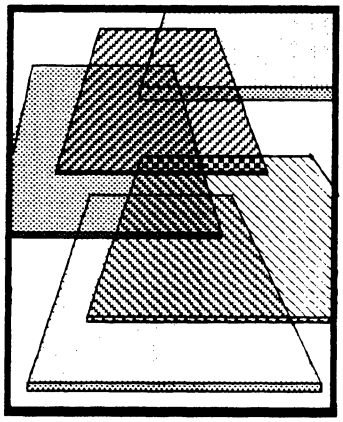
XEROX

Xerox Development Environment



XDE Unsupported Software Description

XDE3.0-9001
Version 3.0
November 1984

Office Systems Division
Xerox Corporation
3450 Hillview Avenue
Palo Alto, California 94304

Notice

This manual is the current release of the Xerox Development Environment (XDE) and may be revised by Xerox without notice. No representations or warranties of any kind are made relative to this manual and use thereof, including implied warranties of merchantability and fitness for a particular purpose or that any utilization thereof will be free from the proprietary rights of a third party. Xerox does not assume any responsibility or liability for any errors or inaccuracies that may be contained in the manual or have any liabilities or obligations for any damages, including but not limited to special, indirect or consequential damages, arising out of or in connection with the use of this manual or products or programs developed from its use. No part of this manual, either in whole or part, may be reproduced or transmitted mechanically or electronically without the written permission of Xerox Corporation.

Preliminary.

-- Activity.doc

Activity is an outgrowth of CpuMonitor and StorageFaultMonitor. Using a very small window, it displays two bar graphs that indicate machine activity. One graph shows the percentage of cpu utilization. The other graph shows a count of either disk IO operations or page faults.

The graphs are updated once per second and give the instantaneous activity (for that second) in black, and an average activity over the past 10 seconds in gray. The cpu utilization percentage and IO activity for the second are also displayed numerically.

You can use a menu to select whether to display all disk IO operations or just page faults. Initially, all disk IO operations are shown. Nearly all page faults result in a disk operation. However, the disk will also be touched by explicit calls on such Pilot operations as Space.ForceOut, File.Create, File.Delete, File.SetSize, and File.GetAttributes.

Activity's default window is 30 pixels high by 184 pixels long and just overlays part of the Herald window. Its window box can also be set from User.cm.

The User.cm entry can also contain a section telling Activity whether to default to showing all DiskIO (default if nothing specified), or faults.

Sample:

```
[Activity]
WindowBox: [x: 512, y: 0, w: 184, h: 30]
IO: Faults
```

– – Adder.doc

Adder is a simple adding machine that may be useful alone or in conjunction with SimpleCalc. Its main feature is that it will extract signed numbers out of text, including monetary amounts like $123.45 or –$12,345.

Adder keeps a single – register running total (a LONG INTEGER), which can be reset to 0 by the "Clear!" command.

The "Add!" command adds this running total together with whatever numbers are found in an input string, and displays the new running total. The input string is "String:" if this is non – empty, else the current selection. In parsing the input string, every character except a digit, period, or comma is a number separator Whatever the intervening text may be, all of the separate numbers are ADDED together, except that an odd number of minus signs " – " makes the succeeding number be subtracted. The "Sub!" command is the same as "Add!" except that it flips the sign of each number it finds in the input string.

(If you do degenerate things, then what you see highlighted may not be what you get as the selection text, e.g. if you type in text after the selection before using the Adder.)

The interpretation of the input digits is controlled by the setting of "Interpret Input String As:", and not by trailing "B"s or other indicators. If this setting is "Money", then the running total is kept in pennies and shown as dollars – and – cents; otherwise it is displayed as an integer in both decimal and octal.

Simple Multiply and Divide commands are provided, using the integer specified after "... by" as the second argument. You can get a buffer of partial results by stretching the height of the upper sub – window. There are no keyboard commands.

- - AddHintMenus.doc

AddHintMenus allows you to add a menu with a list of "hints" for the contents of FormSW string fields. For instance, if you're tired of typing certain host and directory names in the FileTool, you can now have a menu of directory names attached to the "Host" and "Directory:" fields of the FileTool.

The AddHintMenus.~ command reads hints from a file. If no file name is specified on the command line, it reads them from User.cm. The hints are specified in the [Hints] section of the file, in the following format:

[Hints]
ToolName: FieldName FirstValue "Value With Spaces" ThirdValue

where

"ToolName" is the name of a tool,
"FieldName" is the name of a string field in some FormSW of that tool,
and the remainder of the line is a list of items that should make up the menu.

Note that the second item is quoted since it contains white - space characters. (If the field name contains white - space characters, it too must be quoted.)

When you want to change the hints for a particular field, just edit user.cm (or whatever file you store your hints in) and run AddHintMenus.~ again.

AddHintMenus provides expansion of hints. Expansion is done using the Expand interface provided in Tajo. If automatic local expansion is not desired the user can single - quote (') the asterisk (*). Because the Expand interface is used one must work around comments (- -) by inserting a single - quote in front of what would normally be considered a comment delimiter according to the Expand interface.

Note: It may be desireable to place the hints in the User.cm for two reasons. The first reason is that simply typing "AddHintMenus.~" results in the User.cm file being the default. The second reason is that AddHintMenus will automatically be called on activation of a tool (only on the particular tool being activated); and it will parse the hints from the User.cm file.

The following is a sample User.cm "Hints" section:

[Hints]
FileTool: Host RemoteServerName1 RemoteServerName2 RemoteServerName3
FileTool: Directory <Subdirectory>tools <Subdirectory>Fonts <Subdirectory>Private FileTool: LocalDir
<>Filing <>Tools <>Temp <>Symbols
FileTool: Source '*.mesa '*.bcd '*.config
MailSend: File *.msg *.MailForm

-- AlarmClock.doc

AlarmClock is a tool that locks up the notifier until a designated time has passed. This is useful for controlling the time that tests or tools are started. The tool has a form subwindow containing a string field named "Time" and a command named "Wait", the use of which should be obvious. Pushing the Stop button while the curser is within the tool's window will abort the wait. Tool activity is logged in a file subwindow.

