitutes one structure, or sequence of structures, for another structure, or sequence, within the current selection. SEdit prompts you in the SEdit prompt window for the structures to replace, and the structures to replace with.

The selection to substitute within must be a structure selection. To get a structure selection, click with the middle mouse button (not the left), and extend it, if necessary, with the right mouse button. If you begin with the left button, you will get an informational message "Select the structure to substitute within", because the selection was of characters, rather than structures.

Delete Structure: Meta-Control-S [Editor Command]

Removes all occurences of a structure or sequence of structures within the current selection. SEdit prompts the user in the SEdit prompt window for the structures to delete.

16.1.9 General Commands

Arglist: Meta-H or HELP [Editor Command]

Shows the argument list for the function currently selected, or currently being typed in, in the SEdit prompt window. If the argument list will not fit in the SEdit prompt window, it is displayed in the main Prompt Window.

Convert Comments: Meta-; [Editor Command]

Converts old style comments in the selected structure to new style comments. This converter notices any list that begins with an asterisk (*) in the INTERLISP package (IL:*) as an old style comment. Section 16.1.11, Options, describes the converter options .

Edit: Meta-O [Editor Command]

Edits the definition of the current selection. If the selected name has more than one type of definition, SEdit asks for the type to be edited. If the selection has no definition, a menu pops up. This menu lets the user specify either the type of definition to be created, or no definition if none needs to be created.

Eval: Meta-E or Do-It [Editor Command]

Evaluates the current selection. If the result is a structure, the inspector is called on it, allowing the user to choose how to look at the result. Otherwise, the result is printed in the SEdit prompt window. The evaluation is done in the process from which the edit session was started. Thus, while editing a function from a break window, evaluations are done in the context of the break.

Expand: Meta-X or EXPAND [Editor Command]

Replaces the current selection with its definition. This command can be used to expand macros and translate CLISP.

Extract: Meta- / [Editor Command]

Extracts one level of structure from the current selection. If the current selection is an atom, or if there is no selection, the next largest structure containing this atom, or caret, is used. This command can be used to strip the parentheses off a list or a comment, or to unquote a quoted structure.

Inspect: Meta-I [Editor Command]

Inspect the current selection.

Join: Meta-J [Editor Command]

Joins. This command joins any number of sequential Lisp objects of the same type into one object of that type. Join is supported for atoms, strings, lists, and comments. In addition, SEdit permits joining of a sequence of atoms and strings, since either type can easily be coerced into the other. In this case, the result of the Join will be an atom if the first object in the selection is an atom, otherwise the result will be a string.

Mutate: Meta-Z [Editor Command]

Mutates. This command allows the user to do arbitrary operations on a LISP structure. First select the structure to be mutated (it must be a whole structure, not an extended selection). When the user presses Meta-Z SEdit prompts for the function to use for mutating. This function is called with the selected structure as its argument, and the structure is replaced with the result of the mutation.

For example, an atom can be put in upper case by selecting the atom and mutating by the function U-CASE. You can replace a structure with its value by selecting it and mutating by EVAL.

Quote: Meta-’
Meta-‘
Meta-,
Meta-.
Meta-@ or Meta-2
Meta-# or Meta-3 [Editor Command]

Quotes the current selection with the specified kind of quote, respectively, Single Quote, Backquote, Comma, Comma-At-Sign, Comma-Dot, or Hash-Quote.

Parenthesize: Meta-) or Meta-0 [Editor Command]

Parenthesizes the current selection, positioning the caret after the new list.

Parenthesize: Meta- (or Meta-9 [Editor Command]

Parenthesizes the current selection, positioning the caret at the beginning of the new list. Only a whole structure selection or an extended selection of a sequence of whole structures can be parenthesized.

16.1.10 Miscellaneous

Change Print Base: Meta-B [Editor Command]

Changes Print Base. Prompts for entry of the desired Print Base, in decimal. SEdit redisplays fixed point numbers in this new base.

