HARMONY RELEASE NOTES

January 1985

TABLE OF CONTENTS

[This page intentionally left blank]

INTRODUCTION

These notes document features of the Harmony release of Interlisp-D. Harmony is the successor to Carol, the June 1984 release of Interlisp-D.

Harmony is substantially more reliable than its predecessors: over 600 system bugs have fixed. Support for NS file and print servers is now robust and reliable. Major improvements have been made to the font system. An advanced version of Tedit, the Interlisp text editor, is being released with Harmony. Image streams, which allow for the printing of arbitrary text and graphics on Interpress and Press printers, are supported by Harmony.

The following pages present detailed descriptions of these, and many other features, which constitute the Harmony release.

INPUT/OUTPUT

- **Advance Warning: Changes to OPENFILE; multiple streams per file**

At some point in the future, the Interlisp-D i/o system will change so that each call to OPENFILE returns a distinct stream. This differs from the current behavior, inherited from Interlisp-10, that there can only be one stream open on any file, and that a second OPENFILE (assuming both are for INPUT) will return the same "opening". This change is required in order to deal rationally with files in a multiprocessing environment.

This change will of necessity produce the following incompatibilities:

1) OPENFILE will return a STREAM, not a full file name. To make this less confusing, STREAMs will have a print format that reveals the underlying file's actual name, and the functions UNPACKFILENAME and FILENAMEFIELD, when given a STREAM, will operate on the stream's name.

2) A greater penalty will ensue for passing as the FILE argument to i/o operations anything other than the object returned from OPENFILE. Passing the file's name will be significantly slower than passing the stream (even when passing the "full" file name), and in the case where there is more than one stream open on the file it might even act on the wrong one.

Advice for planning for this change:

Users are encouraged to write code which binds a variable to the result of OPENFILE and passes that variable to all i/o operations; such code will likely continue to work. Similar code that will work less well, if at all, is that which binds a variable to the result of an INFILEP and passes that to OPENFILE and all i/o operations; such code works well now, but implicitly assumes that INFILEP and OPENFILE return the same thing, an invalid assumption in this future world. (Code that passes incomplete file names to i/o operations is incurring a substantial performance penalty even now, and should have been changed long ago to use the result of the OPENFILE.)

To see more directly the effects of passing around STREAMs instead of file names, replace your calls to OPENFILE with calls to OPENSTREAM. OPENSTREAM is called in exactly the same way, but returns a STREAM. Streams can be passed to READ, PRINT, CLOSEF, etc just as the file's full name can be currently, but using them is more efficient. The function FULLNAME, when applied to a stream, returns its full file name.

- **New function OPENSTRINGSTREAM: access strings like files**

Interlisp-D inherited a feature from Interlisp-10 such that if a string was given as the file argument to an input function (READ, READC, etc.), that the function would interpret the string as the contents of a file and read the characters of the string. However, this never was a very clean design, and it interferes with the desire to use strings as file names. The following function has been created to handle i/o operations from/to strings more cleanly.

(OPENSTRINGSTREAM STR ACCESS)
Returns a stream that can be used to access the characters of the string STR. ACCESS may be either INPUT, OUTPUT, or BOTH; NIL defaults to INPUT. The stream returned may be used exactly like a file opened with the same access, except that output operations may not extend past the end of the original string. Also, string streams do not appear in the value of (OPENP).

- **Advance Warning: (READ <string>) will no longer read string characters in future release**

In the current release, (READ <string>) continues to work as before. However, in some future release, this feature will be decommissioned, and OPENSTRINGSTREAM will be the ONLY way to treat a string as a file. Users who depend on the old feature are encouraged to change their code now.

- **New function COPYCHARS for copying with EOL convention**

Many parts of the system have been changed to automatically convert between different EOL conventions. COPYFILE, MAKEFILE, and Tedit have been so modified, but we can't claim that every possible case has been taken care of. For user programs, the following function is available to do this conversion automatically.

(COPYCHARS SRCFIL DSTFIL START END)
This is like COPYBYTES, except that it performs the proper conversion if the EOL conventions of SRCFIL and DSTFIL are not the same. START and END are interpreted as byte specifications in SRCFIL. The number of bytes actually output to DSTFIL might be more or less than the number of bytes specified by START and END, depending on what the EOL conventions are. In the case where the EOL conventions happen to be the same, COPYCHARS simply calls COPYBYTES

- **1100/1108 Parallel Port functions**

The 1100 has a parallel port connector with 8 bidirectional data lines, 8 unidirectional output lines, and 5 unidirectional input lines. The 1108 with Extended Processor Option (CPE) has a similar parallel port connector: the differences are (1) it has 6 unidirectional input lines vs. 5, (2) the power lines of the connector are 5 volts vs. 12, and -5, and (3) the pin layouts are different. The Interlisp functions WRITEPRINTERPORT and READPRINTERPORT are available for accessing the parallel port. For more information, see Appendix C (1100 & 1108 CPE Parallel Port).

- **"The infamous PEEKC bug" has been fixed: can backspace over PEEKC-ed chars**

This was a longstanding bug in the Interlisp-D keyboard reader such that following (PEEKC T), the user couldn't backspace over the character that was peeked. This affected a number of functions, such as ASKUSER, FILES? and COMPILE, which peek at the first character of the user's typein.

- **Directory enumeration faster with multiple properties.**

The directory enumeration code has been redone so that remote files do not have to be looked-up repeatedly when accessing multiple properties of files (size, author, etc). This improves the performance of DIR and the file browser.

- **DIRECTORY pattern interpretation improved**

DIRECTORY and FILDIR have been modified to provide a consistent meaning for omitted fields in the file name pattern. Unspecified fields in the pattern default to *, except when the preceding field delimiter is included, in which case the field is explicitly null. Null version is interpreted as "highest". Thus:

DIR * = DIR *.* = DIR *.*;*
enumerates everything.

DIR *. = DIR *.;*
enumerates all versions of files with null extension.

DIR *.;
enumerates highest version of files with null extension.

DIR *.*;
enumerates highest version of everything.

- **Note:** Some hosts/devices are not capable of supporting "highest version" in enumeration. Such hosts instead enumerate ALL versions.

- **WHENCLOSE operations called when streams are closed**

Previously, the WHENCLOSE operations of a file, if any, were only invoked if CLOSEF was called with the file's name, not if called with the stream.

- **COPYFILE now uses FTP protocol whenever possible**

Previously, COPYFILE used the Leaf protocol when copying from a file server that also implemented Leaf. Using the FTP protocol for such file transfers is much more efficient for some servers.

- **COPYFILE infers file type when source file has none**

COPYFILE always tries to create the new file with the same file type as the original file. If the original file's file type is unknown, COPYFILE now infers the type (file is BINARY if any of its 8-bit bytes have their high bit on). Previously, COPYFILE used the value of DEFAULTFILETYPE (initially TEXT), which was often the wrong thing to do, for example, when copying DCOM files.

- **COPYFILE to protected directory succeeds after password given**

Previously, COPYFILE to a protected directory caused a "FILE WON'T OPEN" error even after asking for and receiving the correct password. This problem occurred only for "sequential" files written to a PupFtp server, not for ordinary MAKEFILEs.

- **COPYFILE to {CORE} or {DSK} copies file creation date**

Previously, when copying a file to {CORE} or {DSK}, COPYFILE would ignore the old file's creation date and instead assign the current date and time as the new file's creation date. (COPYFILE to a remote file server has always copied the creation date correctly.)

Defining a user interrupt char no longer turns off previously defined user interrupts

- **INTERRUPTCHAR user interrupts always "soft", no longer do CLEARBUF and FLASHWINDOW**

In Interlisp-D, user interrupts set with INTERRUPTCHAR are always "soft", but are also "immediate", i.e., executing the interrupt does not disturb the process that is running or unwind the stack, but will happen at the next (interruptable) moment. Interrupts no longer clear the input buffer and flash the screen; users that want that behavior should explicitly call CLEARBUF and FLASHWINDOW as appropriate.

- **Control-C no longer calls RAID**

Control-C no longer calls RAID. For users who like to be able to use this low-level interrupt (more useful on 1100 than on 1108), it can be reenabled by executing (INTERRUPTCHAR 3 'RAID).

- **Shift-BS *NOT* equivalent to control-W**

The key Shift-BS has been changed so it is no longer equivalent to control-W in the initial Interlisp loadup. Users can change this by the appropriate call to KEYACTION, e.g. (KEYACTION 'BS '((8 23 NOLOCKSHIFT))) will restore the previous behavior.

- **UNPACKFILENAME works with strange file names: A.B.C**

This is useful when accessing file servers which do not conform to Interlisp's file name conventions, such as NS file servers and UNIX-based file servers.

- **OPENFILE <Unix Leaf Server File> does not return filename with version ;0**

In some versions of the Unix leaf server code, for files without version numbers OPENFILE returns a filename with version zero. If this file name is then passed to OPENFILE again, it fails. Now, Interlisp explicitely looks for that situation, and strips off the version number entirely.

- **Known Bug: Unix FTP server returns file names like "FOO;1."**

Some versions of the Unix Ftp Server have a bug that causes DIR to print the names of extensionless files as, say, "FOO;1." (with a period AFTER the version number).

- **1100/1132 {DSK} device supports file types**

Files created by Interlisp on the 1100/1132 local file system now have TYPE information saved, where TYPE = TEXT or BINARY. Files written outside of Interlisp have TYPE = NIL.

- **Expanded documentation for RENAMEFILE**

(RENAMEFILE OLDFILE NEWFILE)
Renames OLDFILE to be NEWFILE. Causes an error, FILE NOT FOUND if OLDFILE does not exist. Returns the full name of the new file, if successful, else NIL if the rename cannot be performed. In the general case (e.g., when OLDFILE and NEWFILE are on different devices), RENAMEFILE works by copying OLDFILE to NEWFILE and then deleting OLDFILE.

- **Documentation correction: METASHIFT arg has to be T**

The reference manual is incorrect when it says that the FLG argument to METASHIFT can be any non-NIL value. To work correctly, FLG must be T: (METASHIFT T). Other non-NIL values are passed as the ACTIONS argument to KEYACTION. The reason for this is that if someone has set Blank-bottom to some random behavior, then (RESETFORM (METASHIFT T) --) will correctly restore that random behavior.

- **(CLOSEF <display-stream>) is a no-op**

- **GETECHOMODE checks its argument type**

GETECHOMODE will now generate an "ILLEGAL TERMINAL TABLE" error if it is passed an argument that is not a legal terminal table. Previously, it would not check its argument, and cause a more serious error if it was not a terminal table.

- **GETRAISE causes error if given bad terminal table argument, instead of calling RAID**

PRINTING

- **Hardcopy functions cleaned up, documented**

In previous releases, the functions and variables used to send files to various printers have been redesigned repeatedly. We have been trying to design a simple interface that would "do the right thing" for most users, but would also allow users to get around the defaults when necessary. It was also important to provide facilities so users could define their own printers, and hook them into the normal hardcopy functions.

In the Harmony release, the hardcopy facilities have been simplified considerably. Files and bitmaps can be sent to the printer using the functions SEND.FILE.TO.PRINTER and HARDCOPYW. The variable DEFAULTPRINTINGHOST contains information about the available printers, and the variables PRINTERTYPES and PRINTFILETYPES contain the the information necessary to print a file on any given printer. For full documentation, see Appendix A (Hardcopy Facilities).

- **Image streams allow printing arbitrary text and graphics on Press or Interpress printers**

Previously, the only documented way of printing text and graphics on Press or Interpress printers was to use one of the supported tools, such as Tedit. While these tools are sufficient for many needs, there was a need for functions that users could call from their programs to print arbitrary text and graphics. As part of a long-range effort to provide a simple, device-independent interface to the various graphics display routines, "image streams" were created.

An image stream is an output stream which "knows" how to process graphic commands. It can be passed as the FILE/STREAM argument to the ordinary character-output functions (PRINT, etc.) and to the graphic functions as well (DSPXPOSITION, DRAWCIRCLE, etc.). Some image streams, such as display and local-printer streams, may simply execute the appropriate operations to cause the desired image to appear immediately on the output medium. Other image streams (PRESS, INTERPRESS, etc.) interpret the graphic commands by saving information in a file of the appropriate format. If this file is on the {LPT} device, it will automatically be transmitted to a printer device when it is closed by CLOSEF. Non-LPT files can be transmitted later by explicit calls to LISTFILES and SEND.FILE.TO.PRINTER.

Image streams are created by the following function:

(OPENIMAGESTREAM FILE IMAGETYPE OPTIONS)
Opens and returns an output stream of type IMAGETYPE on a destination specified by FILE. IMAGETYPE can currently be either PRESS, INTERPRESS, or DISPLAY. Eventually, other image types will be implemented for other devices. FILE can name a file either on a normal storage device or on a printer device. In the latter case, the file is sent to the printer when the stream is closed.

FILE = NIL is equivalent to FILE = {LPT}. Names for printer files are of the form {LPT}PRINTERNAME.TYPE, where PRINTERNAME, TYPE, or both may be omitted. PRINTERNAME is the name of the particular printer to which the file will be transmitted on closing; it defaults to the first printer on DEFAULTPRINTINGHOST that can print IMAGETYPE files. The TYPE extension supplies the IMAGETYPE when it is defaulted (see below). OPENIMAGESTREAM will generate an error if the specified printer does not accept the kind of file specified by IMAGETYPE.

If IMAGETYPE is NIL, the image type is inferred from the extension field of FILE and the EXTENSIONS properties in the list PRINTFILETYPES. Thus, a PRESS extension denotes a Press-format stream, while IP, IPR, and INTERPRESS indicate Interpress format. If FILE is a printer file with no extension (of the form {LPT}PRINTERNAME), then IMAGETYPE will be the type that the indicated printer can print. If FILE has no extension but is not on the printer device {LPT}, then IMAGETYPE will default to the type accepted by the first printer on DEFAULTPRINTINGHOST.

Example: Assuming that IP: is an Interpress printer, P is a Press printer, and DEFAULTPRINTINGHOST is (IP: P):

(OPENIMAGESTREAM)
Returns an Interpress image stream on printer IP:

(OPENIMAGESTREAM NIL 'PRESS)
Returns a Press stream on P

(OPENIMAGESTREAM '{LPT}.INTERPRESS)
Returns an Interpress stream on IP:

(OPENIMAGESTREAM '{CORE}FOO.PRESS)
Returns a Press stream on the file {CORE}FOO.PRESS

If IMAGETYPE is DISPLAY, then the user is prompted for a window to open. The file name in this case will be used as the title of the window.

OPTIONS is a list in property list format that may be used to specify certain attributes of the image stream; not all attributes are meaningful or interpreted by all types of streams. Among the properties are:

REGION	Value is the region on the page (in stream scale units, 0,0 being the lower-left corner of the page) that text will fill up. It establishes the initial values for DSPLEFTMARGIN, DSPRIGHTMARGIN, DSPBOTTOMMARGIN (the point at which carriage returns cause page advancement) and DSPTOPMARGIN (where the stream is positioned at the beginning of a new page).
FONTS	Value is a list of fonts that are expected to be used in the stream. Some streams (e.g. Interpress) are more efficient if the expected fonts are called out in advance, but this is not necessary. The first font in this list will be the initial font of the stream, otherwise the DEFAULTFONT for that image type will be used.
HEADING	The heading to be placed automatically on each page, NIL means no heading.

Other functions that are part of the device-independent graphics interface:

(IMAGESTREAMP X IMAGETYPE)
Returns X (possibly coerced to a stream) if it is an output image stream of type IMAGETYPE (or of any type if IMAGETYPE = NIL), otherwise NIL.

(IMAGESTREAMTYPE STREAM)
Returns the image type of STREAM.

(DSPSCALE SCALE STREAM)
Returns the scale of the image stream STREAM, a number indicating how many units in the streams coordinate system correspond to one screen point. For example, DSPSCALE returns 1 for display streams, and 35.27778 for Press and Interpress streams (the number of micas per screen point). In order to be device-independent, user graphics programs must either not specify position values absolutely, or must multiply absolute screen-point quantities by the DSPSCALE of the destination stream. (The SCALE argument to DSPSCALE is currently ignored; in future releases it will enable the scale of the stream to be changed under user control, so that the necessary multiplication will be done internal to the stream interface).

Note: Not all graphics operations can be properly executed for all image types. Currently, only display streams support BITBLT, FILLCIRCLE, and the dashing argument to DRAWCURVE. This functionality is still being developed, but even in the long run some operations may be beyond the physical or logical capabilities of some devices or image file formats. In these cases, the stream will approximate the specified image as best it can.

- **Can preview hardcopy on display using MAKEHARDCOPYSTREAM**

The fonts used in the printers are not exactly the same as the display fonts, because low-resolution screen fonts don't look good when printed on high-resolution printers. In particular, the character widths are not the same (even when scaled to take account of the printer resolution). Because of this, it is difficult to format text on the display so that it is EXACTLY where you want it, since it will be slightly different when printed. In order to allow users to "preview" hardcopy without actually printing it, the following functions are useful:

(MAKEHARDCOPYSTREAM DISPLAYSTREAM IMAGETYPE)
Changes the display stream so that measurements of character widths are consistent with the hardcopy device IMAGETYPE (PRESS, INTERPRESS, etc.). This is useful for seeing on the screen how an image will look when it is hardcopied. Caveat: This doesn't work for TEdit windows.

(UNMAKEHARDCOPYSTREAM DISPLAYSTREAM)
Changes a "hardcopy display stream" back into a regular display stream.

Note: When printing to a "hardcopy display stream", the text will not look as good as it will when printed. In particular, the characters may look crunched together. However, it accurately displays the relative positions of the letters, for formatting purposes.

Can print bitmaps on PRESS printers

Bitmaps can be printed on press printers using HARDCOPYW, or by inserting bitmaps into Tedit documents. If the bitmap is too large for the press printer to handle (for example, if you try to print a complete screen image), it is clipped.

- **HARDCOPYW sends bitmaps to both PRESS and FULLPRESS printers**

HARDCOPYW now goes through FULLPRESSBITMAP when going to a full press printer. The function PRESSBITMAP uses the CLIPPINGREGION argument for clipping, while FULLPRESSBITMAP recognizes the SCALEFACTOR argument.

- **LISTFILES automatically detects and prints formatted Tedit files**

- **HARDCOPYW not a no-op if DEFAULTPRINTERTYPE = NIL**

- **Better error message printed if DEFAULTPRINTINGHOST is NIL**

Previously, attempting a hardcopy operation with DEFAULTPRINTINGHOST = NIL would produce an obscure low-level error.

- **HARDCOPYW has new arg: PRINTERTYPE**

By default, HARDCOPYW will create an Interpress file if there are any Interpress printers on DEFAULTPRINTINGHOST. This default can be changed by passing PRESS as the PRINTERTYPE argument to HARDCOPYW.

- **HARDCOPY in the background menu does not reposition the cursor**

[This page intentionally left blank]

- **Incompatible Change: New font directory variables.**

Previously, Interlisp used a confusing group of variables
(FONTDIRECTORIES, NSFONTDIRECTORIES,
NSFONTWIDTHSDIRECTORIES, STARFONTDIRECTORIES,
FONTWIDTHSFILES)
to determine where to search for font bitmap files and font widths files. These variables have been removed, and a new, rationally-named, set has been introduced:

DISPLAYFONTDIRECTORIES
Value is a list of directories searched to find font bitmap files for display fonts.

DISPLAYFONTEXTENSIONS
Value is a list of file extensions used when searching DISPLAYFONTDIRECTORIES for display fonts. Currently, Interlisp can read "STRIKE" and "AC" display font file formats. Eventually, all Interlisp display fonts will be distributed with the extension .DISPLAYFONT. Therefore, this variable should be initialized to (DISPLAYFONT STRIKE AC). Note that the extension on the file is used to locate the file, but once the file is found Interlisp looks inside it to determine what format it is and how to read it. The function (FONTFILEFORMAT FILE LEAVEOPEN) returns the format of a font file (AC, STRIKE, etc.).

INTERPRESSFONTDIRECTORIES
Value is a list of directories searched to find font widths files for Interpress fonts. These files must have the extension "WD".

PRESSFONTWIDTHSFILES
Value is a list of files (not directories) searched to find font widths files for press fonts. Press font widths are packed into large "FONTS.WIDTHS" files.

All of these variables must be set before Interlisp can auto-load font files. These variables should be initialized in the site-specific INIT file.

- **Incompatible Change: FONTDESCRIPTOR datatype changed; Cannot load fonts dumped with UGLYVARS in Carol**

Between the Carol and Harmony releases, the system datatype FONTDESCRIPTOR was changed, to add a few more fields. Normally, changes to system datatypes do not affect users very much: they just have to recompile old files which use the datatypes. However, in the case that users saved Carol display fonts on files using the UGLYVARS file package command, more care is required to update these files to Harmony.

The problem is that datatype objects put on files with UGLYVARS contain the definition of the datatype. If a Carol file was loaded into Harmony which redefined a system datatype such as FONTDESCRIPTOR, Interlisp would almost certainly crash. In order to prevent people accidently redefining the FONTDESCRIPTOR datatype, the file package has been changed so that trying to change a datatype declaration while reading in an UGLYVARS object causes an error.

If users have created display fonts in Carol that they wish to use in Harmony, the upgrade procedure is the following: (1) While running the Carol release, load the lispusers package EDITFONT.DCOM. (2) Use the function (WRITESTRIKEFONTFILE <fontdescriptor> <filename>) to save each font descriptor as a "strike" format file. Note that strike files do not contain information about the font family, size, etc, so give the strike files descriptive names: e.g., GREEK10B.STRIKE. (3) While running the Harmony release, load EDITFONT.DCOM (note: the same package has been tested to work with both Carol and Harmony). (4) Use the function (READSTRIKEFONTFILE <family> <size> <face> <file>) to read in the strike file, and create a fontdescriptor with the specified family name, face, etc.

Note: It is recommended that user-created display fonts be stored as strike fonts, rather than stored as font descriptors on lisp files. If the files are named similarly to strike files distributed with Interlisp, and put on the same directories, they can be used like any other font.

- **Fontclasses are first-class data objects**

Fontclasses have been introduced as a first-class data object which contains a set of related fonts for different devices. The font functions accept fontclasses, from which they extract the appropriate font for their device. The normal font class variables (DEFAULTFONT, CLISPFONT, etc.) are initialized to fontclass objects. Fontclasses are created and manipulated with the following functions:

(FONTCLASS NAME FONTLIST CREATEFORDEVICES)
Returns a new fontclass object with the name NAME and the device font components specified by FONTLIST, which should be a list of the form (<displayfont> <pressfont> <interpressfont> <otherfont1> <otherfont2> ...). <otherfontN> should be a list of the form (<devicename> <font>). Each of the fonts in FONTLIST may be either a font descriptor, or a "font specification list" that FONTCREATE would accept. CREATEFORDEVICES is a list of the devices for which the fonts should be automatically created. Otherwise, the fonts are not actually created until they are accessed. Note: if a display font is specified in FONTLIST, it is always created.

(FONTCLASSCOMPONENT FONTCLASS DEVICE FONT NOERRORFLG)

Returns the font component of the fonctclass for the device DEVICE (DISPLAY, PRESS, INTERPRESS, etc.). If FONT is non-NIL, the specified component is replaced. If NOERRORFLG is non-NIL, FONTCLASSCOMPONENT return NIL if the component is unspecified in the fontclass, rather than causing an error.

Note: Because font classes are no longer represented by lists, old code which accesses the components of a font class with CAR, CADR, etc. will not work, and must be changed.

- **Font functions take font in many forms**

The font functions have been extended to take fonts specified in a variety of different ways. CHANGEFONT, DSPFONT, FONTCREATE, etc. can be applied to fontclasses, font descriptors, and "font lists" such as '(GACHA 10). The printout command ".FONT" has also been extended to accept fonts specified in any of these forms.

- **FONTCREATE accepts streams for DEVICE argument**

The function FONTCREATE has been extended so that the DEVICE argument can be an image stream, not just an image type. If a stream is given, the result will be a font appropriate for that stream.