Times may have these formats:

An empty time string is interpreted as the current time.

28-May-82 13:15:11 EST	
28-May-82 13:15:11	
28-May-82 13:15	(same as 28-May-82 13:15:00)
13:15:11	(the next time it is 13:15:11 in the local time zone)
13:15	(the next time it is 13:15:00 in the local time zone)
1200	(wait for 1200 seconds)

- - Ascii860ConversionTool.doc

Ascii860ConversionTool is used to convert files which are in the 860 format into plain ascii text files, and vice versa. The tool has the following fields:

direction: An enumerated type used to specify what type of a conversion you wish to make. When AsciiFrom860 is selected, the file specified in the Source File Name field will be used as input, converted into ascii text, and written out to the file specified in the Destination File Name field. When AsciiTo860 is selected, the file specified in the Source File Name field is used and input and convered to 860 format, which is written to the file specified as in the 860Filename field.

Source File Name: The local input file.

Destination File Name: The local output file.

Overwrite Switch: Allows an existing output file to be overwritten with the text from the conversion. The default value is "No Overwrite".

Convert!: Execute the conversion.

Ascii860ConversionTool requires a file name in both the 860Filename field, and the AsciiFilename field. It will inform you if you leave out a file name, give a bad file name, or try to overwrite an existing output file, when the Overwrite Allowed boolean is not TRUE (i.e. is not selected).

You can use Ascii860ConversionTool for converting Star documents into ascii text file. To do this, while running Star, first select the document you are converting. Move the pointer into the desktop auxiliary menu, which is the small box with three horozontal lines, located in the upper right corner. Hold down the left mouse button. Select Convert. Your document will be converted and will be named 860<filename>, where <filename> was the name of the file which you converted. Now either (1) store the 860 document on your filedrawer, SH - SH - STOP to CoPilot, fetch the 860 document from your filedrawer with FileTool; or (2) go to CoPilot or Tajo and use StarFileTool to copy your converted document from your Star desktop. The first method is preferred because it doesn't require any rebooting. Now use Ascii860ConversionTool to convert that file into ascii text.

NOTE: When using StarFileTool to copy the 860<filename> to either CoPilot or Tajo, StarFileTool may not be able to find the file. You can avoid this problem if you rename the <filename> you converted to <newfilename>, and then use StarFileTool to copy over the original <filename>.

- - AutoRepeat.doc

Description:
AutoRepeat repeats input character automatically in the text subwindow and form subwindow. If you press any printable key and hold it down, the character corresponding the key will be automatically inserted into the window.

Installation:
TIP file AutoRepeat.TIP into your TIP> directory.

Run:
Type from Executive
AutoRepeat firstNumber secondNumber<CR>
where the firstNumber is the number of milliseconds before the first repeat. the secondNumber is the number of milliseconds between two repeats. The default value of firstNumber is 500 and secondNumber is 100

-- Background.doc

Background.bcd provides the facility formerly provided by the Background line in the [System] section of User.cm. That is, it lets you specify one or more Exec commands that are to be performed the first time you return to Copilot after booting. Actually, the commands are performed the first time you return to Copilot after running Background.bcd, but if you run it as part of your InitialCommand you get the desired effect.

The commands to be performed are found by looking in User.cm for a line called "Command:" within the section labelled [Background]. Unlike the Trinity/Cascade "Background" feature, which required a list of programs to be run, Background.bcd takes a full Exec command line, just like the InitialCommand in the [System] section. The command(s) must be on a single line, so use semicolons instead of CRs between commands.

The commands are executed at background priority. Output is sent to the "indirect output" sink, initially the Herald (see IndirectOutput.doc). The background process can be aborted by issuing a global abort (STOP – STOP).

WARNING! Background.bcd has the same drawback the old [System] Background feature had: You can't Proceed from Copilot until the background stuff is finished. (This is because you might crash if you were to Proceed while the background process was writing on the disk.) Background.bcd therefore vetos any world – swaps until it is finished. If you find this frustrating, you should endeavor to do as little as possible in the background.

A typical use of Background would be:

[Background]
Command: Run FindSource.bcd Flash.bcd

FindSource can't be run via InitialCommand because its changes to the File Window menu are overridden later during booting, after the InitialCommand is finished. (At least, that was true in Sierra.) By running it with Background, you can make sure its changes take place after everything else has settled down. Flash.bcd (see Flash.doc) lets you know that the background command has finished.

If you boot with the 'W switch, Copilot does not boot the client volume at the end of initialisation. In this case, the background process is started when the Copilot log window is created, which is pretty much the end of the Copilot boot sequence.

- - Basic.doc

Basic is a hack that has been adapted from Laurel Basic. It is run in the Executive. There is, unfortunately, not much documentation on this dialect of basic. The keyword lists given below are all there is. If you do not speak basic already, this may not be enough.

Known Bugs:
- Renumber Command - does not work
- Print USING - does not work
- IMAGE - does not word (used with Print USING)
- The support for strings is very weak - No concatenation or substring capability

Variable Names:

Numeric Variables	One letter or one letter and one digit
String Variables	One letter and a dollar sign ($) or one letter, one digit and a dollar sign.