Set Package: Meta-P [Editor Command]

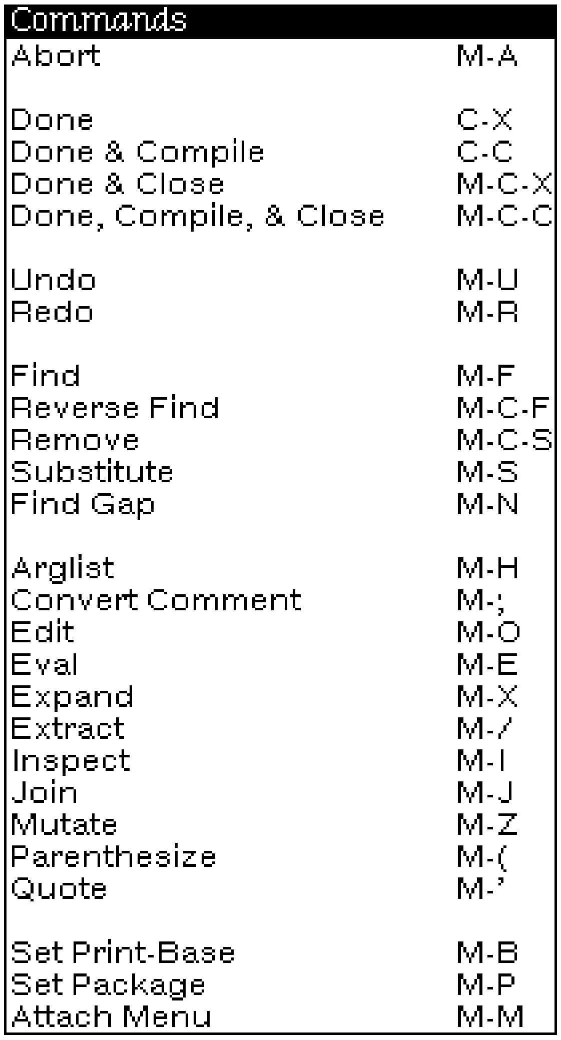
Changes the current package for this edit. Prompts the user, in the SEdit prompt window, for a new package name. SEdit will redisplay atoms with respect to that package.

Attached Menu: Meta-M [Editor Command]

Attaches a menu of the commonly used commands (the SEdit Command Menu) to the top of the SEdit window. Each SEdit window can have its own menu, if desired.

16.1.10 Help Menu

When the mouse cursor is positioned in the SEdit title bar and the middle mouse button is pressed, a Help Menu of commands pops up. The menu looks like this:

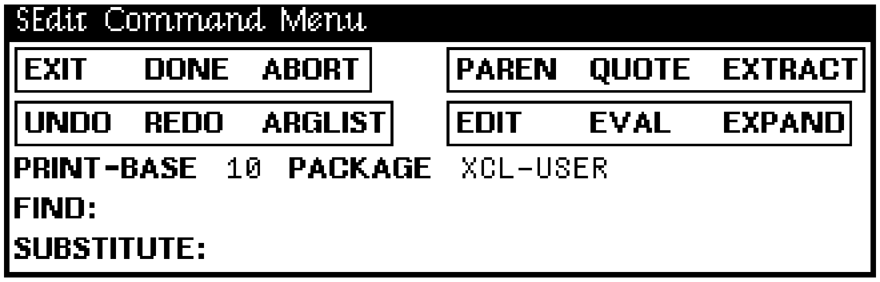


```
Commands
Abort                     M-A

Done                      C-X
Done & Compile            C-C
Done & Close              M-C-X
Done, Compile, & Close    M-C-C

Undo                      M-U
Redo                      M-R

Find                      M-F
Reverse Find              M-C-F
Remove                    M-C-S
Substitute                M-S
Find Gap                  M-N

Arglist                   M-H
Convert Comment           M-;
Edit                      M-O
Eval                      M-E
Expand                    M-X
Extract                   M-/
Inspect                   M-I
Join                      M-J
Mutate                    M-Z
Parenthesize              M-(
Quote                     M-'

Set Print-Base            M-B
Set Package               M-P
Attach Menu               M-M
```

The Help Menu lists each command and its corresponding Command Key. (In the menu, the letter C stands for CONTROL, while M indicates Meta.) The command selected is executed just as if the command had been entered from the keyboard. The menu remembers which command was selected last, and pops up with the mouse cursor next to that same command the next time the menu is used. This provides a very fast way to repeat the same command when using the mouse.

16.1.9 Command Menu

The SEdit Attached Command Menu contains the commonly used commands. Use the Meta-M keyboard command to bring up this menu. The menu can be closed, independently of the SEdit window, when desired. The menu looks like:



All of the commands in the menu function identically to their corresponding keyboard commands, except for Find and Substitute.