- **New FONTPROP properties: SCALE, SPEC, DEVICESPEC**

The function FONTPROP has also been extended to recognize the new properties SCALE, SPEC, and DEVICESPEC. The value of the SCALE property is the units per screen-point in which the font is measured. For example, this is 35.27778 (the number of micas per screen point) for Press and Interpress fonts, which are measured in terms of micas. The value of the SPEC property is the full specification of the font as it is known to Interlisp, a family-size-face-rotation-device quintuple. The value of the DEVICESPEC property is the same as the value of the SPEC property, unless the system has had to coerce the font to another name to find the most appropriate rendering on a specific printing device.

- **New font function: FONTSAVAILABLE**

This function allows programs to determine what fonts are available for different devices.

(FONTSAVAILABLE FAMILY SIZE FACE ROTATION DEVICE CHECKFILESTOO?)
Returns a list of fonts that match the given specification. FAMILY, SIZE; FACE and DEVICE are the same as for FONTCREATE. Additionally, any of them can be the wildcard atom "*", in which case all values of that field are matched. In systems with several font directories, wildcard searches may take a while.

If CHECKFILESTOO? is NIL, only fonts already loaded into virtual memory will be considered. If CHECKFILESTOO? is non-NIL, the font directories for the specified device will be searched. When checking font files, the ROTATION is ignored.

Note: The search is conditional on the status of the server which holds the font. Thus a file server crash may prevent FONTCREATE from finding a file that an earlier FONTSAVAILABLE returned.

Each element of the list returned will be of the form (Family Size Face Rotation Device). For example:

(FONTSAVAILABLE 'MODERN 10 'MRR 0 'DISPLAY)

will return ((MODERN 10 (MEDIUM REGULAR REGULAR) 0 DISPLAY)) if Modern10 for the display is in virtual memory; NIL otherwise.

(FONTSAVAILABLE '* 14 '* '* 'INTERPRESS T)

will return a list of all the size 14 interpress fonts available either loaded into virtual memory or in the font directories.

- **SEE starts printing in correct font**

Previously, if the default font of the exec window was changed, then SEE of an Interlisp source file would start out printing in that font (until the next font change). Now, SEE resets the font at the beginning of printing an Interlisp source file.

- **EDITCHAR, etc. now work with character # 256 (the dummy char)**

1108 LOCAL FILE SYSTEM

- **Incompatible Change: Local file system format changed; MUST reformat 1108 disks**

The 1108 low-level disk format has been changed. To use the Harmony release, do the following: (1) copy any valuable local disk files to floppy or file server; (2) repartition the whole 1108 disk using the Harmony Installation Utility floppy (see the 1108 Users guide). Note that this erases ALL information on the disk; and (3) use DFSCREATEDIRECTORY to recreate any Lisp directories on local disk logical volumes.

Important Warning: Because of the change in disk format, you cannot run a Carol sysout on a Harmony-partitioned 1108, nor a Harmony sysout on a Carol-partitioned 1108. Attempting to do so may destroy information on the local disk.

- **Incompatible Change: Access logical volume FOO by {DSK}<FOO>, instead of {FOO}.**

There is now a single local hard disk file device, {DSK}. Each logical volume with a Lisp directory on it now counts as a separate directory of the device {DSK}. (In Carol, each logical volume with a Lisp directory counted as a separate device.)

- **Incompatible Change: Many local file system functions renamed**

The user functions for the local file system have been redesigned. A number of functions have been renamed, and others have been added or deleted. The following functions have been renamed:

MKDIR = = > DFSCREATEDIRECTORY

MAKEPILOT = = > DFSPURGEDIRECTORY

DFSVOLUMES = = > VOLUMES

For more information, see the 1108 Users Guide.

- **(DISKPARTITION) returns the name of virtual memory logical volume**

When Interlisp is running on an 1108, the function DISKPARTITION returns the name of the 1108 logical volume containing the currently-running Interlisp virtual memory. This is analogous to the behavior of this function on 1100 or 1132s. The function CURRENTVOLUME has been removed.

- **SCAVENGEVOLUME now preserves filenames**

Note: SCAVENGEVOLUME is no longer included in the standard Interlisp-D system. It is available by loading the library package DlionFSScavenge.DCOM.

- **Running local file system functions on non-1108s will fail gracefully**

Previously, calling local file system functions from Interlisp running on a non-1108 would cause strange low-level errors. Now, all of these functions check whether Interlisp is running on an 1108, and generate an appropriate error message if not. One exception: VOLUMEDISPLAY simply returns NIL if not on an 1108, so this function can be called from init files that are run on different machines.

- **Local file system does not allocate large files all at once**

Previously, COPYFILE of a large file from an NS file server to the local file system would fail, because the local file system would try allocating the entire file first, and the connection would time out.

- **Local disk renames files from core device {DSK} to {PSEUDO-DSK}**

If a core device {DSK} exists when the local file system wants to create a device DSK for the local file system, a new device {PSEUDO-DSK} is created, any files are copied over, and the core device {DSK} is deleted. A warning message is also printed. If there are no files on the core device DSK, it is simply deleted with no warning message.

- **1108 Local disk coerces HOST/DEVICE names to upper case**

When returning full file names, the 1108 local file system coerces the "host/device" name to upper case: {dsk} -> {DSK}.

- **The VOLUMEDISPLAY window can be reshaped**

- **Known Bug: Local file system does not preserve file type information.**

- **Known bug: Should format new floppies before doing SYSOUT[{FLOPPY}]**

There have been some cases where SYSOUT[{FLOPPY}] produced an incomplete sysout, when floppies that had never been formated before were used. Workaround: do (FLOPPY.FORMAT NIL NIL T) on new floppies first.

- **Floppy state flushed over LOGOUT**

Previously, if a floppy was left in a drive during LOGOUT and a new floppy was inserted before Interlisp was restarted, it was possible that Interlisp would use the old floppy directory information, which could destroy information on the new floppy. Now, floppy directory information is refetched the first time the floppy is used after Interlisp is restarted.

- **Trying to access FLOPPY on non-1108 no longer hangs**

Trying to access the floppy drive when running Interlisp on a machine other than the 1108 will now print "Floppy: No floppy drive on this machine" before generating the system error "FILE WON'T OPEN". Other operations, such as DIR {FLOPPY}* will not generate an error, but instead make {FLOPPY} look like it has an empty directory so that file searches work correctly when {FLOPPY} is on the search path.

- **COPYFILE of SYSOUT from non-Interlisp floppies works correctly**

Previously, COPYFILE would not copy a sysout file from floppies correctly, if the sysout was not originally put there by Interlisp.

- **(OUTFILEP '{FLOPPY}xxx) works, instead of returning NIL**

- **New messages when creating multi-floppy file**

When copying a sysout or other large file to floppies (with FLOPPY.MODE = SYSOUT or HUGEPILOT), a message is printed at the start saying how many floppies will be required. Between floppies, the message now says "Insert Floppy #n", rather than "Insert next floppy".

- **(INFILEP '{FLOPPY}xxx) returns NIL if no floppy in the drive**

Used to cause an error.

- **Floppy file versions on different floppy directories incremented correctly**

Previously, the floppy system ignored floppy file directories when computing the next version number for a new file. For example, it would create the file {FLOPPY}<BAR>NAME.;2 if there was a file {FLOPPY}<FOO>NAME.;1.

- **FLOPPY.MODE no longer changed to SYSOUT after a sysout**

(SYSOUT '{FLOPPY}) will automatically change the floppy mode to SYSOUT during the sysout. However, after the sysout is completed, it is changed back to what it was before the sysout.

- **FLOPPY now supports file types**

- **Floppy error msgs are printed in the typescript window, rather than the prompt window**

- **(COPYFILE xx '{FLOPPY}) in PILOT mode gives error message**

Previously, copying to {FLOPPY} giving a null file name could damage the information on the floppy. Now it just gives an error message.

- **Bad error msg "ARG NOT LP: NIL" changed**

This obscure error message occurred when trying to read a Pilot floppy file when FLOPPY.MODE was set to SYSOUT. The appropriate work around was to execute (FLOPPY.MODE 'PILOT), and trying again.

- **FLOPPY.COMPACT accepts "No" as answer to confirmation**

- **Floppy errors are now regular file system errors**

- **Sysout to write-protected floppy prints reasonable error message**

- **1108 optional RS232C port is supported; provides more reliable communication**

The Harmony release supports the optional RS232C port on the 1108. This port is buffered independently of Lisp operations, so there is little, if any, chance of dropping characters. Use of this port requires the E30 hardware option.

- **RS232 documentation totally revised; new functions**

The RS232 documentation has been totally revised so it doesn't focus on the implementation on the Xerox 1100. A few of the minor additions explained in the new documentation:

The function RS232SHUTDOWN is a "cleaner" way of doing (CLOSEF '{RS232})

The function RS232INPUTSTRING inserts characters into the input ring buffer. This permits a way to simulate the reception of characters through the actual UART

The function RS232FORCEOUTPUT has a new argument which specifies whether the function should return before all the characters are transmitted (the default).

RS232DEVICEERRORFN is a new global variable used when handling hardware errors

The RS232LOGIN facility is now documented. This provides a way for automating the login procedure when connecting to various hosts.

- **Incompatible Change: Global variable RS232XON\XOFF? replaced by function RS232XON\XOFF?**

The interface to the XON/XOFF protocol has been changed: rather than setting the global variable RS232XON\XOFF?, the new function (RS232XON\XOFF? ON?) should be used to set and unset this state. In future release of the I/O processor code, enabling/disabling XON/XOFF processing by the RS232C port will require a functional interface to what is now merely a global variable.

- RS232 no longer breaks over LOGOUT/restart

- RS232CHAT command ~LocalEcho works as specified

[This page intentionally left blank]

NS FILE SERVERS

- **Many NS filing reliability problems fixed**

The NS filing system has been reworked, and made much more robust. In particular, a number of problems associated with open files timing out have been solved.

- **NS filing directory operations automatic**

NS file servers support a true hierarchical file system, where subdirectories are just another kind of file. In previous releases of Interlisp-D, users had to explicitly create subdirectories using the function NSCREATEDIRECTORY. In Harmony, subdirectories are created automatically as needed: A call to OPENFILE to create a file in a non-existent subdirectory automatically creates the subdirectory; CONN to a non-existent subdirectory asks the user whether to create the directory. The function NSCREATEDIRECTORY has thus been removed. Note: Requires Services Release 8.0.

- **DIR fully enumerates NS files in subdirectories**

In previous releases, DIR enumerated a directory only to the first level; it did not recursively enumerate the contents of subdirectories. In Harmony, DIR can enumerate a directory to arbitrary depth; the exact depth is controlled by the variable FILING.ENUMERATION.DEPTH, which is a small positive integer or T. The default value is T, meaning infinite depth: the entire directory is enumerated, and subdirectory "files" do not appear at all. Also, the special function NSDIRECTORY is no longer needed, and has been removed: DIRECTORY works with NS file servers exactly as with other devices. Note: Requires Services Release 8.0. Earlier versions of Services will continue to behave as if FILING.ENUMERATION.DEPTH = 1.

- **NS file operations prompt for password**

When the user attempts an NS file server operation, Interlisp passes the current username and password (as given to the function LOGIN) to the NS file server. If these are not accepted, Interlisp prompts the user to enter the correct name and password. If the current username and password are correct, the user is not prompted at all.

Note: The user can abort an NS password prompt by typing control-E. The result of the file operation will be as if the NS file server did not exist.

- **SETFILEINFO, GETFILEINFO can access the TYPE attribute of NS files**

GETFILEINFO and SETFILEINFO now accept the TYPE and FILE.TYPE attributes for NS files. TYPE is the standard Lisp file type, with values TEXT and BINARY. FILE.TYPE is the (server-dependent) numeric value of the file's FILE.TYPE property, which a 16-bit number for NS file servers. Using the FILE.TYPE attribute, you can change the file type to other non-lisp file types.

- **GETFILEPTR works with NS files**

SETFILEPTR of an NS file causes an error, since NS file servers do not currently support random access. However, GETFILEPTR now returns the correct character position for open files on NS file servers.

Note: SETFILEPTR works in the special case where the file is open for input, and the file pointer is being set forward. In this case, the intervening characters are automatically read.

- **(FULLNAME <file> 'NEW) and OUTFILEP now work for NS file servers.**

Used to return NIL.

- **Can generate hardcopy of full screen on Xerox 8044 printer**

Use the HARDCOPYW function or the HARDCOPY command in the Background menu.

- **Multiple transmissions to NS printers no longer cause break "not an open NS socket"**

Multiple concurrent NSPRINTs or HARDCOPYWs no longer confuse each other. No more breaks with "not an open NS socket".

- **ROTATION argument to HARDCOPYW works to 8044 printers**

Previously, the rotation argument was not supported when sending bitmaps to the 8044 printer. Now, this is supported for ROTATION = a multiple of 90 degrees.

- **Can print to NS printers with A4 paper: variable NSPRINT.DEFAULT.MEDIUM**

The variable NSPRINT.DEFAULT.MEDIUM can be used to set the default NS printer medium. NIL (the default) means to use the printer's default; T means to use the first medium reported available by the printer; any other value must be a Courier value of type MEDIUM. The format of this type is a list (PAPER (KNOWN.SIZE <TYPE>)) or (PAPER (OTHER.SIZE (<WIDTH> <LENGTH>))). The paper type <TYPE> is one of the atoms US.LETTER, US.LEGAL, A0 through A10, ISO.B0 through ISO.B10, and JIS.B0 through JIS.B10.

For European users who use A4 paper exclusively, it should be sufficient to set NSPRINT.DEFAULT.MEDIUM to (PAPER (KNOWN.SIZE "A4")).

Note: When using different paper sizes, it may be necessary to reset the variable DEFAULTPAGEREGION, the region on the page used for printing (measured in micas from the lower-left corner).

(This page intentionally left blank)

- **SPP, Courier, Clearinghouse reimplemented: low-level incompatibilities**

As part of the improvements to NS Filing and Printing, the underlying implementations of SPP, Courier and Clearinghouse have been substantially rewritten, in several places incompatibly. Users who program applications that use SPP, Courier or Clearinghouse directly should read Appendix D (NS Protocol Support).

- **"SPP Retransmit Queue out of order" errors fixed**

The SPP retransmit strategy has been completely revised, so this intermittent problem should disappear.

- **SETTIME now broadcasts for both PUP and NS time servers.**

SETTIME used to just try for a PUP time server.

- **Superfluous "not responding" messages after NS operations removed**

Also, the function CLOSE.NSFILING.CONNECTIONS has been removed.

- **GETPUPSTRING applied to a blank pup now returns the null string instead of erroring**

- **1108s can "hear" their own Ethernet transmissions**

The 1108 hardware is not capable of receiving the Ethernet packets it transmits. In previous releases, this meant that if an 1108 sent a packet addressed to itself, it would never receive it. In Harmony, the 1108 low-level Ethernet software takes care of this by faking receipt of such a packet. The implication of this is that programmers writing Lisp-based Ethernet servers can now test them out by running user and server code on the same machine.

- **Interlisp no longer hangs on an 1100 running 3MHz ethernet microcode without a 3MHz ethernet card**

(This page intentionally left blank)

- **The ATTACHEDWINDOW package has been added to the standard Interlisp loadup.**

The ATTACHEDWINDOW library package has been added to the standard Interlisp-D window system. Many system tools (inspector, break package, etc.) now use attached windows for managing sets of windows. A number of changes have been made to the attached window facility:

New "attached prompt windows" provide a uniform way to access a small prompt window attached to another window.

Attached windows can be closed without closing the main window

BURY now buries all attached windows correctly

New function (DETACHALLWINDOWS MAINWINDOW) detaches and closes all windows attached to MAINWINDOW.

ATTACHMENU opens menu window immediately; new arg NOOPENFLG

New function MAINWINDOW for getting the mainwindow from an attached window

For complete documentation of the attached window facility, see Appendix B (Attached Windows).

- **Can move icons with LEFT button, expand with MIDDLE button**

Buttoning the LEFT button on an icon allows you to move it. Pressing the MIDDLE button expands it.

- **Changes to WFROMDS reduce empty tty windows**

WFROMDS has a new arg, DONTCREATE. If DONTCREATE is non-NIL, WFROMDS will never create a window, and return NIL if DISPLAYSTREAM does not have an associated window.

TTYDISPLAYSTREAM calls WFROMDS with DONTCREATE = T, so it will not create a window unnecessarily. Also, if WFROMDS does create a window, it calls CREATEW with NOOPENFLG = T. These changes fix many of the empty tty windows that used to appear.

- **Many changes to caret behavior**

There is now one caret per process. This fixes problems with carets being left on the screen and with windows being created just to take the caret down. The caret in the current process is always visible; if it is hidden by another window, its window is brought to the top. The function CARET has been changed, and the function CARETRATE, which changes the caret rate of the current process has been added:

(CARET NEWCARET)
Sets the shape that blinks at the location of the next output to the current process. NEWCARET is either (1) NIL - no changes, returns a CURSOR representing the current caret, (2) OFF - turns the caret off, (3) a CURSOR which gives the new caret shape or (4) T - resets the caret to the default which is the value of the variable DEFAULTCARET. DEFAULTCARET can be set to change the initial caret for new processes. The hotspot of NEWCARET indicates which point in the new caret bitmap should be located at the current output position. The previous caret is returned. Note: it is now permissible for the caret bitmap to be larger than cursor bitmap dimensions (16x16).

(CARETRATE ONRATE OFFRATE)
Sets the rate at which the caret for the current process will flash. The caret will be visible for ONRATE milliseconds, then not visible for OFFRATE milliseconds. If OFFRATE = NIL, the value of ONRATE is used. If ONRATE is T, both the "on" and "off" times are set to the value of the variable DEFAULTCARETRATE (initially 333). The previous value of CARETRATE is returned. If the caret is off, CARETRATE return NIL.

- **Caret flashing doesn't bring window to top during buttoning or copy-selecting**

The caret code has been changed so that it doesn't bring the flashing caret's window to the top if the user is buttoning or has a copy key down. This prevents the destination window (which has the tty and caret flashing) from interfering with the window one is trying to select text to copy from.

- **Cursor reset correctly after going through scroll bar**

Previously, slowly dragging the mouse out of the left of a Tedit window would change the cursor to a right-facing arrow (in the left margin), change it to the scrolling cursor (in the scroll bar), and "restore" it to the right-facing arrow upon leaving the scroll bar. The window system now restores the cursor correctly to the value of DEFAULTCURSOR upon leaving a window.

- **New Background menu when Copy-key pressed; allows copy-inserting a SNAP**

Various system utilities (TEdit, DEdit, TTYIN) allow information to be "copy-inserted" at the current cursor position by selecting it with the "copy" key held down. (Normally the shift keys are the "copy" key, this action can be changed in the key action table.) It is now possible to "copy-insert" the bitmap of a snap into a Tedit document. If the right mouse button is pressed in the background with the copy key held down, a menu with the single item "SNAP" appears. If this item is selected, the user is prompted to select a region, and a bitmap containing the bits in that region of the screen is inserted into the current tty process, if that process is able to accept image objects (like Tedit).

This is implemented by the new variables BackgroundCopyMenu and BackgroundCopyMenuCommands, which are interpreted similar to BackgroundMenu and BackgroundMenuCommands. If the right mouse button is pressed in the background when the copy key is down, the menu stored in the variable BackgroundCopyMenu is envoked. If this is NIL, a new menu is created from the menu commands in BackgroundCopyMenuCommands.

- **RESHAPEBYREPAINTFN uses new strategy to determine window contents after reshape**

Previously, RESHAPEBYREPAINTFN (the default reshaping function) always copied the old image to the lower-left corner of the new window, adding any new image to the top and left. This produced unintuitive results in the case where the lower left corner was grabbed and moved out. The new behavior will display the part of the object in the direction of the expansion (if the opposite side is not moved) and only display white space beyond the extent if the extent is fully visible.

This change required that a fourth argument be passed to the RESHAPEFN of a window: OLDSCREENREGION, the region that the window occupied before being reshaped. This allows RESHAPEBYREPAINTFN to determine which edges of the window have been moved. Note: in some situations, RESHAPEBYREPAINTFN may call a window's REPAINTFN as many as four times on different window regions.

- **New Background Button Event Functions**

The variables
  BACKGROUNDBUTTONEVENTFN,
  BACKGROUNDCURSORINFN, BACKGROUNDCURSOROUTFN
  and BACKGROUNDCURSORMOVEDFN
provide a way of taking action when there is cursor action when the cursor is in the background. If set to the name of a function, that function will be called, respectively, whenever the cursor is in the background and a button changes, when the cursor moves into the background from a window, when the cursor moved out of the background into a window, and when the cursor moves from one place in the background to another. These are analogous to the window properties BUTTONEVENTFN, CURSORINFN, CURSOROUTFN, and CURSORMOVEDFN.

- **New BURYW behavior -- faster algorithm**

BURYW has been changed to take down the windows overlapping the window to be buried, then reopening them in the right order.

- **1108 background border preserved over LOGOUT/restart**

The backgound border (around the screen) on the 1108 can be changed with the function CHANGEBACKGROUNDBORDER. During LOGOUT, the border is changed back to the default shade. Now, the border is restored to its new pattern after LOGOUT/restart on an 1108.

- **Fixed: Caret didn't flash on 1108 after LOGOUT/restart**

Sometimes, the caret would not flash after doing LOGOUT and restarting Interlisp on an 1108. This could be fixed by typing (CARET), so it was not a major problem, but it was annoying.

- **If MENU is called with RELEASECONTROLFLG = T, the menu window is brought to the top.**

Previously, a "released" menu could be hidden by other windows. Now, the released menu will stay visible until it is closed or an item is selected.

- **Moving an off-screen window onto the screen redisplays its contents**

- **DRAWCURVE to the display works correctly in INVERT mode if BRUSH = 1**

DRAWCURVE, DRAWCIRCLE, and DRAWELLIPSE to the display will work if the brush argument is 1, and the "operation" of the displaystream is INVERT. For brushes larger than 1, the INVERT operation is the same as the ERASE operation.

DRAWCURVE to other image streams generally only supports the PAINT operation.

- **New window property: NOSCROLLBARS**

If a window's NOSCROLLBARS property is non-NIL, scroll bars will not be brought up for the window, even if it has both EXTENT and SCROLLFN properties. This allows the creation of windows that can scroll ONLY under program control.

- **New window property: WINDOWTITLESHADE sets shade used in window title bar.**

If a window's WINDOWTITLESHADE property is non-NIL, it should be a texture which is used as the "backgound texture" for the title bar on the top of the window. If this property is NIL, then the value of the variable WINDOWTITLESHADE is used, initially black. Note that black is always used as the background of the title printed in the title bar, so that the letters can be read -- the remaining space is painted with the "title shade".

- **Textures can be BITMAPs up to 16 by 16 bits**

TEXTUREP, BITBLT, DSPTEXTURE, DSPFILL, etc. accept bitmaps up to 16 bits wide by 16 bits high as textures. When a region is being filled with a bitmap texture, the texture is treated as if it were 16 bits wide (if less, the rest is filled with white space).

- **New functions INVERTW, FLASHWINDOW**

(INVERTW WIN SHADE)
Inverts the window WIN, by XOR-ing it with the shade SHADE. If SHADE = NIL, the default is to XOR with the shade BLACK, which simply inverts the bits.

(FLASHWINDOW WIN? N FLASHINTERVAL SHADE)
Flashes the window WIN?, by inverting it twice. N is the number of times to flash the window (default is once). FLASHINTERVAL is the number of milliseconds to wait with the window inverted (default is 200). SHADE is interpreted as in INVERTW.

If WIN? is NIL, the whole screen is flashed. In this case, the SHADE argument is ignored (can only invert the screen).