Key Words:

AUTO n,m	Turn on auto line numbers n = starting number (default is 10) m = increment (default is 10)
CLEAR	Clear the screen? (outputs one blank line)
CONT	Continue a program (only after a pause)
CODELIST n,m	List the source and the internal op codes n = first line to list (default is first line) m = last line to list (default is last line)
DATA	Provide data for a READ statement
DEF func	Start of a function definition
DEG	Use degrees in trigonometric functions
DEL n,m	Delete program lines starting with line number n and ending with line number m
DIM v(n,m)	Dimension variable v to be a n by m array (m is optional)
DISP	See Print
END	End of a program (stop the program)
FNEND	End of a function definition
FOR	Start of a FOR - NEXT loop (FOR I = 1 TO 10 STEP 1 : NEXT I)
GOSUB n	Enter a subroutine starting a line n
GOTO n	Goto line n
HELP	Print the keyword list
IF	If statement (IF condition statement) condition - A > = B or A$ = B$ etc. statement - only LET, GOTO (or THEN) and GOSUB are legal
IMAGE	Print format for use with PRINT USING
INPUT a,b	Input from the user (keyboard) date into the variables a and b.
LET	Assignment statement (LET A = 5) ("LET" is optional)
LIST n,m	List the source n = first line to list (default first line) m = last line to list (default last line)
LOAD file	Load the source file into basic
NEW	Clear the work space (erase the existing program)
NEXT	End of the FOR - NEXT loop (FOR I = 1 TO 10 STEP 1 : NEXT I)
NORMAL	Turn off auto line numbering
ON v GOTO or GOSUB l1, l2, l3 ...	Computed goto or gosub based on v
OPTIONBASE 0	Index arrays staring with 0 or 1
PAUSE	Temporary stop in the execution of a program (CONT will cause it to start again)
PRINT	Print the list of expressions, variables, or quoted strings. A comma (,) will cause a tab to separate each item output and a semicolon (;) will cause no separation between them.
QUIT	Exit basic (program lost if not already saved)
RAD	Use radians in trigonometric functions
READ	Initialize variables using the data in a DATA statement
REM	Remark (comment line) (can also use !)
RENUMBER	Renumber the program lines n = new starting number (default is 10) m = increment (default is 10) o = old line number to start with (default is first line) p = old line number to end with (default is last line)

RESTORE ?
RETURN Return from a subroutine
RUN n Start a program executing
n = starting line number (default is first line)
SAVE file Save the current source into file
STEP n Optionally used in a FOR statement (for i = 0 to 5 STEP 1)
STOP Stop the program
TAB ?
TO n Used in a FOR statement (for i = 0 TO 5 step 1)
THEN n Used in an IF statement (if a = b THEN 100) same as (if a = b GOTO 100)
USING n Used in PRINT to supply format information

Functions:

ABS(x) Absolute value of x
ACS(x) ArcCosine of x
ASN(x) ArcSine of x
ATN(x) ArcTangent of x
ATN2(x,y) ArcTangent of y/x
CEIL(x) Smallest integer > = x
CHR$(x) Character representation of x
COS(x) Cosine of x
COT(x) Cotangent of x
CSC(x) Cosecant of x
DTR(x) Degrees to radians for x
EPS The constant epsilon (0.0099)
EXP(x) The constant e raised to the x power
FLOOR(x) Largest integer < = x
FP(x) Fractional part of x
IP(x) Integer part of x
INT(x) Largest integer < = x
LEN(s$) Length of string s$
LGT(x) Log (base 10) of x
LOG(x) Log (base e) of x
MAX(x,y) The larger value of x or y
MIN(x,y) The smaller value of x or y
NUM(s$) String to numeric conversion of s$
PI The constant pi (3.14159265359)
POS(s$,t$) The position of string t$ in string s$
RMD(x,y) The fractional remainder from x/y (FP(x/y))
RND(x) A uniform random number > = 0 and < x [0..x)
RTD(x) Radians to degrees for x
SEC(x) Secant of x
SGN(x) Sign of x
x<0 = > -1
x=0 = > 0
x>0 = > 1
SIN(x) Sine of x
SQR(x) Square root of x
TAN(x) Tangent of x
UPC$(s$) Conversion to upper case of string s$
VAL(s$) String to numeric conversion of s$
VAL$(x) Numeric to string conversion of x

-- BcdType.doc

BcdType allows you to separate a list of bcds into six lists: BAD BCDS, DEFINITIONS, MODULES, CONFIGURATIONS, PACKAGED, and TABLE COMPILED.

To use, type into the exec: BcdType file1 file2 ... filen

A sample script in the Executive:

```
>BcdType *.bcd
BAD BCDS: a.bcd, bbcd

DEFINITIONS: String.bcd, TextSW.bcd

MODULES: StringImplA.bcd, TextSWsA.bcd

CONFIGURATIONS: TextSWs.bcd

PACKAGED: Compiler.bcd

TABLE COMPILED: Rats.bcd
>
```

-- BinCom.doc.

Environments: CoPilot | Tajo

Description: BinCom compares two files and displays the differences to the exec log or to an optional output file. Output is in hex if h switch, decimal if d switch and octal otherwise.

Sample Commands:

```
>BinCom Old.bcd New.bcd
>BinCom MuchOutput.txt/o /h Old.boot New.boot
```

- - BootVolumeName.doc

Running BootVolumeName.bcd registers the Executive command, BootVolumeName.~:

This hack enables users to boot from Executive command lines. The volume name need not be complete (nor capitalized perfectly) but it should obviously be unambiguous.

Boot - switches are optional immediately after the volume name. If absent, the current switches are used (e.g. see the Herald Window menu's Set Switches command). If an EMPTY switch setting is specified, no switches are used (other than the system defaults).

Normally, BootVolumeName requests confirmation from the user before booting. Auto - confirming can be specified by adding a Y or y to the command line. (The letter must be preceded by at least one blank; otherwise it may look like a switch setting.)