When Find is selected with the mouse cursor, SEdit prompts in the menu window, next to the Find button, for the structures to find. Type in the structures then select Find again. The search begins from the caret position in the SEdit window.

Similarly, Substitute prompts, next to the Find button, for the structures to find, and next to the Substitute button for the structures to substitute with. After both have been typed in, selecting Substitute replaces all occurrences of the Find structures with the Substitute structures, within the current selection.

To do a confirmed substitute, set the edit point before the first desired substitution, and select Find. Then if you want to substitute that occurrence of the structure, select Substitute. Otherwise, select Find again to go on.

Selecting either Find or Substitute with the right mouse button erases the old structure to find or substitute from the menu, and prompts for a new one.

16.1.11 SEdit Window Region Manager

SEdit provides user redefinable functions which control how SEdit chooses the region for a new edit window.

(Get-Window-Region *CONTEXT REASON NAME TYPE***)** [Function**]**

This function is called when SEdit wants to know where to place a window it is about to open. This happens whenever the user starts a new SEdit or expands an Sedit icon.The default behavior is to pop a window region off SEdit's stack of regions that have been used in the past. If the stack is empty, SEdit prompts for a new region.

This function can be redefined to provide different behavior. It is called with the edit *CONTEXT*, a *REASON* for needing a region, the *NAME* of the structure to be edited, and the *TYPE* of the structure to be edited. The edit *CONTEXT* is SEdit's main data structure and can be useful for associating particular edits with specific regions. The *REASON* argument specifies why SEdit wants a region, and will be one of the keywords :CREATE or :EXPAND.

(Save-Window-Region *CONTEXT REASON NAME TYPE REGION***)** [Function]

This function is called whenever SEdit is finished with a region and wants to make the region available for other SEdits. This happens whenever an SEdit window is closed or shrunk, or when an SEdit Icon is closed. The default behavior is simply to push the region onto SEdit's stack of regions.

This function can be redefined to provide different behavior. It is also called with the edit *CONTEXT*, the *REASON*, the *NAME*, the *TYPE*, and additionally the window *REGION* that is being released. The *REASON* argument specifies why SEdit is releasing the region, and will be one of the keywords :CLOSE, :SHRINK, or :CLOSE-ICON.

Keep-Window-Region [Variable**]**

Default **T**. This flag determines the behavior of the default SEdit region manager, explained above, for shrinking and expanding windows. When set to **T**, shrinking an SEdit window will not give up that window's region; the icon will always expand back into the same region. When set to **NIL**, the window's region is made available for other SEdits when the window is shrunk. Then when an SEdit icon is expanded, the window will be reshaped to the next available region.

This variable is only used by the default implementations of the functions **Get-Window-Region** and **Save-Window-Region**. If these functions are redefined, this flag is no longer used.

16.1.12 Options

The following parameters can be set as desired.

Wrap-Parens [Variable]

This SEdit pretty printer flag determines whether or not trailing close parenthesis characters,), are forced to be visible in the window without scrolling. By default it is set to NIL, meaning that close parens are allowed to "fall off" the right edge of the window. If set to T, the pretty printer will start a new line before the structure preceding the close parens, so that all the parens will be visible.

Wrap-Search [Variable]

This flag determines whether or not SEdit find will wrap around to the top of the structure when it reaches the end, or vice versa in the case of reverse find. The default is NIL.

Clear-Linear-On-Completion [Variable]

This flag determines whether or not SEdit completely re-pretty prints the structure being edited when you complete the edit. The default value is NIL, meaning that SEdit reuses the pretty printing.

Convert-Upgrade [Variable]

Default 100. When using Meta-; to convert old-style single- asterisk comments, if the length of the comment exceeds **Convert-Upgrade** characters, the comment is converted into a double semicolon comment. Otherwise, the comment is converted into a single semicolon comment.

Old-style double-asterisk comments are always converted into new-style triple-semicolon comments.

16.1.13 Control Functions

(Reset) [Function**]**

This function recomputes the SEdit edit environment. Any changes made in the font profile, or any changes made to SEdit's commands are captured by resetting. Close all SEdit windows before calling this function.