- **New function DECODE.WINDOW.ARG: coerces window specs to window**

(DECODE.WINDOW.ARG WHERESPEC WIDTH HEIGHT TITLE BORDER NOOPENFLG)
This is a useful function for creating windows. WHERESPEC can be a WINDOW, a REGION, a POSITION, or NIL. If WHERESPEC is a WINDOW, it is returned. In all other cases, CREATEW is called with the arguments TITLE, BORDER, and NOOPENFLG. The REGION argument to CREATEW is determined from WHERESPEC as follows:

If WHERESPEC is a REGION, it is adjusted to be on the screen, then passed to CREATEW. If WIDTH and HEIGHT are not numbers, CREATEW is given NIL as a REGION argument.

If WIDTH and HEIGHT are numbers and WHERESPEC is a POSITION, the region whose lower left corner is WHERESPEC, whose width is WIDTH and whose height is HEIGHT is adjusted to be on the screen, then passed to CREATEW.

If WIDTH and HEIGHT are numbers and WHERESPEC is not a POSITION, then GETBOXREGION is called to prompt the user for the position of a region that is WIDTH by HEIGHT.

If WIDTH and HEIGHT are used, they are used as interior dimensions for the window.

- **New function MAKEWITHINREGION: moves region within another region**

(MAKEWITHINREGION REGION LIMITREGION)
Changes (destructively modifies) the left and bottom of the region REGION so that it is within the region LIMITREGION, if possible. If the dimensions of REGION are larger than LIMITREGION, REGION is moved to the lower left of LIMITREGION. If LIMITREGION is NIL, the value of the variable WHOLEDISPLAY (the screen region) is used. MAKEWITHINREGION returns REGION.

- **INSIDEP now accepts a window as its REGION arg**

If the REGION arg to INSIDEP is a window, the window's interior (its clipping region) is used.

- **REGIONP now true for regions whose components are floating point numbers.**

Previously, only integers were allowed as components of a region.

- **EXPANDBITMAP works without the color package loaded**

EXPANDBITMAP uses the function \FAST4BIT, which was previously only defined in the color library package. \FAST4BIT has been added to the standard Interlisp loadup.

- **READBITMAP, PRINTBITMAP have new argument: FILE**

(READBITMAP FILE)
(PRINTBITMAP BITMAP FILE)
These functions can now be used to read and print bitmaps to arbitrary files, without changing the primary input/output stream.

- **CURSORINFN and CURSOROUTFN window properties extended**

The CURSORINFN and CURSOROUTFN window properties can now be lists of functions as well as single functions. All functions on the list are called.

- **Scrollbar provides better indication when contents are above the window**

Previously, there were some cases where the scrollbar would not hit the bottom unless the bottom of the extent was a small distance above the top of the window.

- **EDITBM does not reposition the cursor to the center of the screen**

- **Control-D during EDITSHADE now closes the window**

- **(BITMAPHEIGHT <texture>) now gives an error for non-bitmap textures**

- **If CHANGEOFFSETFLG menu property is non-NIL, popup menus come up in correct position**

Previously, they would come up one pixel above and to the right of where they were last time (relative to the cursor).

- **Scrolling works correctly after changing window border size**

- **If LEFT button down, GETREGION calls NEWREGIONFN with MOVINGPOINT = NIL**

Previously, if GETREGION was called when one of the mouse buttons is already down (LEFT), the first call to NEWREGIONFN did not have MOVINGPOINT = NIL

- **EXPANDW no longer fails if called on expanded window**

- **(DSPCREATE <bad-arg>) signals "ILLEGAL ARG" error, instead of going into RAID**

- **DRAWCURVE no longer generates an error if dashing is non-NIL**

- **ADDMENU/DELETEMENU do not modify the menu for subsequent use**

(This page intentionally left blank)

- **New Tedit page formatting facilities**

Tedit now includes facilities for specifying the page layout to be used when a document is formatted and printed. The user can now control page formatting such as page numbers, headings, multiple columns, etc.

- **Tedit has separate menus for Para looks, Char looks, and Page Looks**

This solves a number of problems. In particular, it is no longer necessary to scroll a single long menu up and down to set and apply character and paragraph looks.

- **Control-E can be used to abort Get, Put, etc. commands**

After selecting the Tedit commands Get, Put, Include, etc. from the title menu, the user is asked to type in a file name. The operation can be aborted at this time simply by typing control-E.

- **Can shrink an unsaved Tedit document**

Previously, Tedit caused an error, when it tried to print the file name of the document in the icon. Now, it detects this situation and creates an empty icon.

- **Tedit more careful about erasing caret images on the screen**

- **EOFP works correctly for text streams**

- **Tedit hardcopy uses {DSK} to store large files, so larger files can be printed**

- **Tedit uses more compact representation for bitmaps**

The format of bitmaps in Tedit files has been changed. This new format should take up about half the space, and it can be read/written many times as fast of the old format. It does not do any compression. The old bitmap-reading functions have not been removed, so old bitmaps will be converted as they are encountered.

(This page intentionally left blank)

- **"BREAK" or "()" of top-level expression no longer causes stack overflow**

- **Process switch from DEDIT to TEDIT won't cause it to ignore tabs**

- **DEdit's internal data structures revised to take 1/3 less space**

This should improve swapping performance over extended programming sessions.

- **(DF <undefined function>) creates blank function template.**

If DF is called on a name with no function definition, the user is prompted with "No FNS defn for <function name>. Do you wish to edit a dummy defn?". If the user confirms (by clicking left-button), a "blank" definition is displayed in the Dedit window. If any changes are made, on exit from the editor, the definition will be installed as the name's function definition. Exiting the editor with the STOP command will prevent any changes to the function definition.

If DF is called with a second arg of NEW, as in (DF <function name> NEW), a blank definition will be edited whether the function already has a definition or not.

- **Inserting huge piece of code no longer causes bad screen extent.**

Previously, after inserting a huge piece of code into a function, Dedit could lose track of the size of the function, so the user could not scroll up enough to see the last part of the inserted code.

- **DEDIT REPAINTFN** redisplays selection highlighting

- Comments print correctly when inserted or SWITCHed

- Deleting first dotted-pair from list of pairs reprints correctly

- **Buttoning in the Dedit Edit Buffer switches the current process.**

Previously, you had to button in the main Dedit window.

- **Dedit menu only comes to the top when Dedit is the TTY process.**

- **"Shouldn't Happen! DEDITDSPS tangled" errors reduced (**

Previously, this could happen if you called the inspector from Dedit (by EVAL-ing (INSPECT ...)), and called Dedit from the inspector window. This particular symptom has been cured in the current Dedit. However, exiting DEdit processes out of order can still cause this error.

- **Dedit supports the COPY key on the 1108 keyboard**

- Shift select supports both the COPY and the right shift keys.

- **EditOps menu follows when the main Dedit window is moved**

- **Edit buffer doesn't attach to incorrect window**

Previously, after "TTYIn Form" of atom, the Dedit Typein window for DEDIT sometimes would attach itself to the bottom of random windows on the screen.

- **Dedit doesn't reprint function on exit.**

Previously, in some situations Dedit would reprint the entire function after exit, as a side effect of changing the edit date comment.

- **!UNDO command is now undoable**

- **Double deletes give better error message**

Previously, if one deleted a deleted selection (in a serious of commands with the control key down), Dedit would break with the error "Shouldnt: No MapEntry". Now, Dedit detects this situation, prints out the error message "Cant: Already deleted!", and doesn't cause a break.

- **CAP command capitalizes first letter of atom**

It used to do the same as RAISE, capitalizing all the letters of the item selected.

- **"? = " command in Dedit works for fns of no arguments**

Used to give "xxx not a function" error message.

- **Editor called from display break package in broken process**

Inspecting a function in the display frame window now calls the editor in the broken process. Thus variables evaluated in the editor will be in the broken process.

- **Can now REVERT to any frame on the stack.**

Previously, there were restrictions on reverting to internal "DUMMY" frames, because it could cause the system to crash or freeze. Now, REVERT has been fixed so that it is safe to revert to any frame on the stack.

- **Break windows are not opened on "STORAGE FULL" errors.**

This is similar to the treatment of "ARRAYS FULL" errors. In either case, allocating storage for a break window would cause the error to occur repeatedly.

- **Typing control-B in a break window no longer gives "Break within Break" error**

- **AUTOBACKTRACEFLG extended: can cause BT for NON-error breaks**

Previously, if AUTOBACKTRACEFLG was non-NIL, then the command BT would be executed automatically on error breaks, but not on user breaks (calls to functions broken by BREAK). It has been extended as follows: If AUTOBACKTRACEFLG is NIL (the default), no backtrace is brought up. If its value is T, then on error breaks the BT menu is brought up. If its value is BT!, then on error breaks the BT! menu is brought up. If its value is ALWAYS, then on any break the BT menu is brought up. If its value is ALWAYS!, then on any break the BT! menu is brought up.

- **ERRORTYPELST is now a SPECVAR**

It makes sense for users to change the global value of ERRORTYPELST, but programs that rebind it clearly want changed behavior only in their own stack context. It is only looked up under error conditions, so it shouldn't cause a performance problem.

- **Break package more careful about aborting process on closing** window

Closing a break window now only aborts the associated process if it was in tty wait and the closed window was the tty window. This should stop some inadvertant aborts.

Warning: Typed-in BT, BTV commands don't start at top of stack

When a stack frame name is selected in the backtrace menu, the variable LASTPOS is set to the selected stack frame. This allows breaks commands such as REVERT, ?=, etc. to use the selected frame. However, the value of LASTPOS also indicates to the break commands BT, BTV, BTV!, etc. where to start listing the stack.

- **Using SET to set inspector values no longer creates many TTY windows**

Previously, the inspector SET command would create a new window for the user to type a value every time it was used. Now, the default SET routine uses an attached prompt window on top of the inspect window to receive the new value.

- **New inspect window commands: "IT←datum", "IT←selection"**

The values displayed in an inspect window can be accessed by commands on the menu brought up by pressing the MIDDLE button in the title of the window. The command "IT←datum" sets the variable IT to the object being inspected in this window. The command "IT←selection" sets the variable IT to the current property name or value selected in the inspect window.

- **Variable INSPECTPRINTLEVEL used for printing inspector values**

When the inspector prints field values, PRINTLEVEL is reset to the value of INSPECTPRINTLEVEL, initially (2 . 5).

- **Inspector calls INSPECTCODE to inspect compiled code objects**

(This page intentionally left blank)

CHAT

- **Chat does not turn off interrupt characters until AFTER creating the Chat window**

This allows the user to abort the call to Chat by typing control-E while specifying the Chat window region.

- **Reshaping Chat window does not change terminal type**

Previously, reshaping a Chat window caused Chat to reassert the terminal type specified when the connection was first opened. If the user in the meantime had told the remote host that the terminal type was different, then this would set it back.

- **Chat grabs TTY as soon as it starts to reconnect**

Previously, the "reconnect" menu button didn't switch the tty to the chat process until the connection was reestablished.

- **Chat ignores the padding character DEL**

- **Chat display no longer off by 1 character after EMACS insert operation**

Newer versions of EMACS perform character insertion by an unusual sequence that Chat was not emulating correctly.

- **Chat in EMACS mode updates cursor position promptly**

Previously there was a bug that deferred the cursor update following a positioning command with the mouse until the next type-in occurred.

- **Chat displays the EMACS mode state in the window title**

When Chat EMACS-mode is on, "EMACS ON" is printed in the Chat window title.

(This page intentionally left blank)

- **Incompatible change: EDITPREFIXCHAR default is NIL**

The variable EDITPREFIXCHAR is now by default NIL, meaning there is initially no TTYIN prefix meta-character defined. This change was made to avoid confusing users who don't use TTYIN editing commands. If you want to be able to issue editing commands to TTYIN, you should either call (TTYINMETA T) to enable bottom-blank (STOP on 1108's) as a true meta key, or set EDITPREFIXCHAR to the character code of your preferred meta prefix (it used to be 193, for top-blank).

- **TTYIN is enabled in break windows created by control-B during type-in**

- **FIX command with TTYIN prettyprints history events**

The programmer's assistant command FIX calls TTYIN to edit the text of the history event. TTYIN now prettyprints the event for ease of editing.

- **Typing control-E under TTYIN won't cause "NON-NUMERIC ARG" error**

- **Typein lines starting with ";" no longer erased**

Previously, TTYIN interpreted a line starting with the character ";" as a comment, and would ignore it, erasing the line from the screen. Although ";" is defined on LISPXHISTORYMACROS as a no-op anyway, TTYIN's behavior was inappropriate in cases where one was not typing to the Lisp exec.

The old behavior is still available for those desiring it: if the first character on a line of typein is equal to the variable TTYINCOMMENTCHAR (a character code or NIL), then the line is erased, and no input function will see it. TTYINCOMMENTCHAR is initially NIL.

(This page intentionally left blank)

- **Known Bug: Must do (HARDRESET) after stack overflow, or else second stack overflow gives fatal error**

If a stack overflow occurs, rather than type " ↑ " to escape from the break, do a hardreset. Otherwise, the NEXT stack overflow may cause an unrecoverable error. Either evaluate (HARDRESET) from the break window, or type control-D from Teleraid.

- **New function: EVALHOOK**

(EVALHOOK FORM EVALHOOKFN)
EVALHOOK evaluates the expression FORM, and returns its value. While evaluating FORM, the function EVAL behaves in a special way. Whenever a list other than FORM itself is to be evaluated, whether implicitly or via an explicit call to EVAL, EVALHOOKFN is invoked (it should be a function), with the form to be evaluated as its argument. EVALHOOKFN is then responsible for evaluating the form; whatever is returned is assumed to be the result of evaluating the form. During the execution of EVALHOOKFN, this special evaluation is turned off. (Note that EVALHOOK does not effect the evaluations of variables, only of lists).

Here is an example of a simple tracing routine that uses the EVALHOOK feature:

```
←(DEFINEQ (PRINTHOOK (FORM)
        (printout T "eval: " FORM T)
        (EVALHOOK FORM (FUNCTION PRINTHOOK]
(PRINTHOOK)
←(EVALHOOK '(LIST (CONS 1 2) (CONS 3 4)) 'PRINTHOOK)
eval: (CONS 1 2)
eval: (CONS 3 4)
((1 . 2) (3 . 4))
```

- **Internal arithmetic functions changed to have the "right" frame name**

In compiled code, a call to a primitive arithmetic function, such as PLUS, turns into a Lisp opcode, which normally executes entirely in microcode. In exceptional cases, however, the microcode executes a call on an internal arithmetic function, such as \SLOWPLUS2. Previously, if an error occurred in such a function, the backtrace contained the internal function name, rather than the name you would expect from looking at the source code. This has been changed so that the frame names of internal arithmetic functions are the appropriate user-level functions.

- **EVALV has new argument RELFLG: release-stack-ptr flag**

Most of the stack evaluation functions (ENVEVAL, etc.) have a flag argument which determines whether the stack pointer will be automatically released. To be consistent, EVALV now has a RELFLG argument, even though it doesn't strictly need it (EVALV is guaranteed to return, unlike the other functions).

- **(APPLY*) now gives "UNDEFINED FUNCTION: NIL" error**

- **BREAK, TRACE, SEE, etc. recognize quoted arguments: new function NLAMBDA.ARGS**

A number of NLAMBDA functions now recognize if their argument is quoted. For example, (BREAK 'FOO) will now break the function FOO, rather than the function QUOTE. LISPX macros and commands which normally take their args unquoted (DIR, CONN, etc.) also work with quoted arguments. For example, typing DIR 'FOO* is now the same as DIR FOO*.

This change was accomplished by defining a new function (NLAMBDA.ARGS X). This interprets its argument as a list of unevaluated nlambda arguments. If any of the elements in this list are of the form (QUOTE ...), the enclosing QUOTE is stripped off. Actually, NLAMBDA.ARGS stops processing the list after the first non-quoted argument. Therefore, whereas (NLAMBDA.ARGS '((QUOTE FOO) BAR)) -> (FOO BAR), (NLAMBDA.ARGS '(FOO (QUOTE BAR))) -> (FOO (QUOTE BAR)).

- **Error correction of function name doesn't lose args**

Previously, if one had an NLAMBDA nospread function FOO, one could type "FOO ALPHA" to the exec and FOO would be run, with ALPHA as its argument. If however, one mistyped FOO (as foo, FOOX,etc.) and the spelling corrector sucessfully corrected it to FOO, the exec did not pass the arguments along. This has been fixed.

- **PRINTLEVEL UNDO-able from top level exec**

Typing PRINTLEVEL to the top-level exec will substitute a call to the undoable function /PRINTLEVEL.

(This page intentionally left blank)

- **Incompatible Change: Source/DCOM file location algorithm changed**

Each Interlisp source and compiled code file contains the full filename of the file, including the host and directory names, in a FILECREATED expression. The compiled code file also contains the full file name of the source file it was created from. Previously, the file package used this information to locate the appropriate source file when "remaking" or recompiling a file.

This has turned out to be a bad feature in distributed environments, where users frequently move files from one place to another, or where files are stored on removable media. For example, suppose you MAKEFILE to a floppy, and then copy the file to a file server. If you load and edit the file from a file server, and try to do MAKEFILE, it will break, trying to locate the source file on a floppy, which is probably no longer loaded.

In the Harmony release, the file package searches for the source file on the connected directory, and on the directory search path (on the variable DIRECTORIES). If it is not found, the host/directory information from the FILECREATED expression be used.

Warning: One situation where the new algorithm does the wrong thing is if you explicitly LOADFROM a file that is not on your directory search path. Future MAKEFILEs and CLEANUPs will search the connected directory and DIRECTORIES to find the source file, rather than using the file that the LOADFROM was done from. Even if the correct file is on the directory search path, you could still create a bad file if there is another version of the file in an earlier directory on the search path. In general, you should either explicitly specify the SOURCEFILE argument to MAKEFILE to tell it where to get the old source, or connect to the directory where the correct source file is.

- **HPRINT, UGLYVARS, HORRIBLEVARS don't redeclare datatypes**

The file package commands UGLYVARS and HORRIBLEVARS call the function HPRINT to print out loadable representations of arbitrary data structures. If a data structure contains an instance of an Interlisp datatype, the datatype declaration is also printed onto the file.

This has caused problems when a system datatype declaration dumped into a file doesn't match the current declaration. Redefining a system datatype will almost definitely crash Interlisp. The Interlisp system datatypes do not change very often, but there is always a possibility when loading in old files created under an old Interlisp release.

To prevent accidental system crashes, HREAD has been changed so that loading an HPRINTed structure will NOT redefine datatypes. Instead, it will cause an error "attempt to read DATATYPE with different field specification than currently defined". Continuing from this error will redefine the datatype.

- **Incompatible change: User INIT files are now loaded normally, and appear on FILELST**

Previously, the user init files were SYSLOAD-ed, and their filecoms were not saved. This was inconvenient when people wanted to modify their init files. Now, they are loaded with LDFLG = NIL, so their filecoms are saved, and they appear on FILELST. Note that the system "site" init file is still loaded with SYSLOAD.

The function GREET has been changed as follows:

The system greet file (GREETFILENAME T) is loaded with the SYSLOAD parameter. The user greet file (GREETFILENAME <username>) is loaded with normal file package settings, but also under errorset protection and with PRETTYHEADER set to NIL to suppress the "FILE CREATED" message.

Note: Users should try to make sure that their init file is "undoable". If they use the file package command "P" to put expressions on the file to be evaluated, they should use the "undoable" version, e.g. /SETSYNTAX rather than SETSYNTAX, etc. This is so another user can come up, do a (GREET) and have the first user's initialization undone.

- **MAKEFILE "remake" option asks whether to load DONTCOPY expressions**

When a MAKEFILE is performed with the "remake" option to copy definitions from an old file, MAKEFILE checks to see if all of the necessary definitions had been loaded from the old file. In the past, if you had only loaded the compiled version of a file with (DECLARE: .. DONTCOPY ..) expressions, MAKEFILE would automatically and quietly load the definitions from the old file. In some circumstances this could be disastrous -- if the user had circumvented the file package in some way, and loading the old definitions overwrote new ones.

MAKEFILE now asks before performing these operations, e.g.

"Only the compiled version of FOO was loaded, do you want to LOADVARS the (DECLARE: .. DONTCOPY ..) expressions from {DSK}<MYDIR>FOO.;3?"

- **RESOURCES facility provides tools to explicitly manage the allocation and reuse of complex data objects**

Interlisp is based on the use of a storage-management system which allocates memory space for new data objects, and automatically reclaims the space when no longer in use. More generally, Interlisp manages shared "resources", such as files, semaphors for processes, etc. Sometimes users need to explicitly manage the allocation of resources. The filepkg type RESOURCES is available to help with the definition and usage of such classes of data; the definition of a RESOURCE specifies prototype code to do the basic management operations. For more information, see Appendix E (RESOURCES).

- **HASDEF with SOURCE = ? calls WHEREIS database package if loaded**

According to the documentation, passing SOURCE = ? to the file package type functions should try (among other options) calling the function WHEREIS with FILES = T, which will search the WHEREIS hashfile database if the WHEREIS package is loaded. In the case of HASDEF called with SOURCE = ?, WHEREIS was being called with FILES = NIL, so the WHEREIS package was not being used. This produced strange behavior in Dedit, such that evaluating (DF <system-function>) would load and edit the function, but selecting the function in a Dedit window and buttoning "Edit" would not.

- **I.S.OPRS now works as a file package "type" for COPYDEF and UNSAVEDEF**

- **(* * X ...) no longer signifies that X is a filevar**

When a form such as (FNS * FOOFNS) appears in the filecoms of a file, this means that the list of functions should be taken from the variable FOOFNS. In this case, FOOFNS is known as a filevar.

Previously, there was a bug with comments of the form (* * this is a comment), where the first word of the comment ("this") was interpreted as a filevar. This had some strange consequences, such as the first words of such comments appeared in (FILECOMSLST xxx 'VARS), and these atoms were set to NOBIND if the file was loaded with LDFLG = SYSLOAD.

- **Comments allowed in file package commands**

The file package now allows comments to appear in most places in the filecoms. For example:

    (INITVARS (* this is a comment) (FOO 5)).

- **Default setting of CLEANUPOPTIONS changed to (RC)**

Previously, the default value of CLEANUPOPTIONS was (LIST RC), so CLEANUP would list and recompile all files. If you wish to retain that behavior, simply reset CLEANUPOPTIONS.

- **(PF <function> <file>) prints message if file not found, or function not found on file**

Previously, PF just returned NIL if either the function was not found or the file was not found.

DC FOO can find file FOO.LSP

Previously, the user had to type DC FOO.LSP to edit the coms of a file with a non-NIL extension.

- **ADDTOFILE prompt changed from "new file?" to "create new file XXX?"**

- **Incompatible change: Default RECOMPILEDEFAULT changed from EXPRS to CHANGES**

Previously, the default value of RECOMPILEDEFAULT was EXPRS. This meant that when recompiling a file, those functions currently defined by EXPRs would be recompiled. Generally, this is a good indication of which functions had been edited. However, a problem occurs if the user explicitly calls COMPILE to compile a particular function. A later RECOMPILE or CLEANUP would not recompile that function. By setting the default RECOMPILEDEFAULT to CHANGES, RECOMPILE or CLEANUP will recompile those functions which have been changed according to the FILECREATED expression in the source file. Under some circumstances, this may cause functions to be recompiled unnecessarily, but it is safer.

Benefits of RECOMPILEDEFAULT = CHANGES:

If you normally load a source file, edit a few functions, then MAKEFILE and RECOMPILE, the effect of the change to RECOMPILEDEFAULT is that fewer functions are recompiled (only the ones you changed, not all the functions on the file).

If you normally load the compiled file, then LOADFROM the source, and are running with DFNFLG = PROP, so that edited functions are not unsaved, then the effect of the change is that the edited functions do get recompiled, even though they are not defined by EXPRs.

Disadvantages of RECOMPILEDEFAULT = CHANGES:

If you go thru several rounds of the edit-makefile-recompile loop, then possibly MORE functions are recompiled than necessary, since each RECOMPILE will compile ALL the functions that have changed since you first LOADFROMed the file, not just the ones changed since the last recompile.