Some examples:

```
BootVolumeName.~ System     - - boots System using current switches
BootVolumeName.~ sys/dw{    - - boots System with switches: dw{
BootVolumeName.~ CoPilot/   - - boots CoPilot using no switches
BootVolumeName.~ CoPilot/ Y - - ditto, with auto - confirmation
```

-- BringOver.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- BrushDMT.doc -- edited by:

This program reads brush files generated by Doodle and displays them as DMT objects floating around the screen. There are three User.cm entries processed at every activation:

```
[BrushDMT]
BrushFile: <brush>/<switches>          -- name of file containing a brush
Delay: 1000                            -- jump delay in msecs
SmashPassword: TRUE|FALSE
```

You can specify up to ten brushes, in multiple "BrushFile" lines or all on one line, as you prefer. If no BrushFiles are specified, or the specified brushes cannot be opened, the program will look for DMT.brush. If that cannot be found, then the program degenerates into vanilla DMT. You can also specify your brushes in the Exec with the command:

```
BrushDMT <brush>/<switches> ...
```

If the ExecProc is used but the command line is empty, or the specified brushes cannot be opened, BrushDMT will look in User.cm just like for a plain activation. The switches, both in User.cm and in the Exec, are as follows:

```
/g  --  glide around the screen smoothly (well, semi - smoothly)
/j  --  jump around the screen randomly - this is the default
/r  --  jump around the screen at regular intervals
/t  --  paint a vanilla - DMT - style time box at the "active point"
```

You can combine the switches -- /gt for gliding with the time box on, /jg for both gliding and jumping, etc.. You can also set "global" switches. Examples:

```
BrushDMT YodaDMT
BrushDMT /g FlyingV/t Sphere Sphere Sphere Sphere
```

There's a collection of brushes on diskettes.

-- ButtonsAndLights.doc

ButtonsAndLights is a module that throws 10 to 40 small buttons, small lights and 3 inch (200 bit) bar charts on the screen, superimposed over the Tajo or Star display. Any of the lights or bar charts may be set at any time. If the mouse is moved into a button area, a client proc is called, if it's been registered. The client, in turn, may or may not set one of the lights on or off, in addition to affecting his program's behavior. The module is intended for modifying and monitoring program behavior without going to the debugger, and is handy for watching program speed change as the behavior is modified. Timings of the client program may also be AUTOMATICALLY displayed on the bar charts.

ButtonsAndLights consists of a defs module and a program module, and can be loaded into Tajo or Star without change. It writes directly on the screen bitmap, ignoring whatever window package Star or Tajo is using. In case the screen is moving "underneath", a call is available to repaint the entire view area; repaint also happens when any button is hit.

The client may implement and register the CatchAllButtons proc of ButtonsAndLightsDefs in one or more of his modules.
For example, if button 3 is moved over (no mouse buttoning is really necessary), CatchAllButtons[3] will be called.

The client can do anything he wants upon catching a button call. Typically, he'll light an appropriate light and set a boolean in his own program. He sets a light by calling ButtonsAndLightsDefs.SetLight[lightNum: LightNum, color: BOOLEAN].

Shape of the buttons and lights is such that they can be easily read against a background of black, white or gray. The active area is in the upper right corner of the screen, two inches from the top.

BarCharts.
Bar charts are initially invisible. Upon the first call to ButtonsAndLightsDefs.SetBar[barNum: BarNum, value: LONG CARDINAL], the buttons and lights move down on the screen and the bar charts appear above them. Cost of a call to SetBar is 30 to 1000 usec, with more detail in the defs file.

Automatic Scaling.
If the value sent to a bar is greater than 100, it is automatically divided by the proper scale of ten, and the number of zeros to be added to the reading is shown below the bar graph. Automatic scaling may be defeated by FreezeScale[];

Automatic timing of programs.
The calls StartTiming:PROC[barNum: BarNum] and EndTiming:PROC[barNum: BarNum]RETURNS[usec: Usec] placed around any operation will time (and automatically scale) that operation to within the nearest 28 microseconds. Max value is 4000 seconds. The return values of EndTiming may be tallied or whatever, and later shown on the same or another bar.

This tool has proven to be very valuable in timing and tuning up a large set of multi – process programs.

It can be loaded or bound, in Star or Tajo, and a typical YConfig for loading with Star, is included in Hacks.

-- Camera.doc

Camera.bcd is a hack Ive seen running on SmallTalk and Interlisp machines. When you run it, a white window will appear. If you move the cursor over the window and hit Point, the cursor will disappear and rectangle will track the cursor. This rectangle will become a camera, and the window is the viewing screen. As you move the camera around your display, the image inside the rectangle is drawn within the screen window. To quit hit either Abort or Adjust on the mouse.

- - Catalog.doc

Catalog allows a user to display file attributes in the executive. The following attributes can be displayed using the following switches:

Switch	Attribute
b	Bytes
c	Create date
f	File ID
m	Remote file name
p	Pages
r	Read date
t	File Type
w	Write date

To use Catalog, type into the Exec:

>Catalog.~ /switches file1 file2 ... /switches fileN fileN + 1 ..