(Add-Command *KEY-CODE FORM &OPTIONAL KEY-NAME COMMAND-STRING HELP-STRING***)** [Function**]**

This function allows you to write your own SEdit keyboard commands. You can add commands to new keys, or you can redefine keys that SEdit already uses as command keys. If you mistakenly redefine an SEdit command, the funtion Reset-Commands will remove all user-added commands, leaving SEdit with its default set of commands.

KEY-CODE can be a character code, or any form acceptible to il:charcode.

FORM determines the function to be called when the key command is typed. It can be a symbol naming a function, or a list, whose first element is a symbol naming a function and the rest of the elements are extra arguments to the function. When the command is invoked, SEdit will apply the function to the edit context (SEdit's main data structure), the charcode that was typed, and any extra arguments supplied in *FORM*. The extra arguments do not get evaluated, but are useful as keywords or flags, depending on how the command was invoked. The command function must return T if it handled the command. If the function returns NIL, SEdit will ignore the command and insert the character typed.

The optional arguments are used to add this command to SEdit's middle button menu. When the item is selected fromthe menu, the command function will be called as described above, with the charcode argument set to NIL.

KEY-NAME is a string to identify the key (combination) to be typed to invoke the command. For example "M-A" to represent the Meta-A key combination, and "M-C-A" for Meta-Control-A.

COMMAND-NAME is a string to identify the command function, and will appear in the menu next to the *KEY-NAME*.

HELP-STRING is a string to be printed in the prompt window when a mouse button is held down over the menu item.

After adding all the commands that you want, you must call Reset-Commands to install them.

For example:

```
(add-command "↑U" (my-change-case t))

(add-command "↑Y" (my-change-case nil))

(add-command "1,r" my-remove-nil

  "M-R" "Remove NIL"

  "Remove NIL from the selected structure"))

(reset-commands)
```

will add three commands. Suppose `my-change-case` takes the arguments *CONTEXT*, *CHARCODE*, and *UPPER-CASE*?. *UPPER-CASE*? will be set to T when `my-change-case` is called from Control-U, and NIL when called from Control-Y. `my-remove-nil` will be called with only *CONTEXT* and *CHARCODE* arguments when Meta-R is typed.

Below are some SEdit functions which are useful in writing new commands.

(Reset-Commands) [Function**]**

This function installs all commands added by **Add-Command**. SEdits which are open at the time of the **Reset-Commands** will not see the new commands; only new SEdits will have the new commands available.

(Default-Commands) [Function**]**

This function removes all commands added by **Add-Command**, leaving SEdit with its default set of commands. As in **Reset-Commands**, open SEdits will not be changed; only new SEdits will have the user commands removed.

(Get-Prompt-Window *CONTEXT***)** [Function**]**

This function returns the attached prompt window for a particular SEdit.

(Get-Selection *CONTEXT***)** [Function**]**

This function returns two values: the selected structure, and the type of selection, one of NIL, T, or :SUB-LIST. The selection type NIL means there is not a valid selection (in this case the structure is meaningless). T means the selection is one complete structure. :SUB-LIST means a series of elements in a list is selected, in which case the structure returned is a list of the elements selected.

(Replace-Selection *CONTEXT STRUCTURE SELECTION-TYPE* **)** [Function**]**

This function replaces the current selection with a new structure, or multiple structures, by deleting the selection and then inserting the new structure(s). The *SELECTION-TYPE* argument must be one of T or :SUB-LIST. If T the *STRUCTURE* is inserted as one complete structure. If :SUB-LIST, the *STRUCTURE* is treated as a list of elements, each of which is insertd.

16.1.13 Programmer's Interface

This programmer's interface to SEdit provides a way to call SEdit directly. This interface is sketchy and will change inthe future.

Getdef-Fn [Variable]

This function is called with the arguments *NAME*, *TYPE*, and *OLDDEF*, when SEdit needs to refetch the definition for the named object being edited. When SEdit is first started it gets passed the structure, so this function doesn't get called. But after completion, SEdit refetches because it doesn't know if the Edit Interface (File Manager) changed the definition upon installation. The function returns the new definition.

Fetch-Definition-Error-Break-Flag [Variable]

This flag, along with the error options listed below, determines what happens when the Getdef-Fn errors. The default value is NIL, causing errors to be suppressed. When set to T, the break will be allowed.