When Masterscope advises you to UNSAVEDEF a set of functions containing occurrences of records or macros that changed, the unsaving will have NO effect on which functions later get recompiled. You need to set RECOMPILEDEFAULT = EXPRS in order for this to work right.

- **Incompatible change: Compiling with mode = ST or STF redefines functions, even if DFNFLG = PROP**

Previously, when the compiler "redefined" a function, it respected the value of DFNFLG. If DFNFLG = PROP, the compiler put the new definition on the CODE property instead of in the definition cell of the function.

The new behavior is that as functions are compiled, they really ARE "stored and redefined"; the new compiled definition is placed in the definition cell, even though DFNFLG = PROP.

The new behavior is less confusing, but if you are used to the old behavior, be careful. If you run with DFNFLG = PROP to completely avoid inadvertantly redefining something in your running system, you MUST use compile mode F, not ST.

- **Warning: Compiler modified, so don't load Harmony-compiled files into old sysouts.**

A number of modifications have been made to the compiler, which might cause backward incompatibility. In general, old compiled code will work in new releases of Interlisp-D, but compiling in a NEW release and loading into an OLD release is not guaranteed to work.

- **STore-and-Forget option to COMPILE no longer leaves EXPRs on property list**
- **LOADTIMECONSTANT** works in interpreted code
- **Compiler prints warning if user code attempts to bind a variable previously declared as a constant**

- **Masterscope CHECK command smarter about CONSTANTS, blocks**

The CHECK command now knows about CONSTANTS. Previously, constants were treated like any other variable, and CHECK printed a warning if they were used freely without being declared. Also, CHECK now omits the preamble "in no block" (followed by a list of functions) when a file has no block declarations.

- **Show Paths browser properly updated when redisplayed**

When a function in a SHOW PATHS browser graph is edited, the window "greys out", to indicate that (possibly) some of the information has changed. Previously, under some circumstances, when a greyed out browser window was redisplayed, Masterscope would not reanalyze the functions that had changed.

- **". SHOW WHERE X CALLS Y" now finds lowest (not highest) level macro containing call**

- **Masterscope HELP command removed**

This used to print out a two-page summary of the Masterscope commands, which was not very useful in finding out how to use Masterscope.

(This page intentionally left blank)

- **Advance Warning: In future releases, (CLISPDEC 'MIXED) will be default**

In past releases of Interlisp, and in the Harmony release, the default clisp declaration is FIXED, which means that all clisp constructs are translated using integer arithmetic, unless the user explicitly changes the declaration. Therefore, (A + B) translates into (IPLUS A B), and (for X from A to B do ...) is translated using integer arithmetic to increment X and compare it to B.

In Interlisp-D, mixed (generic) arithmetic is not appreciably slower than integer arithmetic, so we are trying to convert the system to use generic arithmetic as much as possible.

Therefore, starting with the next release, the default clisp declaration will be MIXED, so generic arithmetic functions will be used when translating clisp constructs. (A + B) will translate into (PLUS A B), and (for X from A to B do ...) will be translated using PLUS and GREATERP. Of course, the user can change this declaration using CLISPDEC.

We do not expect that this change will effect any programs: the only conceivable problems could be in constructs like (for X from A to B do ...) where the programmer COUNTED on floating A and B being converted to fixed point before the loop.

- **Macro-expansion now independent of DWIM**

Previously, macro-expansion was handled by the MACROTRAN entry on DWIMUSERFORMS. This meant that macros would only be interpreted if DWIM was turned on. The macro-expansion machinery has been moved to a much 'higher' level (closer to the source), before DWIMFLG is tested and a large amount of otherwise unnecessary processing was done. This means that macro expansion can continue even when users turn off DWIM.

- **New variable DWIMINMACROSFLG controls whether args to macros are dwimified**

If the variable DWIMINMACROSFLG = T (the default), DWIM will recursively dwimify the arguments to macros (i.e. macros will be treated like LAMBDA functions). If DWIMINMACROSFLG = NIL, arguments to macros are not dwimified.

To provide finer control over the interpretation of individual macros, DWIM uses the INFO property of the macro name: If the INFO prop is or contains the atom EVAL, the macro arguments are dwimified, even if DWIMINMACROSFLG = NIL. If the INFO prop is or contains the atom NOEVAL, the macro arguments are not dwimified, even if DWIMINMACROSFLG = T.

- **DWIM no longer tries to interpret type-in as edit commands**

Previously, one of the actions DWIM took on unbound atom or undefined function errors was: "if the atom is an edit command, envoke the editor on the last thing edited, passing the atom as an edit command". DWIM is of necessity 'heuristic', attempting to second guess what the user meant. However, this correction is one that, over time, has become wrong far more often than right.

- **Incompatible Change: DWIMIFYENGLISH, CLISPENG package totally de-supported**

The "feature" of translating English into Lisp documented in the 1978 Interlisp Reference Manual is no longer supported in Interlisp-D. The lispusers package CLISPENG is no longer supported, either.

- **DWIM tries upper-casing undefined functions and unbound atoms**

- **DWIM now gives warning on coercion from lower to upper case**

Previously, DWIM would upper-case atoms and functions without warning or notification, which caused a great deal of confusion. Now, the default is to print a warning " = XX" when coercing from "xx" to "XX". This feature is controlled by the variable FIXSPELL.UPPERCASE.QUIET (initially NIL). If non-NIL, no warning is given.

- **CLISPIFY does not translate (fetch A.B of X) to X:A.B**

In the case where a record field has a period in it, it is inappropriate for CLISPIFY to translate a fetch or replace statement into the more concise form X:A.B, since DWIM interprets "A.B" as the "data path" rather than the field name.

- **RUNONFLG initialized to NIL in the default environment**

If the variable RUNONFLG = T, DWIM will attempt "run-on" spelling corrections, breaking up unknown names. From experience, it seems that this hurts more often then it helps. Therefore, the default has been changed so this feature is initially disabled.

- **FIXSPELL only moves words on "real" spelling lists**

When spelling-correcting words on the system spelling lists SPELLINGS1, SPELLINGS2, etc, FIXSPELL moves words to the front of the list when a word is successfully corrected. However, this is not necessarily the correct behavior for user-supplied spelling lists, where it may be wrong to alter the order of the list. If FIXSPELL is called with DONTMOVETOPFLG = non-NIL, words are not moved in the spelling list. As an additional check, FIXSPELL won't move correct words to the front of a spelling list unless the spelling list contains the special marker used to separate the temporary and permanent sections of the system spelling lists (the value of SPELLSTR1).

- **I.S.OPRS work even if CLISPFLG = NIL**

Contrary to the documentation, some iterative statement operators would not be translated correctly is CLISPFLG was NIL, because their definition included forms such as $$VAL←T. These operators now work even if "←" is disabled, either specially or because CLISPFLG is NIL.

(This page intentionally left blank)

- **DOSTATS removed from standard Lisp loadup**

Since the SPY package provides most of the functionality of DOSTATS, in addition to being usable on Xerox 1108's, the function DOSTATS has been removed from the standard Interlisp system. The code for DOSTATS is available by loading in the library files PCALLSTATS and APS (automatically loaded by PCALLSTATS).

- **DOSTATS now resets DFNFLG and compiler optimizations**

Previously, it was possible that DOSTATS would collect stats on the wrong form if DFNFLG was set improperly. For example, if DFNFLG = PROP, the form would be put on a property list, and stats would be collected for whatever happened to be in the definition cell of STATSDUMMYFUNCTION. Also, DOSTATS didn't reset compiler optimizations, so that it might "optimize" forms like (IQUOTIENT 1234567 -1) into a constant.

- **Control-D out of DOSTATS stops statistics-gathering**

Previously, typing control-D during the execution of DOSTATS would stop the computation, but wouldn't stop the gathering of statistics. This was a serious problem, because very quickly the disk would fill up and Interlisp would fall into SWAT, losing everything. Now, exiting DOSTATS with control-D automatically turns off statistics-gathering.

- **TIMEALL now compiles form with optimizations ON**

If TIMEALL is called with #TIMES>1, a dummy form is created, compiled and executed #TIMES times, to provide more accurate measurement of small computations. Previously, this compilation was done with optimizations off if running multiple times. In the face of objections, this has been changed: now TIMEALL compiles the dummy form with compiler optimizations ON.

- **Warning:** An important result of this change is that it is not meaningful to use TIMEALL with very simple forms that are optimized out by the compiler. For example, (TIMEALL '(IPLUS 2 3) 1000) will time a compiled function which simply returns the number 5, since (IPLUS 2 3) is optimized to the integer 5.

- **BREAKDOWN overhead reduced**

The per-call overhead to BREAKDOWN has been substantially reduced, which should give much more meaningful results.

STORAGE & DATA TYPES

- **Incompatible Change: ARRAY default type is POINTER, FLOATP is stored unboxed**

If NIL is given as the TYPE argument to ARRAY, the default array type is POINTER, not DOUBLEPOINTER. Anyone using the DOUBLEPOINTER mechanism should change any instances of (ARRAY x) to (ARRAY x 'DOUBLEPOINTER).

Arrays of type FLOATP are now stored unboxed. This increases the space and time efficiency of FLOATP arrays. Users who want to use boxed floating point numbers should use an array of type POINTER instead of FLOATP.

- Advance Warning: **CAR or CDR of non-list will cause error in future releases; new variable CAR/CDRERR**

According to the Interlisp Reference Manual, the value of applying the functions CAR and CDR to a non-list (other than NIL) is undefined. In Interlisp-D, the actual action depended on the data type: (CAR <atom>) returned NIL, (CDR <atom>) returned the atom's property list, (CAR <anything else>) returned the string "{car of non-list}", and (CDR <anything else>) returned the string "{cdr of non-list}".

This has turned out to be a bad design. This design typically caused obscure bugs in programs which CDR down a list, and stop on NIL. If the tail of the list is not NIL, then the program loops endlessly, taking CDR of "{cdr of non-list}". This problem also occurs with functions like (FMEMB A B), which loop endlessly if B is not a list.

Because of these problems, the Interlisp maintainers decided that CAR and CDR should cause an error on non-lists. Places in the system code which used the old conventions have been cleaned up. In future releases, the default will be changed so that CAR or CDR of non-NIL non-lists will cause errors. This will also effect system functions, such as FMEMB, which use CAR and CDR. User programs which depend on the old conventions will have to be modified.

To root out functions in the system which rely on the old CAR/CDR convention, the global variable CAR/CDRERR has been created.

If CAR/CDRERR = NIL (the current default), then CAR and CDR act as they always have, returning a string for non-lists. If CAR/CDRERR = T, then CAR and CDR of a non-list (other than NIL) causes an error.

If CAR/CDRERR = ONCE, then CAR and CDR of a string causes an error, but CAR/CDR of anything else returns the string "{c...r of non-list}" as before. This catches loops which repeatedly take CAR or CDR of an object, but it allows one-time errors to pass undetected.

If CAR/CDRERR = CDR, then CAR of a non-list returns "{car of non-list}" as before, but CDR of a non-list causes an error. This setting is based on the observation that nearly all infinite loops involving non-lists occur from taking CDRs, but a fair amount of careless code takes CAR of something it has not tested to be a list

- **MKATOM no longer loops forever when the atom hash table is full**

Previously, running out of atoms (the limit is currently ~32K) would cause an infinite loop. Now, Interlisp will cause a storage full error when there are about 7 "pages" of atom space left, and will call RAID (MP 9323 on an 1108) when there are no more atoms left.

- **Hash arrays have been totally reimplemented; better performance, interface**

The hash array facility has been totally reimplemented, to improve performance and provide a better interface to the overflow behavior. Old programs using hash arrays will still work, but not as efficiently as if they were recoded to take advantage of the new implementation.

In the old implementation, the hash array functions accepted a list whose CAR was a hash array datum. If the hash array overflowed during some hash array operation, the action taken (error, automatically enlarging the hash array, etc.) was determined by the CDR of the hash array list.

In the new implementation, the "overflow method" is stored as part of the hash array datatype. The hashing functions will operate correctly on "old-style" hash arrays of the form (harrayp . overflow), but more slowly than with "new-style" hash arrays that contain their overflow methods.

New functions:

(HASHARRAY MINKEYS OVERFLOW)
Creates a hash array containing at least MINKEYS hash keys, with overflow method OVERFLOW (if NIL, the default overflow method is to expand the size of the hasharray and rehash all the entries). The function HARRAY still exists for backward compatibility, equivalent to (HASHARRAY MINKEYS 'ERROR).

(HARRAYPROP HARRAY PROP NEWVALUE)
Returns the property PROP of HARRAY; PROP can have the system-defined values SIZE (returns the maximum occupancy of HARRAY), NUMKEYS (number of occupied slots), or OVERFLOW (overflow method). In the case of OVERFLOW, a new method may be specified as NEWVALUE.

(HASHARRAYP X)
Returns X if X is either an old- or new-style hash array (i.e a hash array datum or a list whose car is a hash array datum). Otherwise returns NIL.

(HARRAYP X)
Returns X if it is a hash array datum, as returned by the function HARRAY or HASHARRAY. Unlike HASHARRAYP, this returns NIL for lists whose CAR is a hash array datum. HASHARRAYP should probably be used instead in most circumstances.

- **STORAGE changes: new arguments; prints free list info**

The function STORAGE in Interlisp-D now takes two optional arguments for filtering the amount of information presented:

(STORAGE TYPES PAGETHRESHOLD)
If TYPES is given, STORAGE only lists statistics for the specified types. TYPES is an atom or list of types. If PAGETHRESHOLD is given, then STORAGE only lists statistics for types that have at least PAGETHRESHOLD pages allocated to them.

Note: These optional arguments are different from the optional arguments to STORAGE in Interlisp-10.

STORAGE now prints out more information about the size of the entries on the array free list, including a breakdown of the free block sizes. The block sizes are broken down by the value of the variable STORAGE.ARRAYSIZES, initially (4 10 100 1000 4000 NIL), which yields a printout of the form:

```
variable-datum free list:
     le 4      11  items;     44  cells.
     le 10     34  items;    240  cells.
     le 100    39  items;   1619  cells.
     le 1000   25  items;   7856  cells.
     le 4000    2  items;   2449  cells.
     others     0  items;      0  cells.
```

This information can be useful in determining if the variable-length data space is fragmented. If most of the free space is composed of small items, then the allocator may not be able to find room for large items, and will extend the variable datum space. If this is extended too much, this could cause an ARRAYS FULL error, even if there is lots of space left in little chunks. This information is primarily of use to system programmers.

- **New CASEARRAY arg for STRPOS**

STRPOS has been extended to take a new argument CASEARRAY. If non-NIL, this should be a casearray like that given to FILEPOS. The casearray is used to map the string characters before comparing them to the search string. See the documentation for FILEPOS, CASEARRAY, etc. in the reference manual.

- **New BACKWARDSFLG arg for STRPOS, STRPOSL**

If non-NIL, this argument specifies that the search should be done backwards from the end of the string.

- **Incompatible Change: LDIFFERENCE always returns copy of list: resolves Interlisp-D/10 difference**

Previously, if (LDIFFERENCE FOO BAR) was EQUAL to FOO (ie, FOO and BAR shared no elements), Interlisp-D would return a result which is EQ to FOO, while Interlisp-10 would return a copy of FOO. Interlisp-D has been changed to make it compatible with the Interlisp-10 behavior.

- **Interpreted REPLACE of a data with a BITS field now correct.**

Previously, the interpreted version of REPLACEFIELD would do the wrong thing if called to replace a datatype declared with a BITS field. This only affected interpreted calls to REPLACE and not compiled calls.

(CREATE ... SMASHING ...) translates into more efficient form

The translation of (CREATE ... SMASHING ...) forms has been changed for RECORD and TYPERECORD records, to produce forms that execute more efficiently when compiled.

- **(APPEND '(A . B)) now runs correctly when compiled**

Previously, (APPEND '(A . B)) returned (A . B) when interpreted, (A) compiled. Now, it returns (A . B) always.

- **ELT, SETA error changed from "ILLEGAL ARG" to "ARG NOT ARRAY"**

ARITHMETIC

- **Advance Warning: (Overflow default will be changed from (OVERFLOW 0) to (OVERFLOW T)**

In the Interlisp Reference Manual, (ZEROP X) is defined to be equivalent to (EQ X 0). Some users have complained that this is inconsistent with other lisp dialects, and that (ZEROP 0.0) should not return NIL. In a future release, (ZEROP X) will be equivalent to (EQP X 0). Users who depend on (ZEROP 0.0) returning NIL should change their code to use (EQ X 0).

In Interlisp-D, the action taken on arithmetic overflow is globally determined by the function OVERFLOW (described below). Currently, the default setting is (OVERFLOW 0), which signifies that arithmetic overflow and division by zero will not cause an error. In a future release, this default will be changed to (OVERFLOW T) so arithmetic overflow and division by zero will cause an error. Users are encouraged to run their programs with (OVERFLOW T), and to change any code which depend on overflow not causing an error.

(OVERFLOW FLG)
Sets a flag that determines the system response to arithmetic overflow and division by zero; returns the previous setting.

For integer arithmetic: If FLG = T, an error occurs on integer overflow or division by zero. If FLG = NIL, the largest integer is returned as the result of the overflowed computation. If FLG = 0, the result is returned modulo 2↑32 (the default action). If FLG = NIL or 0, integer division by zero returns zero.

For floating point arithmetic: If FLG = T, an error occurs on floating overflow or floating division by zero. If FLG = NIL or 0, the largest floating point number is returned as the result of the overflowed computation or floating division by zero.

- **Advance Warning: (ZEROP X) = (EQ X 0); will be equivalent to (EQP X 0)**

In the Interlisp Reference Manual, (ZEROP X) is defined to be the same as (EQ X 0). Some users have complained that this is inconsistant with other lisp dialects, and that (ZEROP 0.0) should not return NIL. In a future release, (ZEROP X) will be changed to be the same as (EQP X 0). Users who depend on (ZEROP 0.0) returning NIL should change their code to use (EQ X 0).

- **FPLUS, FTIMES call microcode when interpreted**

Previously, the functions FPLUS and FTIMES, when called from the interpreter, didn't go thru the microcoded opcodes but always executed the lisp macrocode.

- **Internal function FTIMES2 no longer defined**

In an old version of the compiler, the function FTIMES was compiled into a call to the function FTIMES2, which has been removed. Some programs compiled in 1982 apparently need recompilation before they will run; if you get UNDEFINED FUNCTION, FTIMES2, you should recompile the offending function.

- **(EXPT 3 -1) returns .333333333 instead of 0**

The manual states that (EXPT X Y) returns an integer if and only if X is an integer and Y is a non-negative integer.

- **New process property: BEFOREEXIT used to prevent LOGOUT**

If the process property BEFOREEXIT is the atom DON'T, it will not be interrupted by a LOGOUT. If LOGOUT is attempted before the process finishes, a message will appear saying that Interlisp is waiting for the process to finish. If you want the LOGOUT to proceed without waiting, you must use the process status window (from the background menu) to delete the process.

- **New process property: RESTARTFORM**

If the process property RESTARTFORM is non-NIL, it is the form used if the process is restarted (instead of the original form given to ADD.PROCESS). Of course, the process must also have a non-nil RESTARTABLE prop for this to have any effect.

- **Changes to DISMISS: new arg NOBLOCK**

(DISMISS MSECSWAIT TIMER NOBLOCK)
If MSECSWAIT and TIMER are both NIL, this is equivalent to (BLOCK). If NOBLOCK is T, DISMISS will not allow other processes to run, but will busy-wait until the amount of time given has elapsed.

- **Control-T does not cause a long DISMISS to return**

- **WAIT.FOR.TTY spawns mouse if called under the mouse process**

- **ADD.PROCESS property arguments interpreted correctly**

Previously, some combinations of arguments to ADD.PROCESS would be interpreted incorrectly. For example, (ADD.PROCESS <form> 'SUSPEND T) would create a (non-suspended) process with the name SUSPEND.

- **PROCESSPROP can remove last user-defined property from a process**

Previously, only the last property value, not both the name and value, would get removed from the list.

- **RESTART.PROCESS does not hang**

Previously, if RESTART.PROCESS was called on a process which has been created with SUSPEND = T and never started, this would cause Interlisp to hang (hard reset required).

1108 MICROCODE

- **1108 microcode available in 4K & 12K versions**

The 1108 hardware is now available with either of two processor boards: the standard board with a 4K microstore, or the Extended Processor Option (CPE) board with a 12K microstore. This does not change the installation or operation of Interlisp --- the Interlisp sysout contains microcode for both the microstore options, and the appropriate one is automatically loaded when Interlisp is started. To provide a visual indication of which size microstore is installed, the 1108 MP display will show 1109 when the 12K microcode is running (instead of 1108).

The 12K microcode contains a number of operations in microcode which were formally implemented in Lisp, so there is a performance improvement. For example, DRAWLINE, BIN, and MAKENUMBER are implemented in the 12K microcode. In the future, any announcements of 1108 microcode changes apply to BOTH microcodes, unless explicitly stated otherwise.

- **1108 microcode fixes**

A number of obscure microcode bugs, which could cause intermittent system failures, have been fixed.

- **Pressing 1108 STOP key in RAID will not crash Interlisp**

In some circumstances, pressing the STOP key when the 1108 was in RAID caused an unrecoverable error, whereas typing control-D would succeed. This was due to a microcode bug.

(This page intentionally left blank)

- **BUSEXTENDER, BUSMASTER: new, prototype packages for using high-speed parallel port on 1108 CPE board**

The extended 1108 CPE board includes a high-speed parallel port. Currently, hardware for using this parallel port is in development. BUSEXTENDER and BUSMASTER are the prototype versions of the software used for controlling this port. They are being made available to the user community to provide advance information to potential future users.

BUSEXTENDER contains the low-level Interlisp functions used to access the parallel port.

BUSMASTER is an application which uses BUSEXTENDER and special hardware (currently under development) to communicate to IBM PC- or Multibus-compatible peripheral devices.

- **CMLARRAY: the CMLARRAYS file package command now works as advertised**

- **CMLARRAY: INITIALCONTENTS property works correctly**

- **COLOR: LOGOUT no longer crashes if color display on**

- **COLOR: (COLORDISPLAY T) no longer breaks with "Illegal arg - NOBIND"**

- **FILEBROWSER: Totally rewritten; many improvements**

The most significant changes are:

The Info command has been removed, and the info window has been merged with the browser window. There is a menu of file properties under the main window; this selects the information to be fetched when the Update command is buttoned.

The Rename command now takes a default destination directory when called.

The Copy command now works when you're only copying a single file.

The See command pops up a scrollable window containing the listing of the file. (The old version's window didn't scroll). This window is reused for the next See command if it has been closed.

The file browser has its own prompt window, and no longer pops up superfluous windows.

If you close or shrink a file browser window and there are unexpunged deleted files, an "FB close options" menu will appear, asking whether or not to expunge deleted files before shrinking or closing the window.

The file browser window shrinks to a distinctive "file drawer" icon, which includes the current file browser pattern.

It uses a "nicer" font for the list of files.

Multiple file browsers can "do things" at the same time

Shift-selecting out of a file browser window will shift-select the full name of the file selected. Only one file can be shift-selected at a time.

Can supply new pattern to the filebrowser by middle-buttoning the UPDATE command, and selecting the "New Pattern" option from the menu that pops up.

The file browser window can scroll horizontally, so the user can see all of the properties listed. Above the browser window is a list of colume labels, which scroll horizontally as the browser window does.

- **FTPSERVER: Enumerating files on a remote machine running FTPSERVER now works correctly**

- **FTPSERVER: COPYFILE to remote 1108 won't cause MP 9318 error**

COPYFILE to a remote 1108 running FTPSERVER would sometimes cause a serious error. FTPSERVER has been fixed so this will not happen.