Examples:

```
>Catalog.~ /bp ET.bcd TestParse.bcd

Bytes   Pages  File
15872   32     <Tajo>ET.bcd
12288   25     <Tajo>TestParse.bcd

>Catalog.~ /rwc <Tajo>*.log

Create          Write           Read            File
20-Apr-83 15:40:29 20-Apr-83 15:40:29 20-Apr-83 15:40:29 <Tajo>FileTool.log
20-Apr-83 15:40:36 20-Apr-83 15:40:36 20-Apr-83 15:40:36 <Tajo>Initial.Log
20-Apr-83 15:40:25 20-Apr-83 15:40:25 20-Apr-83 15:40:25 <Tajo>SimpleExec.Log
19-Apr-83 11:30:27 19-Apr-83 11:30:27 19-Apr-83 11:30:27 <Tajo>TestParse.log

>Catalog.~ /bwc <Tajo>*.cm /rp <Tajo>Junk>*.data

Bytes   Create          Write           File
9044    19-Apr-83 15:55:09 19-Apr-83 15:55:11 <Tajo>User.cm

Pages  Read            File
5      19-Apr-83 12:05:22 <Tajo>Junk>adm3a.data
5      19-Apr-83 13:20:04 <Tajo>Junk>bitgraph.data
5      19-Apr-83 12:05:32 <Tajo>Junk>h1500.data
5      19-Apr-83 12:05:24 <Tajo>Junk>hp.data
5      18-Apr-83 16:20:50 <Tajo>Junk>vt00.data
5      19-Apr-83 13:55:38 <Tajo>Junk>vt100.data


>Catalog.~ /m <Tajo>Temp>*

File                    RemoteFileName
<Tajo>Temp>FontMonster.bcd      [Server]<Directory>FontMonster.bcd!1
<Tajo>Temp>FontMonster.mesa     [Server]<Directory>FontMonster.mesa!1
<Tajo>Temp>KeyJump.bcd          [Server]<Directory>KeyJump.bcd!2
<Tajo>Temp>KeyJump.doc          [Server]<Directory>KeyJump.doc!1
<Tajo>Temp>KeyJump.mesa         [Server]<Directory>KeyJump.mesa!2
```

-- Chimes.doc

Chimes makes your terminal into a chiming clock. The Westminster chimes play on the quarter hour. this is a nice addition to the clock intended for display on the rhine.press background picture or, if you choose, as a stand alone.

Just run Chimes.bcd which creates the command "Chimes.~". To turn it off, unload Chimes.

-- ClockTool.doc

ClockTool is a window that contains an analog clock. The size of the clock scales along with the window. It requires the module DisplayImpl.bcd to be loaded. (If you don't wish to explicitly get DisplayImpl, the tool ClockToolWithDisplayImpl has DisplayImpl bound in). ClockTool recognizes the standard stuff in your user.cm.

The clock will run in any time zone and under most major daylight savings time rules. The zone and rules can be changed either by User.cm entries (listed below) or by a property sheet. To invoke the property sheet, push "stuff" (or "open") with the cursor over the clock. The fields are:

ShowName: {true, false} display a name banner, defaults FALSE
Name: the name of the banner, defaults to "Local Time"
MinForSecs: minimum radius of dial to display a seconds hand, defaults to 28
Direction: direction zone is from GMT
Hours: hours zone differs from UT
Minutes: minutes zone differs from UT
FirstDST: largest possible JD for first day of Daylight Savings
LastDST: largest possible JD for last day of Daylight Savings

The time zone and DST parameters default to the local time parameters of your machine.

The property sheet recognizes the standard stuff in your user.cm under the entry "Clock Tool Properties". The property sheet also has two menus containing some time zones and daylight savings rules.

Glossary:

JD: Julian Day, or the number of the day in the year. For example, February 5 is JD 36.

UT: Universal Time, a.k.a. Greenwich Mean Time

For an explanation of local time parameters (e.g., the meaning of FirstDST and LastDST), please consult the Pilot Programmer's Manual.

- - CompareDir.doc

CompareDir compares files on a working directory against those on a snapshot or integration directory. When the Working Host is left blank and a Working directory is specified CompareDir compares the directories on the local disk to the snapshot directory. If the Working Host is left blank, and an asterisk is placed in Working Directory, all of the local directories will be compared to the snapshot directory. CompareDir can help you find:

- o files on the working directory that you forgot to snapshot
- o obsolete files on the working directory that can be deleted
- o files that have the wrong version on one of the directories.

The tool runs in Tajo or CoPilot, and brings up a tool window. It forks a process to do the comparison (so the notifier isn't tied up) and runs at background priority. A log is produced in CompareDir.log.

There are seven boolean items featured in CompareDir: Same, NotInSnapshot, NewerInSnapshot, NotInWorking, NewerInWorking, CanOverWrite, and UseSearchPath. Five of them, Same, NotInSnapshot, NewerInSnapshot, NotInWorking, and NewerInWorking, when true, give information as to which files are the "Same" in both directories, which files can be found in the working directory, but "NotInSnapshot" directory, etc. Also, when true, these five boolean items create separate command files of the same names (e.g. Same.cm, NewerInWorking.cm). These command files contain the names of the corresponding files. "CanOverWrite" is a switch which allows you to overwrite the command files when you are making a new comparision. It is mainly a reminder that the previous command files will be overwritten each time a comparison is made. "UseSearchPath" allows you access to the entire search path and is used with a blank working host and working directory. Every time a new directory on the search path is being enumerated the directory name is displayed.

- - CPMConversionTool.doc

This tool performs the following file conversions:

1. WordStar file to Bravo file.
2. Plain CPM text file to plain Tajo text file.
3. Plain Tajo text file to plain CPM text file.
4. (later: Bravo file to WordStar file).

In order to be converted, a CPM or WordStar file must first be moved to your Dandelion disk. This is done with a tool,

CPMFloppytool.bcd
CPMFloppytool.doc

Why was the Bravo format chosen? Because it was much easier than the Star format and there exists a Bravo - to - Star conversion program.

Following are the details of how characters from one file are copied to the other file. Basically, it "does the right thing". That is to say, letters, digits, punctuation and other symbols are carried to the new file as they should be. Other characters, generally not visible on the 820 screen but having to do with formatting (such as carriage return, tab, line feed) require some special handling. The format of the converted file will probably not look exactly like the original one, except in the case of CPM to Tajo (assuming that the Tajo window as been made at least as wide as the longest line in the original file).