Getdef-Error-Fn [Variable]

This function is funcalled with the arguments *NAME*, *TYPE*, *OLDDEF*, and *PROMPT-WINDOW*, when the Getdef-Fn errors, independent of whether or not the break is suppressed. This function should return the structure to be used in place of the unavailable new definition.

Edit-Fn [Variable]

This function is funcalled with the selected structure as its argument from the Edit (M-O) command. It should start the editor as appropriate, or else generate an error if the selection is not editable.

Compile-Fn [Variable]

This function is funcalled with the arguments *NAME*, *TYPE*, and *BODY*, from the compile completion commands. It should compile the definition, *BODY*, and install the code as appropriate.

(SEdit ***STRUCTURE PROPS OPTIONS*****)** [Function]

This function provides a means of starting SEdit directly. *STRUCTURE* is the structure to be edited.

PROPS is a property list, which may specify the following properties:

:name - the name of the object being edited

:type - the file manager type of the object being edited. If NIL, SEdit will not call the file manager when it tries to refetch the definition it is editing. Instead, it will just continue to use the structure that it has.

:completion-fn - the function to be called when the edit session is completed. This function is called with the *CONTEXT*, *STRUCTURE*, and *CHANGED?* arguments. *CONTEXT* is SEdits main data structure. *STRUCTURE* is the structure being edited. *CHANGED?* specifies if any changes have been made, and is one of NIL, T, or :ABORT, where :ABORT means the user is aborting the edit and throwing away any changes made. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the main arguments above.

:root-changed-fn - the function to be called when the entire structure being edited is replaced with a new structure. This function is called with the new structure as its argument. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the structure argument.

OPTIONS is one or a list of any number of the followng keywords:

:fetch-definition-suppress-errors - If this option is provided, any error under the Getdef-Fn will be suppressed, regardless of the :fetch-definition-allow-errors option or the value of ***Fetch-Definition-Error-Break-Flag***.

:fetch-definition-allow-errors - If this option is provided, any error under the Getdef-Fn will be allowed to break.

:dontwait - This option specifies that the call to **SEdit** should return as soon as the editor is started, rather than waiting for a completion command.

:close-on-completion - This option specifies that SEdit cannot remain active for multiple completions. That is, the SEdit window cannot be shrunk, and the completion commands that normally leave the window open will in this case close the window and terminate the edit.

:compile-on-completion - This option specifies that SEdit should call the *Compile-Fn* to compile the definition being edited upon completion, regardless of the completion command used.

Fixed ARS

AR 7471 ---- You no longer have to Ctrl-X out of SEdit to compile. To compile, select either of two new SEdit commands. The C-C command (Ctrl -C) compiles and leaves the SEdit window open. The M-C-C command (Meta-Ctrl- C) compiles and closes the window.

AR 7783 ---- After typing Meta-O, you are prompted to "Select a type of dummy definition to install." The first option is "Optimizers". This title refers to XCL:DEFOPTIMIZERS, and has nothing to do with CL:OPTIMIZE.

AR 7786 ---- Square bracket characters '[' and ']' are not treated as special characters by SEdit, just as in Common Lisp.

[This page intentionally left blank]

Test script for SEdit version February 24, 1987

SEE CHANGE CONTROL FOR THE FILE SEDIT

FIXED:
7692 M-E breaks on numbers
7682 Ctrl-W with pending delete selection breaks
7705 M-H with extended selection breaks
7717 M-J breaks when nothing selected
7759 forgets package of mutator candidate
7731 M-J breaks with a sequence of numbers
7509 SEDIT.RESET needs IP fonts
7576 SEDIT.RELOAD shoud not be part of SEdit

Select a number and hit M-E, the result will be the number.

CTRL-W when there is a pending delete selection will do nothing. The easy test case is single-quote then ctrl-w.

Make an extended selection and hit M-H (help). Sedit will now say "Select function you want the arguments for"

Just after Starting SEdit, or any other time nothing is selected and/or there isno point, M-J used to break. Will now say "Select items to join."

SEdit now remembers the package of the mutate function candidate. Type in the atom abc. Get into the LISP package (M-P, LISP). Select the atom abc. Hit M-Z, and type IL:U-CASE, cr. Now hit M-Z again, the old SEdit would prompt with U-CASE, the new sedit will prompt with IL:U-CASE.

Select a sequence of numbers, or any sequence with a number first. Hit M-J. SEdit will say "Can't join numbers". SEdit will join AB 123 CD, though, as it used to.