- **GRAPHER: Extensively revised; new function HARDCOPYGRAPH; node formatting extended.**

GRAPHER has been extensively revised, so that it uses much less memory space per node. Whereas the old Grapher created a bitmap per node, the new one doesn't. The price is that scrolling may take a little longer. To REDISPLAYW a very large graph takes twice as long as it used to (if you don't like this, set CACHE/NODE/LABEL/BITMAPS/FLG to T). Also, the GRAPHRECORD was changed to use half as many cons cells. This version will not run in Carol or older <lispcore> systems if the user depends on nodefonts being defaulted to the DEFAULTFONT font class.

(HARDCOPYGRAPH GRAPH/WINDOW FILE IMAGETYPE TRANS). Produces a file from a formated graph (e.g., like SHOWGRAPH, only for files). If GRAPH/WINDOW is a window, HARDCOPYGRAPH will operate on the GRAPH property of the window. If the device field of the file name is LPT, the file will automatically get sent to the appropriate printer. IMAGETYPE is either PRESS or INTERPRESS, and defaults to INTERPRESS. TRANS is a position in screen points of the lower left corner of the graph from the lower left corner of the piece of paper.

(DISPLAYGRAPH GRAPH STREAM CLIP/REG TRANS).
Put the specified graph on STREAM (which can be any image stream) with coordinates translated to TRANS. Some streams might also implement CLIP/REG as a clipping region. This is primarily an efficiency hack for the display.

GRAPHER now allows nodes to be "boxed" with borders of arbitrary shades and widths. Borders work for regular labels and bitmap labels, but not for imageobject labels. The old graphnode field BOXNODEFLG has been renamed NODEBORDER. It takes the following values:

NIL	no border, as before
T	black border, 1 pixel wide, as before
0	no border
,2,3...	black border of the given width
-1,-2...	white border of the given width
(w s)	where w is a fixp and s is a texture or a shade; yields a border w wide filled with the given shade s.

A new graphnode field, NODELABELSHADE, contains the background shade of the node. This allows GRAPHER to remember when a node is inverted. When a node is displayed, the label area for the node is first painted as specified by NODELABELSHADE, then the label is printed in INVERT mode. This does not apply to labels that are bitmaps or image objects. The legal values for the field are: NIL (same as WHITESHADE), T (same as BLACKSHADE), a texture, or a bitmap.

(RESET/NODE/BORDER <node> <border> <stream> <graph>)
(RESET/NODE/LABELSHADE <node> <shade> <stream>)
These functions reset the appropriate fields in the node. If <stream> is a displaystream or a window, the old node will be erased and the new node will be displayed. Changing the border may change the size of the node, in which case the lines to and from the node will be redrawn. The entire graph must be available to RESET/NODE/BORDER for this purpose, either supplied as the <graph> argument or obtained from the GRAPH property of <stream>, if it is a window. Both functions take the atom INVERT as a special value for

<border> and <shade>. They read the node's current border or shade, calculate what would be needed to invert it, and do so.

LAYOUTGRAPH previously used a 1-pixel black box to mark certain nodes in order to indicate where it had snapped links. That is still the default action. However, the appearance of marked nodes can be controlled by adding (MARK ....) to the FORMAT argument of LAYOUTGRAPH. The tail of (MARK ....) is a property list. If the property list is NIL, marking is suppressed altogether. If a BORDER property is specified, the value will be used as the NODEBORDER of marked nodes. If a LABELSHADE property is specified, its value will be used on the marked nodes. Of course, you can specify both a BORDER and NODELABEL property.

LAYOUTGRAPH will read, but not change, the fields NODEBORDER and NODELABELSHADE of the nodes given it (except for the marked nodes, of course). Thus, if one is planning on installing black borders around the nodes after the nodes have been layed out, its a good idea to give LAYOUTGRAPH nodes that have white borders. This will cause the nodes to be layed out far enough apart that when you blacken the borders later, the labels of adjacent nodes will not be overwritten.

When a graphnode is created by the record package, the default values are now taken from the value of the following variables:
    DEFAULT.GRAPH.NODEBORDER,
    DEFAULT.GRAPH.NODELABELSHADE, and
    DEFAULT.GRAPH.NODEFONT. GRAPHER
initializes these to NIL. To get the benefits of this new feature, the user will have to recompile functions that create graphnodes

FLIPNODE now inverts a region that is 1 pixel bigger all around than the node's region. This makes it possible to see black borders after the node has been flipped.

LAYOUTGRAPH takes a new format token. Adding REVERSE/DAUGHTERS to the list of format items will reflect horizontal graphs vertically, and vertical graphs horizontally.

- **LOGOCLOCK process restarts after HARDRESET**

- **SAMEDIR: MIGRATIONS modified: can now have list of directories**

- **SINGLEFILEINDEX: Printing process prevents LOGOUT until finished**

SINGLEFILEINDEX now spawns its process with the process property BEFOREEXIT = DON'T, so that it will not be interrupted by a LOGOUT. If LOGOUT is executed before the process finishes, a message will appear saying that Interlisp is waiting for the process to finish. If you want the LOGOUT to proceed without waiting, you must use the process status window (from the background menu) to delete the process.

- **SINGLEFILEINDEX: new variable \SINGLEFILEINDEX.DONTSPAWN**

If the global variable \SINGLEFILEINDEX.DONTSPAWN = NIL, SINGLEFILEINDEX will spawn a process to process and print the file. If the variable is non-NIL, the processing is done in the current process. When SINGLEFILEINDEX is loaded, \SINGLEFILEINDEX.DONTSPAWN is initialized to NIL if it is not already set.

- **SPY: "recursive merging" reworked, new functions)**

The SPY merge algorithm sometimes produced incorrect results when viewing recursive calls, like functions showing up at 200%. This has been fixed.

The macro WITH.SPY has been added, identical to the inconsistantly-named WITH-SPY.

(SPY.LEGEND) creates a window documenting the meaning of the different SPY node types.

(SPY.BUTTON) creates a button which, when touched once turns on SPY, touched again, turns it off and calls (SPY.TREE 10). This is useful for watching what's going on in the system without typing a lot.

- **SYSEDIT: EXPORTS.ALL (loaded by SYSEDIT) does not reset DIRECTORIES**

EXPORTS.ALL contains definitions for system records, and is used to edit system code. Previously, when this file was loaded, it would reset the variables DIRECTORIES and LISPUSERSDIRECTORIES to point to the directories used by the Interlisp-D maintainance group.

- **WHEREIS: Several changes to help users create and maintain their own databases**

Previously, the WHEREIS package interpreted the value of the variable WHEREIS.HASH as the full file name of the single hash file database to search. Now, WHEREIS.HASH is interpreted as a list of hash file names, to be searched in order. This allows the user to keep a number of separate WHEREIS databases for different projects. Also, instead of accepting the hash file filenames as fully-qualified filenames, they are found by searching the directories on DIRECTORIES. WHEREIS.HASH is initialized to NIL.

The function WHEREISNOTICE has also been extended, to help users create and maintain WHEREIS databases:

(WHEREISNOTICE FILEGROUP NEWFLG DATABASEFILE)
Inserts the information about all of the functions on the files in

FILEGROUP into the WHEREIS database 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 filegroup argument to DIRECTORY, so &, $, etc. may be used.

If NEWFLG is NIL, the information from the files in FILEGROUP is added to the database DATABASEFILE. If NEWFLG is non-NIL, a new version of DATABASEFILE will be created containing the database 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 20000 entries.

- **New variable MAKESYSNAME for identifying Interlisp-D releases**

In the Harmony release sysout, the variable MAKESYSNAME is set to the atom HARMONY. In future releases, this variable will be set to the current release name.

- **For Harmony Release, (LISPVERSION) = 37376**

Previously, the built-in version number was not consistantly changed for different releases of Interlisp-D. In future releases, the Interlisp version number will be incremented, and announced in the release message.

- **PROMPTFORWORD revised; doesn't grab TTY; argument renamed**

PROMPTFORWORD no longer grabs the tty stream by default. Like READ, if it is called in a process that is not the tty process, it waits for the user to click the mouse in its window, then grabs the tty.

The PROMPTFORWORD argument TIMELIMIT.secs has been renamed URGENCY.OPTION, which is interpreted as follows: If NIL, PROMPTFORWORD quietly wait for input, as READ does; if a number, this is the number of seconds to wait for the user to respond; if T, this means to wait forever, but periodically flash the window to alert the user; if TTY, then PROMPTFORWORD grabs the TTY immediately. When URGENCY.OPTION = TTY, the cursor is temporarily changed to a different shape to indicate the urgent nature of the request.

The last argument to PROMPTFORWORD, OLDSTRING, has been deleted.

Typing control-W now has the normal behavior (delete last word), rather than being a synonym of control-Q (delete all type-in).

PROMPTFORWORD only calls RINGBELLS once to attract the attention of the user.

- **Time-zone variables to control date printout: \TimeZoneComp, \BeginDST, \EndDST**

These variables are normally set automatically if you have a properly functioning time server on your net. For standalone machines, or old sysouts, you may need to set them by hand (in your init file) if you are not in the Pacific time zone. \TimeZoneComp is the number of hours west of Greenwich (negative if east); \BeginDST is the day of the year on or before which Daylight Savings Time takes effect (i.e., the Sunday on or immediately preceding this day); \EndDST is the day on or before which Daylight Savings Time ends. Days are numbered with 1 being January 1, and counting the days as for a leap year. In the USA where Daylight Savings Time is observed, \BeginDST = 121 and \EndDST = 305. In a region where Daylight Savings Time is not observed at all, set \BeginDST to 367.

- **(TIMEREXPIRED? X Y) documentation wrong**

If X and Y are variables whose values are timers, (TIMEREXPIRED? X Y) is true if X is set to an EARLIER time than Y. The Reference Manual was wrong: it said that it returned true if X was later than Y.

- **IDATE was wrong in March of leap year -- fixed**

- **GREET now asks for init file in typescript window**

In GREET, if the system can't find the file {DSK}INIT.LISP, the user is asked to type the name of the site initialization file. Previously, this prompt was printed in the prompt window. Now, the prompt is printed in the top level typescript window.

- **New function NORMALCOMMENTS for setting NORMALCOMMENTSFLG**

The interface for setting the "remote comment" facility has changed. The recommended way to enable and disable this facility is to call the new function NORMALCOMMENTS, rather than setting the variable NORMALCOMMENTSFLG.

(NORMALCOMMENTS NIL) enables the "remote comment" facility, and (NORMALCOMMENTS T) disables it (the default).

APPENDIX A

Hardcopy Facilities

Interlisp-D includes facilities for generating hardcopy in both "Press" and "Interpress" formats. Press is a file format used for communicating documents to Xerox prototype laser Xerographic printers known by the names "Dover", "Spruce", "Penguin", and "Raven". Interpress is a file format used for communicating documents to Xerox Network System printers such as the Xerox 8044 and Xerox 5700.

Files can be in a number of formats: Interpress and Press files, plain text files, and formatted Tedit files. In order to print a file on a given printer, it is necessary to identify the format of the file, convert the file to a format that the printer can accept, and transmit it. Rather than require that the user explicitly determine file types and do the conversion, the Interlisp-D hardcopy functions generate Press or Interpress output depending on the appropriate choice for the designated printer. The hardcopy functions use the variables **PRINTERTYPES** and **PRINTFILETYPES** (described below) to determine the type of a file, how to convert it for a given printer, and how to send it. By changing these variables, the user can define other kinds of printers and print to them using the normal hardcopy functions.

(SEND.FILE.TO.PRINTER *FILE HOST PRINTOPTIONS*) [*function*]

The function **SEND.FILE.TO.PRINTER** causes the file *FILE* to be sent to the printer *HOST*. If *HOST* is **NIL**, the first host in the list **DEFAULTPRINTINGHOST** which can print *FILE* is used.

PRINTOPTIONS is a property list of the form (*PROP1 VALUE1 PROP2 VALUE2* ...). Properties can include: **HEADING** - a string to use on the top of each page; **#COPIES** - the number of copies of the file to print; **#SIDES** - if 2, select two-sided printing (if *HOST* can print two-sided copies); **DOCUMENT.NAME** - the 'name' of the document, which often appears on a cover sheet. For example,

(SEND.FILE.TO.PRINTER 'FOO NIL

'(#COPIES 3 #SIDES 2 DOCUMENT.NAME "For John"))

SEND.FILE.TO.PRINTER calls **PRINTERTYPE** and PRINTFILETYPE to determine the printer type of *HOST* and the file format of *FILE*. If *FILE* is a formatted file (e.g., already in Press or Interpress format) in a form that the printer can print, it is transmitted directly. Otherwise, **CONVERT.FILE.TO.TYPE.FOR.PRINTER** is called to do the conversion. All of these functions use the lists **PRINTERTYPES** and **PRINTFILETYPES** to actually determine how to do the conversion.

LISTFILES calls the function **LISTFILES1** to send a single file to a hardcopy printing device. Interlisp-D is initialized with **LISTFILES1** defined to call **SEND.FILE.TO.PRINTER**.

Note: For backwards compatibility, the function **EMPRESS** is defined to pack its arguments into a list, and call **SEND.FILE.TO.PRINTER**.

(**HARDCOPYW** *WINDOW/BITMAP/REGION FILE HOST SCALEFACTOR ROTATION PRINTERTYPE)* [*function*]

Creates a hardcopy file from a bitmap and optionally sends it to a printer. *WINDOW/BITMAP/REGION* can either be a **WINDOW** (open or closed), a **BITMAP**, or a **REGION** (interpreted as a region of the screen). If **NIL**, the user is prompted for a screen region using **GETREGION**.

If *FILE* is non-**NIL**, it is used as the name of the file for output. If *HOST* = **NIL**, this file is not printed.

If *FILE* is **NIL**, output is sent to *HOST*. If *HOST* is **NIL**, the first host on **DEFAULTPRINTINGHOST** of the type *PRINTERTYPE* is used.

To save an image on a file without printing it, perform (**HARDCOPYW** *IMAGE FILE*). To print an image without saving the file, perform (**HARDCOPYW** *IMAGE*), or, to send it to a specific printer, (**HARDCOPYW** *IMAGE* **NIL** *PRINTER*).

SCALEFACTOR is a reduction factor. If not given, it is computed automatically based on the size of the bitmap and the capabilities of the printer type.

ROTATION specifies how the bitmap image should be rotated on the printed page. Most printers (including current **INTERPRESS** and **PRESS**) only support a *ROTATION* of multiples of 90.

PRINTERTYPE specifies what 'kind' of printer to use in environments that have more than one kind of printer around. For example, if you specify *PRINTERTYPE* to be **PRESS** or **INTERPRESS**, **HARDCOPYW** will use that information to select which printer to use or what print file format to convert the output into. If *PRINTERTYPE* is **NIL**, it defaults to **INTERPRESS**.

Note that the "Hardcopy" command in the background menu merely evaluates (**HARDCOPYW**), which prompts the user for a region on the screen, and sends the image to the default printer. The "Hardcopy" command in the paint menu performs (**HARDCOPYW** *WINDOW*), which sends an image of the whole window to the default printer.

DEFAULTPRINTINGHOST [Variable]

The variable **DEFAULTPRINTINGHOST** is used to designate the default printer to be used as the output of printing operations. It should be a list of the known printer host names, for example, (**QUAKE LISPPRINT**:). If an element of **DEFAULTPRINTINGHOST** is a list, is interpreted as (*PRINTERTYPE HOST*), specifying both the host type and the host name. The type of the printer, which determines the protocol used to send to it and the file format it requires, is determined by the function **PRINTERTYPE.**

If **DEFAULTPRINTINGHOST** is a single printer name, it is treated as if it were a list of one element.

(PRINTFILETYPE *FILE*) [Function]

Returns the format of the file *FILE*. Possible values include **PRESS, INTERPRESS, TEDIT**, etc. If it cannot determine the file type, it returns NIL. Uses the global variable **PRINTFILETYPES**.

(PRINTERTYPE *HOST*) [Function]

Returns the type of a printer *HOST*. Currently uses the following simple heuristic: printers whose name have a colon in them (e.g., **PRINTER:PARC:XEROX**) are assumed to be INTERPRESS printers. If *HOST* is a list, the **CAR** is assumed to be the printer type and CADR the name of the printer, e.g., (**PRESS LASSEN**). Otherwise, the printer is assumed to be the type which is the value of **DEFAULTPRINTERTYPE**, initially **PRESS**.

PRINTERTYPES [Variable]

The characteristics of a given printer are determined by the value of the list **PRINTERTYPES**. Each element is a list of the form

(*TYPES* (*PROPERTY1 VALUE1*) (*PROPERTY2 VALUE2*) ...)

TYPES is a list of the printer types that this entry addresses. The (*PROPERTYn VALUEn*) pairs define properties associated with each printer type.

PRINTERTYPES initially contains entries for the printer types **INTERPRESS** (or 8044), **PRESS (SPRUCE, PENGUIN, DOVER), FULLPRESS (RAVEN)**.

The printer properties include **CANPRINT** - a list of the file types that the printer can print directly; **STATUS** - a function that knows how to find out the status of the printer, used by **PRINTERSTATUS**; **PROPERTIES** - a function which returns a list of known printer properties; **SEND** - a function which invokes the appropriate protocol to send a file to the printer; **BITMAPSCALE** - a function of arguments *WIDTH* and *HEIGHT* in bits which returns a scale factor for scaling a bitmap; and **BITMAPFILE** - a form which, when evaluated, converts a **BITMAP** to a file format that the printer will accept.

PRINTFILETYPES [Variable]

The variable **PRINTFILETYPES** contains information about various file formats, such as Tedit files, Press files, and Interpress files. The format is similar to **PRINTERTYPES**. The properties that can be specified include **TEST** - a function which tests a file if it is of the given type; **CONVERSION** - a property list of other file types and ways to convert from one to the other; and **EXTENSION** - a list of possible file extensions for files of this type.

Hardcopy output may also be obtained by writing a file on the printer device **LPT**, e.g. **(COPYFILE 'FOO '{LPT})**. When a file on this device is closed, it is converted to Press or Interpress format (if necessary) and sent to the default printer (the first host on *DEFAULTPRINTINGHOST*). Thus, **{LPT}** acts like the device **LPT**: in Interlisp-10. One can include the printer name directly in the file name, e.g. **(COPYFILE 'FOO {LPT}QUAKE)** will send the file to the printer **QUAKE**.

APPENDIX B

Interlisp-D Attached Window Facility

The Interlisp-D attached window facility is a package designed to make it easy to manipulate a group of windows as a unit. Standard operations like **MOVE, RESHAPE, OPEN** and **CLOSE** can be done so that it appears to the user as if the windows are a single entity. Each collection of attached windows has one main window and any number of other windows that are "attached" to it. Moving or reshaping the main window causes all of the attached windows to be moved or reshaped as well. Moving or reshaping an attached window does not affect the main window. The initial motivation for attached windows was to allow multiple menus to be associated with the same window but there is no restriction on what windows can be attached.

(**ATTACHWINDOW** *WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE WINDOWCOMACTION*)

Associates *WINDOWTOATTACH* with *MAINWINDOW* so that shape, move, close, shrink and expand operations done to *MAINWINDOW* are also done to *WINDOWTOATTACH*. **ATTACHWINDOW** moves *WINDOWTOATTACH* to its position relative to *MAINWINDOW* but does not open it.

EDGE and *POSITIONONEDGE* indicate where *WINDOWTOATTACH* is to be positioned. The argument *EDGE* determines which edge: **TOP, BOTTOM, LEFT**, or **RIGHT**. The default, used if EDGE is NIL, is TOP. The argument *POSITIONONEDGE* determines where along edge the window is positioned:

JUSTIFY means that the attached window is to fill the entire edge. **ATTACHWINDOW** reshapes the window if necessary;

LEFT or **RIGHT** for the left or right (for a **TOP** or **BOTTOM** edge);

BOTTOM or **TOP** for the bottom or top (of a **LEFT** or **RIGHT** edge);

CENTER for the center of the edge.

The default for *POSITIONONEDGE* is **JUSTIFY**.

The size that is filled by the justification includes the extent of any other windows that have already been attached to *MAINWINDOW*. Thus **(ATTACHWINDOW A MW 'RIGHT 'JUSTIFY)** followed by **(ATTACHWINDOW B MW 'TOP 'JUSTIFY)** will put **B** across the top of both **MW** and **A**.

WINDOWCOMACTION provides a convenient way of setting the way the attached window responds to right buttoning. If *WINDOWCOMACTION* is **MAIN**, the **DOWINDOWCOMFN** of *WINDOWTOATTACH* is set to **DOMAINWINDOWCOMFN**. If *WINDOWCOMACTION* is **HERE**, the *DOWINDOWCOMFN* of *WINDOWTOATTACH* is not changed. If *WINDOWCOMACTION* is LOCALCLOSE, the DOWINDOWCOMFN of *WINDOWTOATTACH* is set to **DOATTACHEDWINDOWCOM2**. Otherwise, the *DOWINDOWCOMFN* of *WINDOWTOATTACH* is set to **DOATTACHEDWINDOWCOM**. These functions are described below in the section on "attached window menu commands."

(DETACHWINDOW *WINDOWTODETACH***)**

Detaches *WINDOWTODETACH* from its main window. Returns a dotted pair whose **CAR** is **EDGE** and whose **CDR** is **POSITIONONEDGE** if *WINDOWTODETACH* was an attached window. Returns **NIL** otherwise. This does not close *WINDOWTODETACH*.

Behavior on standard window operations

When a window operation, such as moving or clearing, is performed on a window, there is a question about whether or not that operation also be performed on the windows attached to it or performed on the window it is attached to. The following are the default behaviors of main and attached windows under the window operations when invoked programmatically, e.g., from the functions **MOVEW, SHAPEW,** etc.

The behavior when an operation is invoked from the window menu depends on the *WINDOWCOMACTION* argument to **ATTACHWINDOW,** or ultimately the window's **DOWINDOWCOMFN** property. Mention of "menu" below assumes that the window was attached using the default **(NIL)** *WINDOWCOMACTION.*

The behavior for any particular operation can, of course, be changed for particular windows by setting the standard window properties (e.g., **MOVEFN** or **CLOSEFN)** of the particular attached window.

Move: If the main window moves, all attached windows move with it, and the relative positioning between the main window and the attached windows is maintained. If the region is determined interactively, the prompt region for the move is the union of the extent of the main window and all attached windows.

MOVEW moves an attached window without affecting the main window. The Move command in the window menu is by default passed on to the main window, so that all windows in the group move.

Reshape: If the main window is reshaped, the minimum size of it and all of its attached windows is used as the minimum of the space for the result. Any space greater than the minimum is distributed among the main window and its attached windows.

SHAPEW reshapes an attached window independently. The Shape command in the window menu is by default passed on to the main window.

Close: If the main window is closed, all of the attached windows are closed also and the links from the attached windows to the main window are broken. This is necessary for the windows to be garbage collected.

CLOSEW closes an attached window without affecting the main window. Close in the window menu is by default passed on to the main window. If *WINDOWCOMACTION* of **LOCALCLOSE** was specified in the call to **ATTACHWINDOW**, the menu Close operates independently. Note that closing an attached window does *not* detach it.

Open: If the main window is opened, it opens all attached windows and reestablishes links from them to the main window.

Attached windows can be opened independently and this does not affect the main window.

Shrink: The collection of windows shrinks as a group. The **SHRINKFNs** of the attached windows are evaluated but the only icon displayed is the one for the main window.

Redisplay: The main or attached windows can be redisplayed independently.

Totop: If any main or attached window is brought to the top, all of the other windows are brought to the top also.

Expand: Expanding any of the windows expands the whole collection.

Scrolling: All of the windows involved in the group scroll independently.

Clear: All windows clear independently of each other.

The question of how to handle the window command menu of any particular window (either by right buttoning or by a call to the function **DOWINDOWCOM**) is handled by the window's

Attached window menu commands

DOWINDOWCOMFN property. The *WINDOWCOMACTION* argument to **ATTACHWINDOW** selects one of the following three functions to be the attached window's **DOWINDOWCOMFN** property, or leaves the property **NIL** if *WINDOWCOMACTION* is **HERE**, meaning to use the standard window command menu, ignorant of the window's attachments. The programmer can instead supply her own **DOWINDOWCOMFN** property if some other behavior is desired.