1. WordStar to Bravo:

Carriage Return makes a Bravo paragraph.
Bravo paragraph margins are approximately equal to the margins in
the last line of the WordStar paragraph.
Paragraph justification is always off.
12 - pitch font becomes TimesRoman10. 10 - pitch font becomes TimesRoman8.
Bold, underlined, subscripted and superscripted properties
are carried over.

2. CPM to Tajo:

Codes 0 to 11 and 13 to 177 are copied as is.
New Line (code 12) is not copied.
Codes 200 to 211 are copied as 0 to 11 (high bit dropped).
New Line (code 12) is not copied.
Codes 213 to 377 are copied as 13 to 177 (high bit dropped).

3. Tajo to CPM:

Codes 0 to 7 are dropped.
Codes 10 to 14 are copied as is.
Carriage Return (code 15) becomes Carriage Return followed by New Line.
Codes 16 to 37 are dropped.
Codes 40 to 177 are copied as is.
Codes 200 to 377 are replaced by a question mark.
A long string of ControlZ (code 32) is appended at the end of the file
to prevent truncation when the file will be moved to floppy disk.

4. Bravo to WordStar:

Not available.
Use Strip to eliminate Bravo code then Tajo to CPM conversion.
Resulting file still acceptable to WordStar but all formatting is gone.

- - CPMFloppyTool.Doc

**I. Overview

The CPMFloppyTool runs under Tajo or CoPilot and allows the user to read or write files to/from a CP/M formatted 8" floppy disks from/to the Mesa Development Environment. The floppy disk must be formatted as a standard IBM 3740 soft sectored floppy and hence readable by the D* (Dolphin or Dandelion) series hardware.

The tool handles single/double sided or single/double density formatted floppies and correctly reads/write Xerox 820 single density, single sided (SDSS) floppies. Due to the lack of access to an Xerox 820 - II I have yet to test all the double density combinations. My default doubleDensity parameters are 128 bytes/sector and 52 sectors/track, 6 sector interleave and 1024 bytes/allocation unit. If these are not adequate then the tool allows different values to be supplied. I envision no difficulties but wider user experience may prove otherwise.

When double density floppies are desired to be read/written caution should be noted. Presently, to my knowledge no widely accepted standard exists between the various OEM vendors using CP/M as to the sector interleave value to used for anything beyond beyond single - density single - sided (SDSS). The sector interleave value(s) used are supplied by the person performing the sysgen of the CP/M BIOS. Since these values are used in the mapping of logical to physical sector numbers incorrect values will not produce the correct results. In the interest of retaining a standard set of values that covered both the SDSS case, I chose to support the interleave values of 1, 3 or 6 sectors. These values can be chosen from a choice menu, in the window that appears from selecting the "cpmOther" option of the Format parameter. The appearance of the term "sector Interlace" should be taken as equivalent to the more common term of "sector interleave" in this tool and documentation so subsequently they will be used interchangeably.

For the CP/M hacker interested in more details, Section V. Implementation Details should be consulted.

Problems, bugs or suggested enhancements are welcome via Laurel/Hardy messages to either Bill Fisher.ES or Rex Walden.ES.

**II. Parameter Subwindow paramters

Sides: A choice menu item that allows either "single" or "double" sided floppy media to be chosen.

Density: A choice menu item that allows either "single" or "double" density floppy media to be chosen.

Format: A choice menu item that allows either "cpm" or "cpmOpt" or "cpmOther" to be chosen.

When "cpm" is chosen the following parameters are used:

1) If single density = > 128 Bytes/Sector and 26 physical sectors/track
   If double density = > 256 bytes/sector and 26 physical sectors/track
2) 6 sectors interlace is used.
3) 77 tracks/side
4) 1024 bytes/allocation unit.

These are the standard values used in the single and doulbe density CP/M floppy cases.

When "cpmOpt" is chosen the parameters are the same as in the "cpm" case except that a 3 sector or "optimized" sector interlace value is used:

When "cpmOther" is chosen a parameter subwindow is created that allows integer values to be supplied for the following parameters used in reading the media; a) bytesPerSector
b) tracksPerSide
c) physicalSectorsPerTrack
d) allocationSizeInBytes.
e) In addition the sector Interlace value can be chosen from a menu to be either 1, 3 or 6 sectors.

Source:

The list of source filenames supplied for the next command to act upon. These filenames should be separated by blanks and should conform to either legal Mesa or CP/M filenames depending on the next particular command to be executed.
Notes:
1) For the Floppy - List! command, the only wildcard pattern - matching supported in Version 1.0 is either * or leaving the field blank. It is planned in future versions to support a general pattern directed Retrieve!, Store!, Floppy - Delete! or Floppy - List! commands.
2) For the Retrieve!, Store!, Floppy - Delete! and Floppy - Rename! commands, the list of filenames supplied in this field should be separated with blanks.
3) It should be noted that NO wildcard expansion is implemented in Version 1.0 for the filenames appearing in either Source: or Dest'n.

Dest'n:

The list of filenames for the destination of a transfer. If this field is left blank, then the destination file name is the same as the source.

Notes:

1) The entry in this field must conform to the file naming conventions of the destination of the transfer. Thus, when using the Retrieve! command, this field should contain a list of legal Mesa filenames, separated by blanks, and the Source field should contain the list of CP/M filenames desired to be retrieve. In the case of the Store! command just the opposite should be the case.

2) It should be noted in the cases of the Retrieve!, Store!, and Floppy – Rename! commands their should be a one – to – one correspondence of filenames in the source and destination lists. If it becomes the case where this constraint is not satisfied then the last filename in the list is replicated in the case of either the Store! or Floppy – Rename! commands.

For the Floppy – Rename! command this field should contain the list of NEW filenames corresponding as the Source field should contain the list of CP/M filenames to delete from the floppy.