With INTERPRESSFONTDIRECTORIES set to NIL, call SEDIT.RESET. It won't try to create IP fonts, and thus won't break.

The function SEDIT.RELOAD is not longer in the sysout.

,. wasn't recognized as a quote type. (SEDIT '(BQUOTE (A (\,. FOO) B] used to display as '(A (\,. FOO) B) and now will display as '(A ,.FOO B).

And now for all the shift selecting stuff:

Weren't able to Move select a structure into a pending delete selection. Now you can unless the structure overlaps the pending delete selection in some way. SEdit will say "Can't move a structure which overlaps the selection" if you try.

Can move select an object out of a quote, into any destination (exec, same sedit, different sedit), and the object will be replaced by a gap, which is selected pending delete.

Shift selecting something into a string didn't use to be completely undoable. Try Move selecting an atom into a string, then hit undo.

Move selecting something into a different destination when there is a main selection used to leave the selection messed up. Now it doesn't.

TWODINSPECTOR

By: Jan Pedersen (Pedersen.PA @ Xerox.ARPA)

The TWODINSPECTOR package provides a two-dimensional inspector window abstraction, very similar in form to the standard one-dimensional inspector but laid out in rows and columns, instead of just rows.

The top level function is TWODINSPECTW.CREATE

(TWODINSPECTW.CREATE *DATUM ROWPROPS COLUMNPROPS FETCHFN ˆ*
STOREFN VALUECOMMANDFN ROWPROPCOMMANDFN ˆ
COLUMNPROPCOMMANDFN TITLE TITLECOMMANDFN
WHERE TOPRIGHT) [Function]

Datum is the object to be inspected. Rowprops is a list of properties of the datum which will be laid out vertically, or a function which will be called with datum as an argument and returns such a list. Similarly, columnprops is a list of properties of the datum which will be laid out horizontally, or a function which will be called with datum as an argument and returns such a list. Each pair (rowprop, columprop) specifies a cell of the twodimensional inspector window. Fetchfn is a function which if called with arguments datum, rowprop, and columprop returns the value in that cell. Storefn is a function which if called with arguments newvalue, datum, rowprop, and columprop stores newvalue in the cell.

The cells of the inspector window are selectable. If valuecommandfn is given, it must be a function which will be called with arguments cellvalue, rowprop, columnprop, datum, and twodinspectwindow when the cell specified by (rowprop, columnprop) is selected. A default valuecommandfn is provided which allows the cellvalue to be inspected, set, or bound to the litatom IT.

Similarly the rowprops and the columnprops themselves are selectable. If rowpropcommandfn is given it must be a function which will be called with args rowprop, datum, and twodinspectwindow when rowprop is selected. If columnpropcommandfn is given it must be a function which will be called with args columnprop, datum, and twodinspectwindow when columnprop is selected. No default rowpropcommandfn or columnpropcommandfn is provided. If rowpropcommandfn is not given, the rowprops will not be selectable. Similarly, If columnpropcommandfn is not given, the columnprops will not be selectable.

Title will be the title for the window -- a default is provided. Titlecommandfn is a function which will be called with the single argument twodinspectwindow if the middle button is depressed in the title bar of the window.

Where may be a window, in which case it will be used as at least part of the twodinspector (the twodinspector is composed of five window), This is especially useful if where is the result of a previous call to TWODINSPECTW.CREATE. The dimensions of where will not be used to position the twodinspector unless topright is NIL.

Where may also be region or a position specifying the lower left hand corner of the twodinspector. If where is NIL, the user will be prompted for a position.

Topright allows the user to specify the top right-hand corner of the twodinspector. Topright must be a position, and if given overrides any specification which may have been provided by the argument where.

Returns the main window of an attached window group.

The arguments to TWODINSPECTW.CREATE are cached on windowprops of the same name on the returned main window.

Several functions are provided for use in the various command functions.

(TWODINSPECT.REDISPLAY *TWODINSPECTW SOMEROWPROPS* ˆ
SOMECOLUMNPROPS) [Function]

Redisplay selected cells of twodinspectw. Somerowprops may either be a single rowprop, a list of rowprops, or NIL. Somecolumnprops may either be a single columnprop, a list of columnprops, or NIL. If either are NIL the entire twodinspectw is recomputed and redisplayed. Otherwise, the cells specified by the cross product of somerowprops and somecolumnprops are redisplayed, possibly forcing the entire twodinspectw to redisplay if the printed representation of a cell overflows its column width.