(**DOATTACHEDWINDOWCOM** *ATTACHEDW LOCALCLOSEFLG*)

The default (when *WINDOWCOMACTION* is **NIL**). Brings up the window command menu and then, depending upon the command selected, either passes the command to the main window of *ATTACHEDW* or performs it on *ATTACHEDW*. If *LOCALCLOSEFLG* is non-**NIL**, the **CLOSE** command is applied to *ATTACHEDW*. Otherwise, the **CLOSE** command is passed to the main window. The commands **MOVE, RESHAPE, SHRINK** and **BURY** are passed to the main window. The other commands are performed on *ATTACHEDW*.

(**DOATTACHEDWINDOWCOM2** *ATTACHEDW*)

Used when *WINDOWCOMACTION* is **LOCALCLOSE**. Performs (**DOATTACHEDWINDOWCOM** *ATTACHEDW* **T**) so that the command **CLOSE** is performed on *ATTACHEDW*.

(**DOMAINWINDOWCOMFN** *ATTACHEDW*)

Used when *WINDOWCOMACTION* is **MAIN**. Performs **DOWINDOWCOM** on the window that is the **MAINWINDOW** property of *ATTACHEDW*. In other words, assuming the **DOWINDOWCOMFN** of the main window hasn't been changed, this passes all window commands on to the main window.

Attaching menus to windows

ATTACHEDWINDOW supersedes the **MENUEDWINDOW** package and users of it are encouraged to convert their code. The following functions are provided to associate menus to windows.

(**ATTACHMENU** *MENU MAINWINDOW EDGE POSITIONONEDGE NOOPENFLG*)

Creates a window that contains the menu *MENU* (by calling **MENUWINDOW**, see below) and attaches it to the window *MAINWINDOW* on edge *EDGE* at position *POSITIONONEDGE*. The menu window is opened unless *MAINWINDOW* is closed, or *NOOPENFLG* is **T**

(**MENUWINDOW** *MENU VERTFLG*)

Returns a closed window that has the menu *MENU* in it. If *MENU* is a list, a menu is created with *MENU* as its items. Otherwise, *MENU* should be a menu. The returned window has the appropriate **RESHAPEFN, MINSIZE** and **MAXSIZE** window properties to allow its use in a window group. *VERTFLG* is provided to allow convenient setting of the default menu shape and will only be considered if both the **MENUROWS** and **MENUCOLUMNS** fields of *MENU* are **NIL**. If *VERTFLG* is non-**NIL**, the **MENUROWS** field of *MENU* will be set to 1; otherwise the **MENUCOLUMNS** field of *MENU* will be set to 1.

(**CREATEMENUEDWINDOW** *MENU WINDOWTITLE LOCATION WINDOWSPEC*)

Creates a window with an attached menu and returns the main window. *MENU* is the only required argument, and may be a menu or a list of menu items. *WINDOWTITLE* is a string specifying the title of the main window. *LOCATION* specifies the edge on which to place the menu, as with **ATTACHWINDOW**'s *EDGE* argument; the default is **TOP**. *WINDOWSPEC* is a **REGION**, specifying a region for the aggregate window; if **NIL**, the user is prompted for a region.

This function is similar to **MENUEDWINDOW**'s function **MAKEMENUEDWINDOW**. However, note that it is not an exact replacement. In particular, the **MENUWINDOW** property is not used.

Examples:

```
(SETQ MENUW
   (MENUWINDOW (create MENU
                       ITEMS ← '(smaller LARGER)
                       MENUFONT ← '(GACHA 12)
                       TITLE ← "zoom controls"
                       CENTERFLG ← T
                       WHENSELECTEDFN ← (FUNCTION
                           ZOOMMAINWINDOW))))
```

creates a menu window that contains the two items "smaller" and "**LARGER**" with the title "zoom controls" and that calls the function **ZOOMMAINWINDOW** when an item is selected.

(ATTACHWINDOW MENUW (CREATEW '(50 50 150 150)) **'TOP 'JUSTIFY)**

creates a window on the screen and attaches the above created menu window to its top.

```
(CREATEMENUEDWINDOW (CREATE MENU
                            ITEMS ← '(smaller LARGER)
                            MENUFONT ← '(GACHA 12)
                            TITLE ← "zoom controls"
                            CENTERFLG ← T
                            WHENSELECTEDFN ← (FUNCTION
                               ZOOMMAINWINDOW))))
```

creates the same sort of window in one step, prompting the user for a region.

Attached Prompt Windows

Many packages have a need to display status information or prompt for small amounts of user input in a place outside their standard window. A convenient way to do this is to attach a small window to the top of the program's main window. The following functions do so in a uniform way that can be depended on among diverse applications.

(GETPROMPTWINDOW *MAINWINDOW #LINES FONT DONTCREATE*)

Returns the attached prompt window associated with *MAINWINDOW*, creating it if necessary. The window is always attached to the top of *MAINWINDOW*, has **DSPSCROLL** set to **T**, and has a **PAGEFULLFN** of **NILL** to inhibit page holding. The window is at least *#LINES* lines high (default 1); if a pre-existing window is shorter than that, it is reshaped to make it large enough. *FONT* is the font to give the prompt window (defaults to the font of *MAINWINDOW*), and applies only when the window is first created. If *DONTCREATE* is true, returns the window if it exists, otherwise **NIL** without creating any prompt window.

(REMOVEPROMPTWINDOW *MAINWINDOW*)

Detaches the attached prompt window, if any, associated with *MAINWINDOW*, and closes it.

Window properties of attached windows

Windows that are involved in a collection either as a main window or as an attached window have properties stored on them. The only properties that are intended to be set be set by the user are the **MINSIZE** and **MAXSIZE** properties. The other properties should be considered read only; they are maintained by the **ATTACHEDWINDOW** package.

MINSIZE, MAXSIZE: Each should be a dotted pair (width . height) or a function to apply to the window that returns a dotted pair. The numbers are used when the main window is reshaped. The MINSIZE is used to determine the size of the smallest region acceptable during reshaping. Any amount greater than the collective minimum is spread evenly among the windows until each reaches MAXSIZE. Any excess is given to the main window. This algorithm may change as experience is gained.

Note: If you give the main window of an attached window group a **MINSIZE** or **MAXSIZE** property, its value is moved to the **MAINWINDOWMINSIZE** or **MAINWINDOWMAXSIZE** property, so that the main window can be given a size function that computes the minimum or maximum size of the entire group. Thus, if you want to change the main window's minimum or maximum size after attaching windows to it, you should change the **MAINWINDOWMINSIZE** or **MAINWINDOWMAXSIZE** property instead.

Note: This doesn't address the hard problem of overlapping attached windows side to side, for example if window A was attached as **[TOP, LEFT]** and B as **[TOP, RIGHT]**. Initially the reshape getregion won't worry about the overlap.

Default **MAXSIZE** is **NIL**, which will let the region grow indefinitely.

MAINWINDOW: pointer from attached windows to the main window of the group. This link is not available if the main window is closed. The function **MAINWINDOW** is the preferred way to access this property.

ATTACHEDWINDOWS: pointer from a window to its attached windows. For functional access to this information, the function **ATTACHEDWINDOWS** is documented below.

WHEREATTACHED: for attached windows, a list whose first element is the **EDGE** and whose second element is the POSITIONONEDGE that determine the placement condition for this window.

The **TOTOPFN** property on attached windows and the properties **TOTOPFN, DOSHAPEFN, MOVEFN, CLOSEFN, OPENFN, SHRINKFN, EXPANDFN** and **CALCULATEREGIONFN** contain elements that implement the window manipulation facilities. Care should be used in modifying or replacing these properties.

Notes

A window can be attached to only one other window. Attaching a window to a second window will detach it from the first.

Attachments can not form loops. That is, a window cannot be attached to itself or to a window that is attached to it. **ATTACHWINDOW** will generate an error if this is attempted.

Attached windows can have other windows attached to them. Thus, it is possible to attach window **A** to window **B** when **B** is already attached to window **C**. Similarly, if **A** has other windows attached to it it can still be attached to **B**.

Moving the main window will maintain the relationships between windows.

Reshaping the main window will restore the conditions established by the call to **ATTACHWINDOW**, moving the main window does not. Thus, if **A** is attached to the top of **B** and then moved by the user, its new position relative to **B** will be maintained if **B** is moved. If **B** is reshaped, **A** will be reshaped to the top of **B**. Additionally, if, while **A** is moved away from the top of **B**, **C** is attached to the top of **B**, **C** will position itself above where **A** used to be.

The attached windows can be closed by themselves. They will be reopened whenever the mainwindow is reshaped. The closefn for the main window breaks the links from the attached windows to it to allow them to be garbage-collected. The reopenfn for the main window reestablishes these links. Thus it is possible to reopen a closed, attached window and not have it linked to its mainwindow.

Example of use

(ATTACHWINDOW ATWIN MAINWIN 'TOP 'CENTER)

Will move **ATWIN** to immediately above **MAINWIN** and maintain its attachment there.

(ATTACHWINDOW NOTHERWIN MAINWIN 'TOP 'CENTER)

Will move **NOTHERWIN** to immediately above **ATWIN** and maintain its attachment there.

Miscellaneous functions

(MAINWINDOW *WINDOW RECURSEFLG*)

If *WINDOW* is not a **WINDOW**, it generates an error. If *WINDOW* is closed, it returns *WINDOW*. If *WINDOW* is not attached to another window, it returns *WINDOW* itself. If *RECURSEFLG* is **NIL** and *WINDOW* is attached to a window, it returns that window. If *RECURSEFLG* is **T**, it returns the first window up the "main window" chain starting at *WINDOW* that is not attached to any other window.

(WINDOWREGION *WINDOW*)

Returns the screen region occupied by *WINDOW* and its attached windows, if it has any.

(WINDOWSIZE *WINDOW*)

Returns the size of *WINDOW* (a dotted pair of width and height) and its attached windows, if it has any.

(MINATTACHEDWINDOWEXTENT *WINDOW*)

Returns the minimum extent (a dotted pair of width and height) of *WINDOW* and its attached windows (if any) will accept.

(ATTACHEDWINDOWS *MAINWINDOW*)

Returns the list of windows attached to this window.

(ALLATTACHEDWINDOWS *MAINWINDOW*)

Returns a list of all of the windows attached to *MAINWINDOW* or attached to a window attached to it.

(DETACHALLWINDOWS *MAINWINDOW*)

Detaches and closes all windows attached to *MAINWINDOW.*

APPENDIX C

Parallel Port for the 1100 and for the 1108 with Extended Processor Option (CPE)

The 1100 has a parallel port connector with 8 bidirectional data lines, 8 unidirectional output lines (with inverted duplicates for noise immunity), and 5 unidirectional input lines. The 1108 with Extended Processor Option (CPE) has a similar parallel port connector: the differences are (1) it has 6 unidirectional input lines vs. 5, (2) the power lines of the connector are 5 volts vs. 12, and (3) the pin layouts are "reversed". (These differences are also described below in the "Pin List".) A cable adapter is available to map the 1108's parallel port's pin layout into that of the 1100, or vice versa.

This document describes the protocol for the parallel port connector, and basic Interlisp-D functions for accessing it.

The maximum transfer rate when the parallel port is accessed directly from the Interlisp-D functions below is about 240K operations/second. Centronics-driver block-transfer microcode will be able to transfer data as fast as the Centronics standard will allow.

On the 1108, the parallel port connector is the CPE board's upper D-37 connector J2 (female), also cabled to the "PAR PORT" connector on the rear of the 1108.

Description of signals on the connector

PIO.0-7	Bidirectional data lines. Driven by the central processor if PO.7 = 1, input if PO.7 = 0.
PI.0-4/5	Input data lines. (PI.0-4 on the 1100; PI.0-5 on the 1108.)
PO.0-6	Output data lines.
PO.7	Output data line. Special in that: If PO.7 = 0 = 'output' then the bidirectional lines PIO.0-7 are driven by the central processor (and, on the 1108, a program reading the parallel port will read a set of miscellaneous status lines rather than the values of PIO.0-7 as on the 1100); If PO.7 = 1 = 'input' then the bidirectional lines PIO.0-7 are not driven by the central processor, and a program reading the parallel port will read the input values of PIO.0-7;

PO.0'-7' Output data lines. Inverted copies of PO.0-7.

Signals are TTL levels. All output lines are latched; input lines are read asynchronously.

On the 1108 only, other hardware has status flags "hidden" in the parallel port. Microcode for this other hardware must sometimes set PO.7 = 1 to enable access to these flags. As a result, the user device on the parallel port, and its handshaking protocol, must tolerate that at any time when PO.7 = 1 (PIO.0-7 output drivers disabled), PO.7 *may* be toggled to 0 then back to 1, briefly enabling the central processor's output drivers on PIO.0-7. This disruption is otherwise invisible to the parallel port user. This disruption can be avoided during uninterruptable microcode.

Pin List

pin no. (1100)	signal name	pin no. (1108)	pin no. (1100)	signal name	pin no. (1108)
1	PIO.0	19	20	PO.0	37
2	PIO.1	18	21	PO.0'	36
3	PIO.2	17	22	PO.1	35
4	PIO.3	16	23	PO.1'	34
5	PIO.4	15	24	PO.2	33
6	PIO.5	14	25	PO.2'	32
7	PIO.6	13	26	PO.3	31
8	PIO.7	12	27	PO.3'	30
9	PI.0	11	28	PO.4	29
10	PI.1	10	29	PO.4'	28
11	PI.2	9	30	PO.5	27
12	PI.3	8	31	PO.5'	26
13	PI.4	7	32	PO.6	25
14	12volt 5volt	6	33	PO.6'	24
15	12volt 5volt	5	34	PO.7	23
16	resv'd	4	35	PO.7'	22
17	GND	3	36	GND PI.5	21
18	GND	2	37	GND	20
19	GND	1			

The 1108's 5volt lines will drive about 1 amp.

Interlisp-D access

(**WRITEPRINTERPORT** *datum*)

Writes the less significant 16 bits of the integer *datum* to the parallel port. The format is: PO.0,..PO.7,PIO.0,..,PIO.7 (least significant bit). All outputs to the parallel port are latched. Note that the value written to PO.7 controls whether PIO.0-7 are driven (output) or passive (input). On the 1108, the value written to PO.7 also affects the results of **READPRINTERPORT**.

(**READPRINTERPORT**)

Reads a 16-bit datum from the parallel port, returning it as a positive integer. The more significant 6 bits of these 16 are (most significant first) PI.0,..,PI.5. The next less significant two bits are not relevant to parallel port i/o. On the 1100, the least significant 8 bits of the result are PIO.0,..,PIO.7 (least significant bit). On the 1108, the least significant 8 bits of the result are relevant to parallel port i/o only if the value most recently written to PO.7 was 1, in which case they are PIO.0,..,PIO.7 (least significant bit) as on the 1108.

(This page intentionaly left blank)

APPENDIX D

Higher-Level NS Protocol Functions

The following is a description of the Interlisp-D facilities for using Xerox SPP and Courier protocols and the services based on them. The sections on naming conventions, Printing, and Filing are of general interest to users of Network Systems servers. The remaining sections describe interfaces of interest to those who wish to program other applications on top of either Courier or SPP.

Name and Address Conventions

Addresses of hosts in the NS world consist of three parts, a network number, a machine number, and a socket number. These three parts are embodied in the Interlisp-D datatype **NSADDRESS**. Objects of type **NSADDRESS** print as "net#h1.h2.h3#socket", where all the numbers are printed in octal radix, and the 48-bit host number is broken into three 16-bit fields. Most functions that accept an address argument will accept either an **NSADDRESS** object or a string that is the printed representation of the address.

Higher-level functions accept host arguments in the form of a symbolic name for the host. The **NS** world has a hierarchical name space. Each object name is in three parts: the *Organization*, the *Domain*, and the *Object* parts. There can be many domains in a single organization, and many objects in a single domain. The name space is maintained by the *Clearinghouse*, a distributed network database service.

A Clearinghouse name is standardly notated as *object:domain:organization*. The parts *organization or domain:organization* may be omitted if they are the default (see below). Alphabetic case is not significant. Internally, names are represented as objects of datatype **NSNAME**, but most functions accept the textual representation as well, either as a litatom or a string. Objects of type **NSNAME** print as *object:domain:organization, with fields omitted when they are equal to the default. A Dom*ain is standardly represented as an **NSNAME** in which the object part is null. If frequent use is to be made of an NS name, it is generally preferable to convert it to an **NSNAME** once, by calling **PARSE.NSNAME**, then passing the resultant object to all functions desiring it.

CH.DEFAULT.ORGANIZATION [Variable]

This is a string specifying the default Clearinghouse organization.

CH.DEFAULT.DOMAIN [Variable]

This is a string specifying the default Clearinghouse domain. If it or the variable **CH.DEFAULT.ORGANIZATION** is **NIL**, they are set by Lisp system code (when they are needed) to be the first domain served by the nearest Clearinghouse server.

In small organizations with just one domain, it is reasonable to just leave these variables **NIL** and have the system set them appropriately. In organizations with more than one domain, it is wise to set them in the site initialization file, so as not to be dependent on exactly which Clearinghouse servers are up at any time.

(**PARSE.NSNAME** *NAME #PARTS DEFAULTDOMAIN*) [Function]

When *#PARTS* is **3** (or **NIL**), parses *NAME*, a litatom or string, into its three parts, returning an object of type **NSNAME**. If the domain or organization is omitted, defaults are supplied, either from *DEFAULTDOMAIN* (an **NSNAME** whose domain and organization fields only are used) or from the variables **CH.DEFAULT.DOMAIN** and **CH.DEFAULT.ORGANIZATION**.

If *#PARTS* is **2**, *NAME* is interpreted as a domain name, and an **NSNAME** with null object is returned. In this case, if *NAME* is a full 3-part name, the object part is stripped off.

If *#PARTS* is 1, *NAME* is interpreted as an organization name, and a simple string is returned. In this case, if *NAME* is a 2- or 3-part name, the organization is extracted from it.

If *NAME* is already an object of type **NSNAME**, then it is returned as is (if *#PARTS* is 3), or its domain and/or organization parts are extracted (if *#PARTS* is 1 or 2).

(**NSNAME.TO.STRING** *NSNAME FULLNAMEFLG*) [Function]

Converts *NSNAME*, an object of type **NSNAME**, to its string representation. If *FULLNAMEFLG* is true, the full printed name is returned; otherwise, fields that are equal to the default are omitted.

Programmers who wish to manipulate **NSADDRESS** and **NSNAME** objects directly should load the Library package **ETHERRECORDS**.

Clearinghouse Functions

This section describes functions that may be used to access information in the Clearinghouse.

(START.CLEARINGHOUSE *RESTARTFLG***)** [Function]

Performs an expanding ring broadcast in order to find the nearest Clearinghouse server, whose address it returns. If a Clearinghouse has already been located, this function simply returns its address immediately, unless *RESTARTFLG* is true, in which case the cache of Clearinghouse information is invalidated and a new broadcast is performed. **START.CLEARINGHOUSE** is normally performed automatically by the system the first time it needs Clearinghouse information; however, it may be necessary to call it explicitly (with *RESTARTFLG* set) if the local Clearinghouse server goes down.

CH.NET.HINT [Variable]

A number or list of numbers, giving a hint as to which network the nearest Clearinghouse server is on. When **START.CLEARINGHOUSE** looks for a Clearinghouse server, it probes the network(s) given by **CH.NET.HINT** first, performing the expanding ring broadcast only if it fails there. If the nearest Clearinghouse server is not on the directly connected network, setting **CH.NET.HINT** to the proper network number in the local **INIT** file can speed up **START.CLEARINGHOUSE** considerably.

(SHOW.CLEARINGHOUSE *ENTIRE.CLEARINGHOUSE? DONT.GRAPH***)** [Function]

This function displays the structure of the cached Clearinghouse information in a window. Once created, it will be redisplayed whenever the cache is updated, until the window is closed. The structure is shown using the Library package **GRAPHER**.

If *ENTIRE.CLEARINGHOUSE?* is true, then this function probes the Clearinghouse to discover the entire domain:organization structure of the Internet, and graphs the result. If *DONT.GRAPH* is true, the structure is not graphed, but rather the results are returned as a nested list indicating the structure.

(LOOKUP.NS.SERVER *NAME TYPE FULLFLG***)** [Function]

Returns the address, as an **NSADDRESS**, for the object *NAME*. *TYPE* is the property under which the address is stored, which defaults to **ADDRESS.LIST**. The information is cached so that it need not be recomputed on each call; the cache is cleared by restarting the Clearinghouse. If *FULLFLG* is true, returns a list whose first element is the canonical name of *NAME* and whose tail is the address list.

The following functions perform various sorts of retrieval operations on database entries in the Clearinghouse. Here, "The Clearinghouse" refers to the collective service offered by all the Clearinghouse servers on an internet; Lisp internally deals with which actual server(s) it needs to contact to obtain the desried information. The argument(s) describing the objects under consideration can be strings or **NSNAME**'s, and in most cases can contain the wild card "*", which matches a subsequence of zero or more characters. Wildcards are permitted only in the most specific field of a name (e.g., in the object part of a full three-part name). When an operation intended for a single object is instead given a pattern, the operation is usually performed on the first matching object in the database, which may or may not be interesting.

(CH.LOOKUP.OBJECT *OBJECTPATTERN***)** [Function]

Looks up *OBJECTPATTERN* in the Clearinghouse database, returning its canonical name (as an **NSNAME**) if found, **NIL** otherwise. If *OBJECTPATTERN* contains a "*", returns the first matching name.

(CH.LIST.ORGANIZATIONS *ORGANIZATIONPATTERN***)** [Function]

Returns a list of organization names in the Clearinghouse database matching *ORGANIZATIONPATTERN*. The default pattern is "*", which matches anything.

(CH.LIST.DOMAINS *DOMAINPATTERN***)** [Function]

Returns a list of domain names (two-part **NSNAME**'s) in the Clearinghouse database matching *DOMAINPATTERN*. The default pattern is "*", which matches anything in the default organization.

(CH.LIST.OBJECTS *OBJECTPATTERN PROPERTY***)** [Function]

Returns a list of object names matching *OBJECTPATTERN* and having the property *PROPERTY*. *PROPERTY* is a number'or a symbolic name for a Clearinghouse property; the latter include **USER, PRINT.SERVICE, FILE.SERVICE, MEMBERS, ADDRESS.LIST** and **ALL**.

For example,

(CH.LIST.OBJECTS "*:PARC:Xerox" (QUOTE USER))

returns a list of the names of users in the domain PARC:Xerox.

(CH.LIST.OBJECTS "*lisp*:PARC:Xerox" (QUOTE MEMBERS))

returns a list of all group names in **PARC:Xerox** containing the substring "lisp".

(CH.LIST.ALIASES *OBJECTNAMEPATTERN***)** [Function]

Returns a list of all objects in the Clearinghouse database that are aliases and match *OBJECTNAMEPATTERN*.

(CH.LIST.ALIASES.OF *OBJECT***)** [Function]

Returns a list of all objects in the Clearinghouse database that are aliases of *OBJECT*.

(CH.RETRIEVE.ITEM *OBJECT PROPERTY INTERPRETATION***)** [Function]

Retrieves the value of the *PROPERTY* property of *OBJECT*. Returns a list of two elements, the canonical name of the object and the value. If *INTERPRETATION* is given, it is a Clearinghouse type (see section X.XX) with which to interpret the bits that come back; otherwise, the value is simply of the form **(SEQUENCE UNSPECIFIED)**, a list of 16-bit integers representing the value.

(CH.RETRIEVE.MEMBERS *OBJECT PROPERTY* ---**)** [Function]

Retrieves the members of the group *OBJECT*, as a list of **NSNAME**'s. *PROPERTY* is Clearinghouse Group property under which the members are stored; the usual property used for this purpose is **MEMBERS**.