In the Floppy – Delete! command this field is unused as the Source field should contain the list of CP/M filenames to delete from the floppy.

VerifyCPMFilenames:

If this boolean is selected, indicating TRUE, then verify that the individual filenames appearing in the list of filenames appearing in either the Source or Dest'n fields for the Retrieve!, Store!, Floppy – Delete! or Floppy – Rename! commands satisfy the legal CP/M filenaming conventions.

If this boolean is selected then on reading or writing the floppy the filenames supplied for the CP/M directory entries are checked to see whether they conform to what "CP/M" considers a legal filename. In the default case this boolean is FALSE and any filename that does not exceed 11 ascii characters in length can be supplied. It should be noted that it will be the case that illegal filenames can be written to the floppy by this tool that cannot be retrieved by a CP/M system because the filenames appearing in the CP/M directory do not conform to the "standard". The standard allows most legal 7 bit ascii characters with the exception of the following characters, (* : ; ? < > & ~ % $ # @ ! + = ADD MORE?????).

**III. Commands

1) CloseFloppy!

Writes the in – memory CP/M directory back to the floppy disk if necessary and closes the floppy volume.

2) Floppy – Delete!

Deletes the list of CP/M filename appearing on the source line from the floppy disk. Checks to see if each individual filename appearing in the list is a legal CP/M filename if VerifyCPMFilenames is selected

3) Floppy – DeleteAll!

Deletes all files from the floppy disk. Note: A red mouse click is required for confirmation of this command before it is executed. This is a powerful command so caution should be noted.

4) Floppy – List!

Lists the CP/M filename and size of the file in bytes of each file that appears in the CP/M directory. Note that in Version 1.0 NO pattern directed list is supported.

5) Floppy – Rename!

Rename the list of OLD CP/M filename(s) appearing in the Source field on the floppy and give each of them the NEW filenames listed on the Destn field.

6) OpenFloppy!

Reads the CP/M directory from the floppy disk into memory. Hence any directory modifications are performed on the in – memory version and are only reflected back to the floppy disk when the CloseFloppy! command is invoked. This allows for faster execution of directory related commands by avoiding a large number of floppy drive access requests. It should be noted that if directory modifying commands were invoked and the directory is not rewritten back to the floppy then the previous contents of the directory should still be intact if major changes in the directory were not invoked.

6) Retrieve!

Retrieve the list of filename(s) appearing in the Source field from the floppy to the currect local directory giving them the corresponding Mesa filenames listed on the Destn field. If the Dest'n field is blank then the CP/M filename will be used in each case.

7) Store!

Store the list of Mesa filename(s) appearing in the Source field to the floppy giving them the corresponding CP/M filenames listed on the Destn field. If the Dest'n field is blank then the Mesa filename will be used in each case if possible.

8) Storage!

Print the number of allocated sectors out of the total number available on the floppy disk. Notice this gives the sector count and NOT the byte count of storage used.

**IV. Implementation Details

1) It should be mentioned that due to speed limitations of accessing single random sectors on the same track of the floppy drive via Pilot's FloppyChannel interface, a track sector cache was implemented to avoid the slow accessing times. It appears that the 8085 support of the floppy is unable to transfer multiple single sector requests for the same track without losing a revolution between each sector. The addition of the cache provided significant performance improvements over the single sector times and allowed read/writes from/to the floppy to be performed at track transfer rates.

2) In the cases of cache invalidations or on closing the floppy volume, the track cache is flushed to the floppy to maintain consistency of the floppy disk data.

3) In the interest of avoiding excessive floppy access times to update the CP/M directory, the directory is read into memory on the Open – Floppy! command and modified in memory during subsequent operations that modify the directory. The directory is flushed to the floppy disk if it was modified when the Close – Floppy! command is invoked. Hence the user should be warned that the Close – Floppy! command should be used after operations on the floppy are finished under the CPMFloppyTool.

4) The "cpmOther" option of the Format parameter allows the user to specify the detailed constants used in reading the floppy disk. These include the sectorInterlace value, bytesPerSector, sectorsPerTrack, etc. Hence the implementation is structured so that a wider variety of "home – brew" values can be accomodated. The only major limitation I see is noted as point 2 in section VI. on Future Enhancements below

5) It should be noted that for the single – density single – sided (SDSS) CP/M floppy case, the CP/M directory is allocated the first two groups of the available groups on the floppy. The SDSS case has 128 bytes/sector, 8 sectors/group or 1024 bytes/group. Each CP/M directory entry requires 32 bytes so the maximum number of entries is 64 is this case.

The design of the CP/M filesystem is constrainted to have a maximum file size of:
128 Extents*16 Groups*8 Sectors/Group*128 Bytes/Sector = > ~2 MBytes.

Since each extent requires a directory entry, the maximum usable storage in the SDSS case is:

64 Extents*16 Groups*8 Sectors/Group*128 Bytes/Sector = > ~1 MByte.

**V. Future Enhancements.

1) Pattern directed support on the following commands: Retrieve, Store, Delete, List.
2) Allowing the sector interlace/interleave value to be either supplied or designating a file containing the logical – to – physical sector mapping table constants to be used.

- - CreateWindow.doc

CreateWindow replaces the default MCRs for creating windows (ExecOps$FileWindow and FileWindow$Create) with a new one similar to Smalltalk's.

If you hold down point and drag the mouse, it will display a box on the screen the size of the window it will create. It creates the window when you let up on point. It cancels the operation if you hit Adjust. If you click Point the behavior is the same as the current MCRs. All of this is specified in CreateWindow.TIP. If while holding down on point, you "back over" your original starting point, it will move the starting point (the same as Star's "lasso" selection mechanism).