(TWODINSPECT.REPLACE *TWODINSPECTW ROWPROP COLUMNPROP* ˆ
NEWVALUE) [Function]

Replaces the cell specified by (rowprop, columprop) with newvalue and updates the display.

(TWODINSPECT.SELECTITEM *TWODINSPECTW ROWPROP COLUMNPROP*] [Function]

Selects the cell specified by (rowprop, columprop). That cell is inverted and put on the window prop SELECTION of twodinspectw. If either of rowprop or columprop is NIL, then the current selection is simply deselected.

(TWODINSPECT.SELECTROWPROP *TWODINSPECTW ROWPROP*] [Function]

Selects rowprop. If rowprop is NIL, then the currently selected rowprop is deselected.

(TWODINSPECT.SELECTCOLUMNPROP *TWODINSPECTW* COLUMNPRO*P*] [Function]

Selects columprop. If rowprop is NIL, then the currently selected columprop is deselected.

Note: there is no provision for redisplaying selected row or column props -- although this may be effected by redisplaying the entire twodinspectw.

Since the Twodinspector windows differ stylistically from the standard inspector windows, a stylistically similar onedinspector window is also provided.

(ONEDINSPECTW.CREATE *DATUM PROPS FETCHFN STOREFN VALUECOMMANDFN* ˆ
PROPCOMMANDFN TITLE TITLECOMMANDFN WHERE TOPRIGHT) [Function]

Datum is the object to be inspected.Props is a list of properties of the datum which will be laid out horizontally, or a function which will be called with datum as an argument and returns such a list. Each prop specifies a cell of the onedimensional inspector window. Fetchfn is a function which if called with arguments datum, and prop returns the value in that cell. Storefn is a function which if called with arguments newvalue, datum, and prop stores newvalue in the cell.

The cells of the inspector window are selectable. If valuecommandfn is given, it must be a function which will be called with arguments cellvalue, prop, datum, and onedinspectwindow when the cell specified by prop is selected. A default valuecommandfn is provided which allows the cellvalue to be inspected, set, or bound to the litatom IT.

Similarly the props themselves are selectable. If propcommandfn is given it must be a function which will be called with args prop, datum, and onedinspectwindow when prop is selected. No default propcommandfn is provided. If propcommandfn is not given, the props will not be selectable.

Title will be the title for the window -- a default is provided. Titlecommandfn is a function which will be called with the single argument onedinspectwindow if the middle button is depressed in the title bar of the window.

Where may be a window, in which case it will be used as at least part of the onedinspector (the onedinspector is composed of three window), This is especially useful if where is the result of a previous call to ONEDINSPECTW.CREATE or TWODINSPECTW.CREATE. The dimensions of where will not be used to position the onedinspector unless topright is NIL.

Where may also be region or a position specifying the lower left hand corner of the onedinspector. If where is NIL, the user will be prompted for a position.

Topright allows the user to specify the top right-hand corner of the onedinspector. Topright must be a position, and if given overrides any specification which may have been provided by the argument where.

Returns the main window of an attached window group.

The arguments to ONEDINSPECTW.CREATE are cached on windowprops of the same name on the returned main window.

(ONEDINSPECT.REDISPLAY *ONEDINSPECTW SOMEPROPS*) [Function]

Redisplay selected cells of onedinspectw. Someprops may either be a single prop, a list of props, or NIL, in which case the entire onedinspectw is recomputed and redisplayed. Otherwise, the cell(s) specified by the someprops are redisplayed, possibly forcing the entire someprops to redisplay if the printed representation of a cell overflows the column width.

(ONEDINSPECT.REPLACE *ONEDINSPECTW PROP NEWVALUE*) [Function]

Replaces the cell specified by prop with newvalue and updates the display.

(ONEDINSPECT.SELECTITEM *ONEDINSPECTW PROP*) [Function]

Selects the cell specified by prop. That cell is inverted and put on the window prop SELECTION of onedinspectw. If prop is NIL, then the current selection is simply deselected.

(ONEDINSPECT.SELECPROP *ONEDINSPECTW PROP*) [Function]

Selects prop. If prop is NIL, then the currently selected prop is deselected.