(**CH.ISMEMBER** *GROUPNAME PROPERTY SECONDARYPROPERTY NAME*) [Function]

Tests whether *NAME* is a member of *GROUPNAME*'s *PROPERTY* property. This is a potentially complex operation; see the description of procedure **IsMember** in the Clearinghouse Protocol documentation for details.

NS Printing

This section describes the facilities that are available for printing Interpress masters on NS Print servers.

(**NSPRINT** *PRINTER FILE OPTIONS*) [Function]

This function prints an Interpress master on *PRINTER*, which is a Clearinghouse name represented as a string or **NSNAME**. If *PRINTER* is **NIL**, **NSPRINT** uses the first print server registered in the default domain. *FILE* is the name of an Interpress file to be printed. *OPTIONS* is a list in property list format that controls details of the printing. Possible properties are as follows:

DOCUMENT.NAME	The document name to appear on the header page (a string). Default is the full name of the file.
DOCUMENT.CREATION.DATE	The creation date to appear on the header page (a Lisp integer date, such as returned by **IDATE**). The default value is the creation date of the file.
SENDER.NAME	The name of the sender to appear on the header page (a string). The default value is the name of the user.
RECIPIENT.NAME	The name of the recipient to appear on the header page (a string). The default is none.
#COPIES	The number of copies to be printed. The default value is 1.
MEDIUM	The medium on which the master is to be printed. If omitted, this defaults to the value of **NSPRINT.DEFAULT.MEDIUM**, as follows: **NIL** means to use the printer's default; **T** means to use the first medium reported available by the printer; any other value must be a Courier value of type **MEDIUM**. The format of this type is a list (**PAPER** (**KNOWN.SIZE** *TYPE*)) or (**PAPER** (**OTHER.SIZE** (*WIDTH LENGTH*))). The paper *TYPE* is one of **US.LETTER**, **US.LEGAL**, **A0** through **A10**, **ISO.B0** through **ISO.B10**, and **JIS.B0** through **JIS.B10**.
STAPLE?	True if the document should be stapled.

#SIDES or 2 to indicate that the document should be printed on one or two sides, respectively. The default is the value of **EMPRESS#SIDES**.

PRIORITY The priority of this print request, one of **LOW, NORMAL**, or **HIGH**. The default is the printer's default.

(NSPRINTER.STATUS *PRINTER*) [Function]

This function returns a list describing the printer's current status---whether it is available or busy, and what kind of paper is loaded.

(NSPRINTER.PROPERTIES *PRINTER*) [Function]

This function returns a list describing the printer's capabilities at the moment---type of paper loaded, whether it can print two-sided, etc.

NS Filing

Lisp accesses Xerox NS fileservers using the NS Filing Protocol. For most operations, the programmer simply treats the NS fileserver as any other host or device. **OPENFILE, GETFILEINFO, DIRECTORY** all work appropriately when the filename specifies an NS file server host name, e.g., **{PHYLEX:PARC:XEROX}<LISP>LIBRARY>GRAPHER.DCOM;1**. Note that all NS File Server host names must contain a colon, even if the domain and organization fields are defaulted, in order that they be distinguishable from other types of host names (e.g., Pup server names). Note also that spaces are allowable characters in NS names. Thus, if an NS file name contains spaces (in the file name or the server name) and is presented as a litatom, the spaces must be quoted with %. However, all the standard file operations also work when the name is presented as a string, in which case there is no problem with spaces.

The following are features specific to NS fileservers.

(BREAK.NSFILING.CONNECTION HOST) [Function]

Closes any open connections to NS file server *HOST*.

FILING.ENUMERATION.DEPTH [Variable]

The full **NS** Filing Protocol supports a true hierarchical name space for files. This leaves some ambiguity about how deep the function **DIRECTORY** should enumerate files. The depth is controlled by the variable **FILING.ENUMERATION.DEPTH**, which is either a number, specifying the number of levels deep to enumerate, or T, meaning enumerate to all levels. In the former case, when the enumeration reaches the specified depth, only the subdirectory name rooted at that level is listed, and none of its descendants is listed. When **FILING.ENUMERATION.DEPTH** is **T**, all files are listed, and no subdirectory names are listed. **FILING.ENUMERATION.DEPTH** is initially **T**.

Independent of **FILING.ENUMERATION.DEPTH**, a request to enumerate the top-level of a file server's hierarchy lists only the top level, i.e., assumes a depth of 1. For example, **(DIRECTORY '{PHYLEX:})** lists exactly the top-level directories of the server **PHYLEX:**.

SPP Stream Interface

This section describes the stream interface to the Sequenced Packet Protocol. SPP is the transport protocol for Courier, which in turn is the transport layer for Filing and Printing.

(SPP.OPEN *HOST SOCKET PROBEP NAME WHENCLOSEDFN***)** [Function]

This function is used to open a bidirectional SPP stream. There are two cases: user and server.

User: If *HOST* is specified, an SPP connection is initiated to *HOST*, an **NSADDRESS** or string representing an NS address. If the socket part of the address is null (zero), it is defaulted to *SOCKET*. If both *HOST* and *PROBEP* are specified, then the connection is probed for a response before returning the stream; **NIL** is returned if *HOST* doesn't respond.

Server: If *HOST* is **NIL**, a passive connection is created which listens for an incoming connection to local socket *SOCKET*.

SPP.OPEN returns the input side of the bidirectional stream; the function **SPPOUTPUTSTREAM** is used to obtain the output side. The standard stream operations **BIN, READP, EOFP** (on the input side), and **BOUT, FORCEOUTPUT** (on the output side), are defined on these streams, as is **CLOSEF**, which can be applied to either stream to close the connection.

NAME is a mnemonic name for the connection process, mainly useful for debugging. *WHENCLOSEDFN* is an optional function or list of functions to call when the stream is closed, either by the user or the server.

(SPPOUTPUTSTREAM *STREAM***)** [Function]

Applied to the input stream of an SPP connection, this function returns the corresponding output stream.

SPP.USER.TIMEOUT [Variable]

Specifies the time, in milliseconds, to wait before deciding that a host isn't responding.

(SPP.SENDEOM *STREAM***)** [Function]

Transmits the data buffered so far on this output stream, if any, with the End of Message bit set. If there is nothing buffered, sends a zero-length packet with End of Message bit set.

(SPP.DSTYPE *STREAM DSTYPE***)** [Function]

Accesses the current datastream type of the connection. If *DSTYPE* is **NIL**, returns the datastream type of the current packet being read. If *DSTYPE* is non-**NIL**, sets the datastream type of all subsequent packets sent on this connection, until the next call to SPP.DSTYPE. Since this affects the current partially-filled packet, the stream should probably be flushed (via FORCEOUTPUT) before this function is called.

(SPP.EOMP *STREAM***)** [Function]

This function returns T or **NIL** depending on whether or not an End of Message indication has been reached. This is only true after the last byte of data in the message has been read. By convention, EOFP is true of a stream that is at **EOM**.

(SPP.CLEAREOM *STREAM NOERRORFLG***)** [Function]

Clears the End of Message indication on *STREAM*. This is necessary in order to read beyond the EOM. Causes an error if the stream is not currently at the End of Message, unless NOERRORFLG is true.

(SPP.SENDATTENTION *STREAM ATTENTIONBYTE*) [Function]

Sends an SPP "attention" packet, one with the Attention bit set and containing the single byte of data *ATTENTIONBYTE*.

Courier Remote Procedure Call Protocol

Courier is the Xerox Network Systems Remote Procedure Call protocol. It uses the Sequenced Packet Protocol for reliable transport. Courier uses procedure call as a metaphor for the exchange of a request from a user process and its positive reply from a server process; exceptions or error conditions are the metaphor for a negative reply. A family of remote procedures and the errors they can raise constitute a remote program. A remote program generally represents a complete service, such as the Filing or Printing programs described earlier in this chapter.

For more detail about Courier, the reader is referred to the published specification of the Courier protocol. The following documentation assumes some familiarity with the protocol. It describes how to define a Courier program and use it to communicate with a remote system element that implements a server for that program. This section does not discuss how to construct such a server.

Defining Courier Programs

A Courier program definition is a file package type and command, **COURIERPROGRAMS**. Thus, you can use **GETDEF, PUTDEF**, and **EDITDEF** to manipulate them, or use the file package command (**COURIERPROGRAMS** name1 name2 ...) to save them. The function **COURIERPROGRAM** can be used to define a Courier program initially.

(COURIERPROGRAM *NAME* ...) [NLambda NoSpread Function]

This function is used to define Courier programs. The syntax is **(COURIERPROGRAM** *NAME (PROGRAMNUMBER VERSIONNUMBER) . DEFINITIONS*)

The tail *DEFINITIONS* is a property list where the properties are selected from **TYPES, PROCEDURES, ERRORS** and **INHERITS**; the values are lists of pairs of the form (*LABEL . DEFINITION*). These are described in more detail as follows:

The **TYPES** section lists the symbolically-defined types used to represent the arguments and results of procedures and errors in this Courier program. Each element in this section is of the form (*TYPENAME TYPEDEFINITION*), e.g., **(PRIORITY INTEGER)**. The *TYPEDEFINITION* can be a predefined type (see next section), another type defined in this **TYPES** section, or a qualified typename taken from another Courier program; these latter are written as a dotted pair (*PROGRAMNAME* . *TYPENAME*).

The **PROCEDURES** section lists the remote procedures defined by this Courier program. A procedure definition is a stylized reduction of the Courier definition syntax defined in the Courier Protocol specification:

(*PROCEDURENAME NUMBER ARGUMENTS*
    RETURNS *RESULTTYPES* **REPORTS** *ERRORNAMES*)

ARGUMENTS is a list of type names, one per argument to the remote procedure, or **NIL** if the procedure takes no arguments. *RESULTTYPES* is a list of type names, one for each value to be returned. *ERRORNAMES* is a list of names of errors that can be raised by this procedure; each such error must be listed in the program's **ERRORS** section. The atoms **RETURNS** and **REPORTS** are noise words to aid readability.

The **ERRORS** section lists the errors that can be raised by procedures in this program. An error definition is of the form

(*ERRORNAME NUMBER ARGUMENTS*),

Where *ARGUMENTS* is a list of type names, one for each argument, if any, reported by the error.

The **INHERITS** section is an optional list of other Courier programs, some of whose definitions are "inherited" by this program. More specifically, if a type, procedure or error referenced in the current program definition is not defined in this program, the system searches for a definition of it in each of the inherited programs in turn, and uses the first such definition found.

The **INHERITS** section is useful when defining variants of a given Courier program. For example, if one wanted to try out version 4 of Courier program **BAR**, and version 4 differed from version 3 of program **BAR** only in a small number of procedure or type definitions, one could define a program **NEWBAR** with an **INHERITS** section of **(BAR)** and only need to list the few changed definitions inside **NEWBAR**.

Courier Type Definitions

This section describes how the Courier types described in the Courier Protocol document are expressed in a Lisp Courier program definition, and how values of each type are represented. Each type in a Courier program's **TYPES** section must ultimately be defined in terms of one of the following "base" types, although the definition can be indirect through arbitrarily many levels. That is, a type can be defined in terms of any other type known by an extant Courier definition. The names of the base types are "global"; they need no qualification, nor do type names mentioned in the same Courier program. To refer to a type not defined in the same Courier program (or to any non-base type when there is no program context), one writes a *Qualified name*, in the form (*PROGRAM* . *TYPE*). In general, a Qualified name is legal in any place that calls for a Courier type.

Pre-defined Types

Pre-defined (atomic) types are expressed as uppercase litatoms from the following set:

BOOLEAN	Values are represented by T and NIL.
INTEGER	Values are represented as small integers in the range [-32768..32767].
CARDINAL	Values are represented as small integers in the range [0..65535].
UNSPECIFIED	Same as CARDINAL.
LONGINTEGER	Values are represented as FIXP's.
LONGCARDINAL	Same as LONGINTEGER. Note that Interlisp-D does not (currently) have a datatype that truly represents a 32-bit unsigned integer.
STRING	Values are represented as Lisp strings.

In addition, the following types not in the document have been added for convenience:

TIME	Represents a date and time in accordance with the Network Time Standard. The value is a FIXP such as returned by the function IDATE, and is encoded as a LONGCARDINAL.

NSADDRESS	Represents a network address. The value is an object of type NSADDRESS (section X.XX), and is encoded as six items of type UNSPECIFIED.
NSNAME	Represents a three-part Clearinghouse name. The value is an object of type NSNAME (section X.XX), and is encoded as three items of type STRING.
NSNAME2	Represents a two-part Clearinghouse name, i.e., a domain. The value is an object of type NSNAME (section X.XX), and is encoded as two items of type STRING.

Constructed Types

Constructed Types are composite objects made up of elements of other types. They are all expressed as a list whose **CAR** names the type and whose remaining elements give details. The following are available:

(**ENUMERATION** (*NAME INDEX*) ... (*NAME INDEX*)) Each *NAME* is an arbitrary litatom or string; the corresponding *INDEX* is its Courier encoding (a **CARDINAL**). Values of type **ENUMERATION** are represented as a *NAME* from the list of choices. For example, a value of type (**ENUMERATION (UNKNOWN 0) (RED 1) (BLUE 2)**) might be the litatom **RED**.

(**SEQUENCE** *TYPE*) A **SEQUENCE** value is represented as a list, each element being of type *TYPE*. A **SEQUENCE** of length zero is represented as **NIL**. Note that there is no maximum length for a **SEQUENCE** in the Lisp implementation of Courier.

(**ARRAY** *LENGTH TYPE*) An **ARRAY** value is represented as a list of *LENGTH* elements, each of type *TYPE*.

(**CHOICE** (*NAME INDEX TYPE*) ... (NAME INDEX TYPE)) The **CHOICE** type allows one to select among several different types at runtime; the *INDEX* is used in the encoding to distinguish the value types. A value of type **CHOICE** is represented in Lisp as a list of two elements, (*NAME VALU*E). For example, a value of type

(CHOICE (STATUS 0 (ENUMERATION (BUSY 0) (COMPLETE 1))) (MESSAGE 1 STRING)) could be **(STATUS COMPLETE)** or **(MESSAGE "Out of paper.")**.

(**RECORD** (*FIELDNAME TYPE*) ... (*FIELDNAME TYPE*)) Values of type **RECORD** are represented as lists, with one element for each field of the record. The field names are not part of the value, but are included for documentation purposes.

For programmer convenience, there are two macros that allow Courier records to be constructed and dissected in a manner similar to Lisp records. These compile into the appropriate composites of **CONS**, **CAR** and **CDR**.

(COURIER.CREATE *TYPE FIELDNAME* ← *VALUE ... FIELDNAME* ← *VALUE*) [Macro]

Creates a value of type *TYPE*, which should be a fully-qualified type name that designates a **RECORD** type, e.g., **(MAILTRANSPORT . POSTMARK)**. Each *FIELDNAME* should correspond to a field of the record, and all fields must be included. Each *VALUE* is evaluated; all other arguments are not. The assignment arrows are for readability, and are optional.

(COURIER.FETCH *TYPE FIELD OBJECT*) [Macro]

Analogous to the Record Package operator fetch. Argument *TYPE* is as with **COURIER.CREATE**; *FIELD* is the name of one of its fields. **COURIER.FETCH** extracts the indicated field from *OBJECT*. For readability, the noiseword "of" may be inserted between *FIELD* and *OBJECT*. Only the argument *OBJECT* is evaluated.

For example, if the program **CLEARINGHOUSE** has a type declaration

```
(USERDATA.VALUE (RECORD   (LAST.NAME.INDEX CARDINAL)
        (FILE.SERVICE STRING))),
```

then the expression

```
(SETQ INFO (COURIER.CREATE (CLEARINGHOUSE
        USERDATA.VALUE)
                    LAST.NAME.INDEX ← 12
                    FILE.SERVICE ← "Phylex:PARC:Xerox")
```

Would set the variable **INFO** to the list **(12 "Phylex:PARC:Xerox")**. The expression

```
(COURIER.FETCH (CLEARINGHOUSE . USERDATA.VALUE)
FILE.SERVICE of INFO)
```

would produce **"Phylex:PARC:Xerox"**.

User Extensions to the Type Language

The programmer can add new base types to the Courier language by telling the system how to read and write values of that type. The programmer chooses a name for the type, and gives the name a **COURIERDEF** property. The new name can then be used anywhere that the type names listed in the previous sections, such as **CARDINAL**, can be used. Such extensions are useful for user-defined objects, such as datatypes, that are not naturally represented by any predefined or constructed type. The **NSADDRESS** and **NSNAME** Courier types are defined by this mechanism.

COURIERDEF [Property Name]

The format of the **COURIERDEF** property is a list of up to four elements, (*READFN WRITEFN LENGTHFN WRITEREPFN*). The first two elements are required; if the latter two are omitted, the system will simulate them as needed. The elements are as follows:

READFN	This is a function of three arguments, (*STREAM PROGRAM TYPE*). The function is called by Courier when it needs to read a value of this type from *STREAM* as part of a Courier transaction. The function reads and returns the value from *STREAM*, possibly using some of the functions described in section X.XX. *PROGRAM* and *TYPE* are the name of the Courier program and the type. In the case of atomic types, *TYPE* is a litatom, and is provided for type discrimination in case the programmer has supplied a single reading function for several different types. In the case of constructed types, *TYPE* is a list, **CAR** of which is the type name.
WRITEFN	This is a function of four arguments, (*STREAM VALUE PROGRAM TYPE*). The function is called by Courier when it needs to write *VALUE* to *STREAM*. *PROGRAM* and *TYPE* are as with the reading function. The function should write *VALUE* on *STREAM*. The result returned from this function is ignored.
LENGTHFN	This function is called when Courier wants to write a value of this type in the form (**SEQUENCE UNSPECIFIED**), and then only if the *WRITEREPFN* is omitted. The function is of three arguments, (*VALUE PROGRAM TYPE*). It should return, as an integer, the number of 16-bit words that the *WRITEFN* would require to write out this value. If values of this type are all the same length, the *LENGTHFN* can be a simple integer instead of a function. See discussion of

COURIER.WRITE.SEQUENCE.UNSPECIFIED (section **X.XX**.

WRITEREPFN This function is called when Courier wants to write a value of this type in the form **(SEQUENCE UNSPECIFIED)**. The function takes the same arguments as the *WRITEFN*, but must write the value to the stream preceded by its length. If this function is omitted, Courier invokes the *LENGTHFN* to find out how long the value is, and then invokes the *WRITEFN*. If the *LENGTHFN* is omitted, Courier invokes the WRITEFN on a scratch stream to find out how long the value is.

Performing Courier Transactions

The normal use of Courier is to open a connection with a remote system element using COURIER.OPEN, perform one or more remote procedure calls using COURIER.CALL, then close the connection with CLOSEF.

(COURIER.OPEN *HOSTNAME SERVERTYPE NOERRORFLG NAME WHENCLOSEDFN*) [Function]

Opens a Courier connection to the Courier socket on *HOST*, and returns an **SPP** stream that can be passed to **COURIER.CALL**. *HOST* can be an NS address, or a symbolic Clearinghouse name in the form of a string, litatom or **NSNAME**. In the case of a symbolic name, *SERVERTYPE* specifies the Clearinghouse property under which the server's address may be found; normally, this is **NIL**, in which case the **ADDRESS.LIST** property is used. If *NOERRORFLG* is true, **NIL** is returned if a connection cannot be made, or the server supports the wrong version of Courier. The Courier connection process is named *NAME*, if specified. *WHENCLOSEDFN* is a function of one argument, the Courier stream, that will be called when the connection is closed, either by user or server.

(COURIER.CALL *STREAM PROGRAM PROCEDURE ARG1 ... ARGN NOERRORFLG*) [NoSpread Function]

This function calls the remote procedure *PROCEDURE* of the Courier program *PROGRAM*. STREAM is the stream returned by **COURIER.OPEN**. The arguments should be Lisp values appropriate for the Courier types of the corresponding formal parameters of the procedure. There must be the same number of actual and formal arguments. If the procedure call is successful, Courier returns the result(s) of the call as specified in the **RETURNS** section of the procedure definition. If there is only a single result, it is returned directly, otherwise a list of results is returned.

Procedures that take a Bulk Data argument (source or sink) are treated specially; see section **X.XX**.

If the procedure call results in an error, one of three possible courses is available. The default behavior is to cause a Lisp error. To suppress the error, an optional keyword can be appended to the argument list, as if an extra argument. This *NOERRORFLG* argument can be the atom **NOERROR**, in which case **NIL** is returned as the result of the call. If *NOERRORFLG* is **RETURNERRORS**, the result of the call is a list (**ERROR** *ERRORNAME . ERRORARGS*). If the failure was a Courier Reject, rather than Error, then *ERRORNAME* is the atom **REJECT**.

Examples:

```
(COURIERPROGRAM PERSONNEL (17 1)
    TYPES
    ((PERSON.NAME (RECORD   (FIRST.NAME STRING)
                            (MIDDLE MIDDLE.PART)
                            (LAST.NAME STRING)))
    (MIDDLE.PART (CHOICE (NAME 0 STRING)
                         (INITIAL 1 STRING)))
    (BIRTHDAY (RECORD    (YEAR CARDINAL)
                         (MONTH STRING)
                         (DAY CARDINAL))))
    PROCEDURES
    ((GETBIRTHDAY 3 (PERSON.NAME)
                    RETURNS       (BIRTHDAY)       REPORTS
(NO.SUCH.PERSON)))
    ERRORS
    ((NO.SUCH.PERSON 1))
)
```

This expression defines **PERSONNEL** to be Courier program number 17, version number 1. The example defines three types, **PERSON.NAME, MIDDLE.PART** and **BIRTHDAY**, and one procedure, **GETBIRTHDAY**, whose procedure number is 3. The following code could be used to call the remote **GETBIRTHDAY** procedure on the host with address **HOSTADDRESS**.

```
(SETQ STREAM (COURIER.OPEN HOSTADDRESS))
(PROG1 (COURIER.CALL STREAM 'PERSONNEL 'GETBIRTHDAY
    (COURIER.CREATE (PERSONNEL . BIRTHDAY)
                    FIRST.NAME ← "Eric"
                    MIDDLE ← '(INITIAL "C")
                    LAST.NAME ← "Cooper"))
    (CLOSEF STREAM))
```

COURIER.CALL in this example might return a value such as (1959 "January" 10).

Expedited Procedure Call

Some Courier servers support "Expedited Procedure Call", which is a way of performing a single Courier transaction by a Packet Exchange protocol, rather than going to the expense of setting up a full Courier connection. Expedited calls must have no bulk data arguments, and their arguments and results must each fit into a single packet.

(**COURIER.EXPEDITED.CALL** *ADDRESS SOCKET# PROGRAM PROCEDURE ARG1 ... ARGN NOERRORFLG*) [NoSpread Function]

Attempts to perform a Courier call using the Expedited Procedure Call. *ADDRESS* is the **NS** address of the remote host and *SOCKET#* is the socket on which it is known to listen for expedited calls. The remaining arguments are exactly as with **COURIER.CALL**. If the arguments to the procedure do not fit in one packet, or if there is no response to the call, or if the call returns the error **USE.COURIER** (which must be defined by exactly that name in *PROGRAM*), then the call is attempted instead by the normal, non-expedited method---a Courier connection is opened with *ADDRESS*, and **COURIER.CALL** is invoked on the arguments given.

Expanding Ring Broadcast

"Expanding Ring Broadcast" is a method of locating a server of a particular type whose address is not known in advance. The system broadcasts some sort of request packet on the directly-connected network, then on networks one hop away, then on networks two hops away, etc., until a positive response is received.

For use in locating a server for a particular Courier program, a stylized form of Expanding Ring Broadcast is defined. The request packet is essentially the call portion of an Expedited Procedure Call for some procedure defined in the program. The response packet is a Courier response, and typically contains at least the server's address as the result of the call. The designer of the protocol must, of course, specify which procedure to use in the broadcast (usually it is procedure number zero) and on what socket the server should listen for broadcasts.

START.CLEARINGHOUSE uses this procedure to locate the nearest Clearinghouse server.

(COURIER.BROADCAST.CALL *DESTSOCKET# PROGRAM PROCEDURE ARGS RESULTFN NETHINT MESSAGE*) [Function]

Performs an expanding ring broadcast for servers willing to implement *PROCEDURE* in Courier program *PROGRAM*. *DESTSOCKET#* is the socket on which such servers of this type are known to listen for broadcasts, typically the same socket on which they listen for expedited calls. *ARGS* is the argument list, if any, to the procedure (note that it is not spread, unlike with **COURIER.CALL**).

If a host responds positively, then the function *RESULTFN* is called with one argument, the Courier results of the procedure call. If *RESULTFN* returns a non-null value, the value is returned as the value of **COURIER.BROADCAST.CALL** and the search stops there; otherwise, the search for a responsive host continues. If *RESULTFN* is not supplied (or is **NIL**), then the results of the procedure call are returned directly from **COURIER.BROADCAST.CALL**; i.e., *RESULTFN* defaults to the identity function.

NETHINT, if supplied, is a net number or list of net numbers as a hint concerning which net(s) to try first before performing a pure expanding-ring broadcast. If *MESSAGE* is non-**NIL**, it is a description (string) of what the broadcast is looking for, to be printed in the prompt window to inform the user of what is happening. For example, **START.CLEARINGHOUSE** passes in the message "Clearinghouse servers" and the hint **CH.NET.HINT**.

Using Bulk Data Transfer

When a Courier program needs to transfer an arbitrary amount of information as an argument or result of a Courier procedure, the procedure is usually defined to have one argument of type "Bulk Data". The argument is a "source" if it is information transferred from caller to server (as though a procedure argument), a "sink" if it is information transferred from server to caller (as though a procedure result). These two "types" are indicated in a Courier procedure's formal argument list as **BULK.DATA.SOURCE** and **BULK.DATA.SINK**, respectively. A Courier procedure may have at most one such argument.

In a Courier call, the bulk data is transmitted in a special way, between the arguments and the results. There are two basic ways to handle this in the call. The caller can specify how the bulk data is to be interpreted (how to read or write it), or the caller can request to be given a bulk data stream as the result of the Courier call. The former is the preferred way; both are described below.

In the first method, the caller passes as the actual argument to the Courier call (i.e., in the position in the argument list occupied by **BULK.DATA.SOURCE** or **BULK.DATA.SINK**) a function to perform the transfer. Courier sets up the transaction, then calls the supplied function with one argument, a stream on which to write (if a source argument) or read (if a sink) the bulk data. If the function returns normally, the Courier transaction proceeds as usual; if it errors out, Courier sends a Bulk Data Abort to abort the transaction.

In the case of a sink argument, if the value returned from the sink function is non-**NIL**, it is returned as the result of COURIER.CALL; otherwise, the result of **COURIER.CALL** is the usual procedure result, as declared in the Courier program.

For convenience, a Bulk Data sink argument to a Courier call can be specified as a fully qualified Courier type, e.g., **(CLEARINGHOUSE . NAME)**, in which case the Bulk Data stream is read as a "stream of" that type (see **COURIER.READ.BULKDATA**, below).

The second method for handling bulk data is to pass **NIL** as the bulk data "argument" to **COURIER.CALL**. In this case, Courier sets up the call, then returns a stream that is open for OUTPUT (if a source argument) or **INPUT** (if a sink). The caller is responsible for transferring the bulk data on the stream, then closing the stream to complete the transaction. The value returned from CLOSEF is the Courier result. This method is required if the caller's control structure is open-ended in a way such that the bulk data cannot be transferred within the scope of the call to **COURIER.CALL**.

In either method, the stream on which the bulk data is transferred is a standard Interlisp stream, so **BIN**, **BOUT**, **COPYBYTES** are all appropriate.

Many Courier programs define a "Stream of <type>" as a means of transferring an arbitrary number of objects, all of the same type. Although this is typically specified formally in the printed Courier documentation as a recursive definition, the recursion is in practice unnecessary and unwieldy; instead, the following function should be used.

(COURIER.READ.BULKDATA *STREAM PROGRAM TYPE*) [Function]

Reads from *STREAM* a "Stream of *TYPE*" for Courier program *PROGRAM*, and returns a list of the objects read.

Passing (**X** . **Y**) as the bulk argument to a Courier call is thus equivalent to passing the function **(LAMBDA (STREAM) (COURIER.READ.BULKDATA STREAM X Y))**.

Courier Subfunctions for Data Transfer

The following functions are of interest to those who transfer data in Courier representations, e.g., as part of a function to implement a user-defined Courier type.

(**COURIER.READ** *STREAM PROGRAM TYPE*) [Function]

Reads from the stream *STREAM* a Courier value of type *TYPE* for program *PROGRAM*. If *TYPE* is a predefined type, then *PROGRAM* is irrelevant; otherwise, it is required in order to qualify *TYPE*.

(**COURIER.WRITE** *STREAM ITEM PROGRAM TYPE*) [Function]

Writes *ITEM* to the stream *STREAM* as a Courier value of type *TYPE* for program *PROGRAM*.

(**COURIER.READ.SEQUENCE** *STREAM PROGRAM BASETYPE*) [Function]

Reads from the stream *STREAM* a Courier value **SEQUENCE** of values of type *TYPE* for program *PROGRAM*. Equivalent to (**COURIER.READ** *STREAM PROGRAM* (**SEQUENCE** *BASETYPE*)).

(**COURIER.WRITE.SEQUENCE** *STREAM ITEM PROGRAM BASETYPE*) [Function]

Equivalent to (**COURIER.WRITE** *STREAM ITEM PROGRAM* (**SEQUENCE** *BASETYPE*)).

Some Courier programs traffic in values whose interpretation is left up to the clients of the program; the values are transferred in Courier transactions as values of type (**SEQUENCE UNSPECIFIED**). For example, the Clearinghouse program transfers the value of a database property as an uninterpreted sequence, leaving it up to the caller, who knows what type of value the particular property takes, to interpret the sequence of raw bits as some other Courier representation. The following functions are useful when dealing with such values.

(**COURIER.WRITE.REP** *VALUE PROGRAM TYPE*) [Function]

Produces a list of 16-bit integers, i.e., a value of type (**SEQUENCE UNSPECIFIED**), that represents *VALUE* when interpreted as a Courier value of type *TYPE* in *PROGRAM*. Examples:

```
(COURIER.WRITE.REP T NIL 'BOOLEAN) => (1)
(COURIER.WRITE.REP "Thing" NIL 'STRING) =>
          (5 52150Q 64556Q 63400Q)
(COURIER.WRITE.REP '(10 25) NIL '(SEQUENCE INTEGER)) =>
          (2 10 25)
```

(COURIER.READ.REP *LIST.OF.WORDS PROGRAM TYPE*) [Function]

Interprets *LIST.OF.WORDS*, a list of 16-bit integers, as a Courier object of type *TYPE* in the Courier program *PROGRAM*.

(COURIER.WRITE.SEQUENCE.UNSPECIFIED *STREAM ITEM PROGRAM TYPE*) [Function]

Writes to the stream *STREAM* in the form (**SEQUENCE UNSPECIFIED**) the object *ITEM*, whose value is really a Courier value of type *TYPE* for program *PROGRAM*. Equivalent to, but usually much more efficient than, (**COURIER.WRITE** *STREAM* (**COURIER.WRITE.REP** *ITEM PROGRAM TYPE*) NIL '(**SEQUENCE UNSPECIFIED**)).

APPENDIX E

Resources

A Resource Management Tool

Interlisp is based on the use of a storage-management system which allocates memory space for new data objects, and automatically reclaims the space when no longer in use. More generally, Interlisp manages shared "resources", such as files, semaphors for processes, etc. The protocols for allocating and freeing such resources resemble those of ordinary storage management.

Sometimes users need to explicitly manage the allocation of resources. They may desire the efficiency of explicit reclamation of certain temporary data; or it may be expensive to initialize a complex data object; or there may be an application that must not allocate new cells during some critical section of code.

The file package type RESOURCES is available to help with the definition and usage of such classes of data; the definition of a RESOURCE specifies prototype code to do the basic management operations. The filepkg command RESOURCES is used to save such definitions on files, and INITRESOURCES causes the initialization code to be output (similar to INITRECORDS).

The basic needs of resource management are (1) obtaining a data item from the Lisp memory management system and configuring it to be a totally new instance of the resource in question, (2) freeing up an instance which is no longer needed, (3) getting an instance of the resource for temporary usage [whether "fresh" or a formerly freed-up instance], and (4) setting up any prerequisite global data structures and variables. A RESOURCES definition consists of four "methods": INIT, NEW, GET, and FREE; each "method" is a form that will specialize the definition for four corresponding user-level macros INITRESOURCE, NEWRESOURCE, GETRESOURCE, and FREERESOURCE. PUTDEF is used to make a RESOURCES definition, and the four components are specified in a proplist:

```
(PUTDEF '<resourceName>
    RESOURCES
    '(NEW <new-instance-generation code>
        FREE <freeing-up code>
        GET <get-instance code>
        INIT <initialization code>))
```

In each case the <.. code> is a form that will appear as if it were the body of a substitution macro definition for the corresponding macro [see the discussion on the macros below].

A Simple Example

Suppose one has several pieces of code which use a 256-character string as a scratch string. One could simply generate a new string each time, but that would be inefficient if done repeatedly. If the user can guarantee that there are no re-entrant uses of the scratch string, then it could simply be stored in a global variable. However, if the code might be re-entrant on occasion, the program has to take precautions that two programs do not use the same scratch string at the same time. [Note: this consideration becomes very important in a multi-process environment. It is hard to guarantee that two processes won't be running the same code at the same time, without using elaborate locks.] A typical tactic would be to store the scratch string in a global variable, and set the variable to NIL whenever the string is in use (so that re-entrant usages would know to get a "new" instance). For example, assuming the global variable TEMPSTRINGBUFFER is initialized to NIL:

```
[DEFINEQ (WITHSTRING NIL
    (PROG ((BUF (OR (PROG1 TEMPSTRINGBUFFER (SETQ
        TEMPSTRINGBUFFER NIL))
    (ALLOCSTRING 256))))
    ... use the scratch string in the variable BUF ...
    (SETQ TEMPSTRINGBUFFER BUF)
    (RETURN]
```

Here, the basic elements of a "resource" usage may be seen: (1) a call (ALLOCSTRING 256) allocates fresh instances of "buffer", (2) after usage is completed the instance is returned to the "free" state, by putting it back in the globalvar TEMPSTRINGBUFFER where a subsequent call will find it, (3) the prog-binding of BUF will get an existing instance of a string buffer if there is one -- otherwise it will get a new instance which will later be available for reuse, and (4) some initialization is performed before usage of the resource (in this case, it is the setting of the global variable TEMPSTRINGBUFFER).

Given the following RESOURCES definition:

```
(PUTDEF 'STRINGBUFFER
    RESOURCES
    '(NEW (ALLOCSTRING 256)
        FREE (SETQ TEMPSTRINGBUFFER (PROG1 . ARGS))
        GET (OR (PROG1 TEMPSTRINGBUFFER (SETQ
                TEMPSTRINGBUFFER NIL))
            (NEWRESOURCE TEMPSTRINGBUFFER)))
        INIT (SETQ TEMPSTRINGBUFFER NIL)))
```

we could then redo the example above as

```
(DEFINEQ (WITHSTRING NIL
    (PROG ((BUF (GETRESOURCE STRINGBUFFER)))
        ... use the string in the variable BUF ...
        (FREERESOURCE STRINGBUFFER BUF)
        (RETURN]
```

The advantage of doing the coding this way is that the resource management part of WITHSTRING is fully contained in the expansions of GETRESOURCE and FREERESOURCE, and thus there is no confusion between what is WITHSTRING code and what is resource management code. This particuar advantage will be multiplied if there are other functions which need a "temporary" string buffer; and of course, the resultant modularity makes it much easier to contemplate minor variations on, as well as multiple clients of, the STRINGBUFFER resource.

In fact, the scenario just shown above in the WITHSTRING example is so commonly useful that an abbreviation has been added; if a RESOURCES definition is made with *only* a NEW method, then appropriate FREE, GET, and INIT methods will be inferred, along with a coordinated globalvar, to be parallel to the above definition. So the above definition could be more simply written

```
(PUTDEF 'STRINGBUFFER
    'RESOURCES
    '(NEW (ALLOCSTRING 256)))
```

and every thing would work the same.

The macro WITH-RESOURCES simplifies the common scoping case, where at the beginning of some piece of code, there are one or more GETRESOURCE calls the results of which are each bound to a lambda variable; and at the ending of that code a FREERESOURCE call is done on each instance. Since the resources are locally bound to variables with the same name as the resource itself, the definition for WITHSTRING then simplifies to

```
(DEFINEQ (WITHSTRING NIL
    (WITH-RESOURCES (STRINGBUFFER)
        ... use the string in the variable STRINGBUFFER ...)
```

Trade-offs in More Complicated Cases

This simple example presumes that the various functions which use the resource are generally not re-entrant. While an occasional re-entrant use will be handled correctly (another example of the resource will simply be created), if this were to happen too often, then many of the resource requests will create and throw away new objects, which defeats one of the major purposes of using RESOURCES. A slightly more complex GET and FREE method can be of much benefit in maintaining a pool of available resources; if the resource were defined to maintain a list of "free" instances, then the GET method could simply take one off the list and the FREE method could just push it back onto the list. In this simple example, the SETQ in the FREE method defined above would just become a "push", and the first clause of the GET method would just be (pop TEMPSTRINGBUFFER)

A word of caution: if the datatype of the resource is something very small that Interlisp system is "good" at allocating and reclaiming, then explicit user storage management will probably not do much better than the combination of cons/createcell and the garbage collector. This would especially be so if more complicated GET and FREE methods were to be used, since their overhead would be closer to that of the built-in system facilities. Finally, it must be considered whether retaining multiple instances of the resource is a net gain; if the re-entrant case is truly rare, it may be more worthwhile to retain at most one instance, and simply let the instances created by the rarely-used case be reclaimed in the normal course of garbage collection.

User-Level Macros for Accessing RESOURCES

Four user-level macros are defined for accessing RESOURCES:

(NEWRESOURCE <resourceName> . ARGS) [Macro]
(FREERESOURCE <resourceName> . ARGS) [Macro]
(GETRESOURCE <resourceName> . ARGS) [Macro]
(INITRESOURCE <resourceName> . ARGS) [Macro]

Each of these macros behave as if they were defined as a substitution macro of the form

```
((RESOURCENAME . ARGS) ...macrobody...)
```

where the expression "...macrobody..." is selected by using the "code" supplied by the corresponding method from the <resourceName> definition.

Note that it is possible to pass "arguments" to the user's resource allocation macros. For example, if the "GET" method for the resource FOO is (GETFOO . ARGS), then (GETRESOURCE FOO X Y) is transformed into (GETFOO X Y). This form was used in the "FREE" method of the STRINGBUFFER resource described above, to pass the old STRINGBUFFER object to be freed.

(WITH-RESOURCES (<resource1> <resource2> ...) <form1> <form2> ...) [Macro]

The WITH-RESOURCES macro binds lambda variables of the same name as the resources (for each of the resources <resource1>, <resource2>, ...) to the result of the GETRESOURCE macro; then executes the forms <form1>, <form2>, etc., does a FREERESOURCE on each instance, and returns the value of the last form (evaluated and saved before the FREERESOURCEs).

Note: (WITH-RESOURCES <resourceName> ...) is interpreted the same as (WITH-RESOURCES (<resourceName>) ...). Also, the singular name WITH-RESOURCE is accepted as a synonym for WITH-RESOURCES.

Saving RESOURCES in a File

RESOURCES definitions may be saved on files using the RESOURCES filepkg command. Typically, one only needs the full definition available when compiling or interpreting the code, so it is appropriate to put the filepkg command in a (DECLARE: EVAL@COMPILE DONTCOPY ...) declaration, just as one might do for a RECORDS declaration. But just as certain record declarations need *some* initialization in the run-time environment, so do most resources. This initialization is specified by the resource' INIT method, which is executed automatically when the resource is defined by the PUTDEF output by the RESOURCES command. However, if the RESOURCES command is in a DONTCOPY expression and thus is not included in the compiled file, then it is necessary to include a separate INITRESOURCES command in the filecoms to insure that the resource is properly initialized.

[This page intentionally left blank]

APPENDIX F

The Use of Fonts in Harmony

Harmony introduces some new conventions for using fonts within Interlisp-D. This is a description of such use of fonts, especially as they relate to output on the screen and product (Interpress) printers.

Display Fonts

Each display font distributed with the Harmony release has the extension .DISPLAYFONT. The variable
DISPLAYFONTEXTENSIONS
should be initially set to the list (DISPLAYFONT). Other extensions such as .AC or .STRIKE can be included in the list for backward compatibility to other display fonts.

DISPLAYFONTDIRECTORIES should be initialized to a list containing the names of directories to be searched for display fonts.

With these variables set, and given the format <family><size>{-B}{-I}-C0,where:

<family> is the font family
<size> is the point size of the font
-B is present only for Bold fonts
-I is present only for Italic fonts,
-C0 is Character Set 0

and **DISPLAYFONTEXTENSIONS** set to (DISPLAYFONT STRIKE AC),

the system will perform the search for display fonts as follows:

Search DISPLAYFONTDIRECTORIES for <family><size>
{-B}{-I}-C0.DISPLAYFONT
Search DISPLAYFONTDIRECTORIES for <family><size>
{B}{I}.STRIKE
Search DISPLAYFONTDIRECTORIES for <family><size>
{-B}{-I}-C0.AC

If bold or italic of the specified font is not found,it will be synthesized from the existing medium or regular font respectively.

Printer Fonts

In order to print a specified font on a product (Interpress) printer, a width file must be accessed for each font. These width files all have extensions of .WD and come from various character sets; character sets for widths distributed with the Harmony release include sets 0, 357 and certain others for special fonts. The names of directories to be searched for these files should be included in the list **INTERPRESSFONTDIRECTORIES**.

In order to print a specified font on a Press printer, the list **PRESSFONTWIDTHSFILES** should include the file names (including host and directory) of the FONTS.WIDTHS file.

The following is a sample procedure for loading fonts in the Harmony release of Interlisp-D.

[1] Check with the system administrator to determine what fonts are loaded on the printer.

Example: On a product printer running Services 7.0.1 software, one should **Logon** and **List Fonts**. On a product printer running Services 8.0 software, one should **Logon, Enable**, and **List Software Services**. Fonts on the printer AND printable from Lisp will be a subset of the following fonts:

Classic
Modern
Terminal
BoldPS
Titan
ClassicThin
LetterGothic

[2] Copy the all corresponding display and widths font files distributed with the Harmony release from floppies to a centralized directory. It will be useful to have the Lisp Library package COPYFILES.DCOM loaded.

Example: **(COPYFILES '{FLOPPY}CLASSIC*.DISPLAYFONT '{DSK}<LISPFILES>FONTS>DISPLAY>CLASSIC*.DISPLAYFONT)**

and **(COPYFILES '{FLOPPY}CLASSIC*.WD '{DSK}<LISPFILES>FONTS>WIDTHS>CLASSIC*.WD)**

[3] Set DISPLAYFONTEXTENSIONS to a list of all extensions used for Display fonts.

Example: **(SETQ DISPLAYFONTEXTENSIONS '(DISPLAYFONT))**

[4] Set DISPLAYFONTDIRECTORIES to a list of all directories to be searched for Display fonts.

Example: Note the directory as chosen in step [2] to be included in the list.
(SETQ DISPLAYFONTDIRECTORIES '({DSK}<LISPFILES>FONTS>DISPLAY>)

[5] Set INTERPRESSFONTDIRECTORIES to a list of all directories to be searched for font widths files.

Example: Note the directory as chosen in step [2] to be included in the list.
(SETQ INTERPRESSFONTDIRECTORIES '({DSK}<LISPFILES>FONTS>WIDTHS>)

- **CUSTOMERS WITH PRESS PRINTERS:**

[6] Copy the FONTS.WIDTHS file to a file in a centralized directory.

Example: **(COPYFILE '{FLOPPY}FONTS.WIDTHS '{DSK}<LISPFILES>FONTS.WIDTHS)**

[7] Set PRESSFONTWIDTHSFILES to a list of all files where FONTS.WIDTHS might be found.

Example: **(SETQ PRESSFONTWIDTHSFILES '({DSK}<LISPFILES>FONTS.WIDTHS))**

- **Other important font-related variables within Interlisp-D are described below.**

FONTDEFS is a dictionary in a-list form of complete "fontsets". Fontsets are used to facilitate switching back and forth between different styles of listing streams. The Harmony Lisp.sysout FONTDEFS includes three such fontsets called PARC, STANDARD and SMALL. Each of these sets contains variable settings and a **FONTPROFILE**.

A **FONTCLASS** is a record consisting of a name, three fonts (display, press, interpress) in a form acceptable to **FONTCREATE**, and a number which identifies that **FONTCLASS**. The PRETTYPRINTER uses eight **FONTCLASSes**, each of which is described on p. 6.55 of the Interlisp Reference Manual. FONTCLASSes are also used by the function CHANGEFONT (p. 6.57) and the PRINTOUT command .FONT (p. 6.27).

A **FONTPROFILE** is a list of lists, each of which consists of a FONTCLASSNAME, PRETTYFONT#, DISPLAYFONT, PRESSFONT and INTERPRESSFONT. Where the second element of the sub-list is a litatom (as opposed to PRETTYFONT#) the first FONTCLASSNAME becomes an alias for the second. Note that the second FONTCLASSNAME must already have been defined.

Because each **FONTCLASS** is in fact a record of font information, functions such as FONTCOPY and FONTCREATE can be used to create new fonts easily by using fields of any given **FONTCLASS** as parameters. For example, given the following **FONTCLASS** with:

FONTCLASSNAME	BIGFONT
PRETTYFONT#	4
DISPLAYFD	(HELVETICA 12 BRR)
PRESSFD	(HELVETICA 10 BRR)
INTERPRESSFD	(MODERN 10 BRR)

One could specify a new italic interpress font named NEWFONT from the **FONTCLASS** BIGFONT by saying:

```
(SETQ NEWFONT (FONTCOPY BIGFONT 'SLOPE 'ITALIC 'DEVICE
'INTERPRESS))
```

The result would be a fontdescriptor of the same font name [MODERN], size [10], weight [BOLD] and expansion [REGULAR] as specified for the INTERPRESSFD of the **FONTCLASS** BIGFONT, but with a different slope [ITALIC].

APPENDIX G

Known Problems in this Release

- If Harmony and Carol are installed on the same machine, the local file system will become unusable by either.

- **AR** (Action Request) 2726 - shift selecting from **DEDIT** will smash **TAB** key in **TEDIT**;

- AR 869 - **DEDIT** from inspector smashes **TAB**. The recovery action is **KEYACTION(TAB ((9 9 NOLOCKSHIFT) . IGNORE)**.

- If users have created display fonts in Carol that they wish to use in Harmony, the upgrade procedure is the following: (1) While running the Carol release, load the lispusers package **EDITFONT.DCOM**. (2) Use the function (**WRITESTRIKEFONTFILE** <fontdescriptor> <filename>) to save each font descriptor as a "strike" format file. Note that strike files do not contain information about the font family, size, etc, so give the strike files descriptive names: e.g., **GREEK10B.STRIKE**. (3) While running the Harmony release, load **EDITFONT.DCOM** (note: the same package has been tested to work with both Carol and Harmony). (4) Use the function (**READSTRIKEFONTFILE** <family> <size> <face> <file>) to read in the strike file, and create a fontdescriptor with the specified family name, face, etc.

- If **COPYRIGHTFLG** is **NIL** (the default when the lisp.sysout is installed), the system will still put out a message "(* Copyright (c) by NIL. All rights reserved.)" A simple workaround is **ADVISE(PRINTCOPYRIGHT1 (OR OWNER (RETURN]**