- - - - - - - - -
- - File: CreateWindow.TIP


```
SELECT TRIGGER FROM
   Point Down AND Point Up BEFORE 200 = > COORDS, MakeDefaultWindow;
   Point Down = > COORDS, StartLasso;
   MOUSE WHILE Point Down = > COORDS, MoveLasso;
   Point Up = > COORDS, FinishLasso;
   Adjust Down = > Abort;
   ENDCASE...
```

- - Crypt.doc

Environments: CoPilot|Tajo
Description: Crypt encrypts and/or decrypts files using the DES encryption scheme with cipher - block chaining. The encryption is implemented in software and is not particularly speedy: it takes approximately 1.6 ms per byte + .75 sec per file.

How to Use Crypt:

Tersely: Crypt.~ <keystring> <fileName>[/mode] {[<newkey>/k] <fileName>[/mode]}

where <keystring> is a password string; if it includes any separators (such as spaces), it should be enclosed in quotation marks. Case is significant.

<fileName> is the name of the unencrypted member of the input/output file pair (" - cr" will be appended for the name of the encrypted member);

<mode> is either e or d, (for encrypt or decrypt). If unspecified, <mode> continues its last value (originally e).

Brackets [] indicate optional items; braces {} indicate repetition.

Example:
>Crypt "Now is the time." clearfile1 cipherfile1/d "different strokes for different folks"/k cipherfile2 clearfile2/e

Clearfile1 is encrypted with the key "Now is the time.", yielding the ciphertext file clearfile1 - cr. Cipherfile1 - cr is decrypted with the same key, yielding the cleartext file cipherfile1. The key is changed to "different strokes for different folks" and the same process is repeated for clearfile2 and cipherfile2.

Crypt registers itself as a command with the Executive. When invoked, it takes a password string, followed by a series of filenames, each with an optional mode flag appended; new passwords may be inserted in the sequence of filenames. The password is translated into an internal 64 - bit DES key (of which 8 bits are parity), and used to encrypt or decrypt files named thereafter.

Crypt assumes that ciphertext files are named the same as their cleartext counterparts, with " - cr" appended. It uses this scheme to create the output file for encryption and to find the input file for decryption: when asked to decrypt "foo," it looks for "foo - cr" as a ciphertext source, and writes "foo." Its initial mode is to encrypt files; it maintains whatever mode it is in for succeeding files until told to change via the mode switch on a file name.

CAUTION: Several implementations of the DES are floating around. To the extent they are correct, they will all produce the same output when presented with the same raw data, mode, and 64 - bit key. However, the functions for converting from a password string to the 64 - bit key, and for initializing the cipher - block - chaining approach to encrypting streams are NOT equivalent. Files encrypted with Crypt will likely not be decipherable with other systems built on DES and vice versa, despite use of the same password string.

Details of the Encryption:

In the following discussion, bytes, words, etc. have their bits numbered from 1 at the high - order end. Crypt is a shell which handles communication with the Exec, file i/o, and driving the actual encryption routines contained in DESImpl.

The password string is converted into a key by stripping the high - order bit (bit 1) of each successive character, and then XORing the remaining bits (2 - 8) into successive 7 - bit slots in the key - buffer, treated circularly. [Thus, for the 10th character of the string, its bit 2 is deposited in bit 64 of the key, and its bits 3 - 8 are XORed with bits 2 - 7 of the first character, stored in bits 1 - 6 of the key.] After the password string is exhausted, bits 8, 16, ..., 64 of the key are clobbered so that each byte has odd parity; this imposition of parity is the only portion of the procedure which is covered by standard. Evidently, long passwords yield much more random keys.

A file is encrypted by taking successive 64 - bit blocks, XORing them against a 64 - bit "scrambler" block, and then encrypting it under the key provided. The initial XOR eliminates regularities in cleartext. For all blocks but the first in the source, the "scrambler" block is the ciphertext of the preceding block (hence, Cipher - Block Chaining). For the first block, Crypt scrambles against the encryption key. (DESImpl will accept any 64 - bit initial value for the chaining. This simplification is not very compromising, since on decryption, the unscrambler is available as the previous block's ciphertext.)

- - DebugAssist.doc

DebugAssist is an improved interface to CoPilot's interpreter. It makes the interpreter easier to use by making it easy to edit the strings passed to the interpreter. It also provides a better interface to certain commonly – used forms of LOOPHOLE.

The DebugAssist tool is a form subwindow with seven items:

Expr (string): This is the expression you want to interpret. This string may be modified before it is passed to the interpreter, subject to the values of other fields in the tool.

Type (string): If this is non – NIL, the expression "LOOPHOLE [Expr, Type]" is constructed and passed to CoPilot's interpreter.

deref (boolean): If this is TRUE, an up – arrow character (↑) is appended to the expression.

DoIt! (command): Invoking this command invokes the interpreter with a string, constructed as described above. The same effect may be achieved by pressing the "DOIT" key (labelled "MARGINS" on the standard DLion keyboard) while the cursor is in the tool window.

SIGNAL! (command): Interpret the "Expr" parameter as a SIGNAL (i.e. "LOOPHOLE [Expr, SIGNAL]"). The "Type" and "deref" parameters are ignored.

Another! (command): Create a new instance of the DebugAssist tool. The cursor will change to a picture of a mouse, asking you to click the red mouse button over the location of the desired new window, or the blue mouse button to cancel the command. Any number of instances may be created.

Destroy! (command): Destroy this instance of the DebugAssist tool. If only one instance exists, it cannot be destroyed.

Examples of use:

Expr = "someVariableName", type = (empty string), deref = FALSE
=> interprets "someVariableName"

Expr = "someVariableName", type = (empty string), deref = TRUE
=> interprets "someVariableName ↑ "

Expr = "condition", type = "PSB$Queue", deref = FALSE
=> interprets "LOOPHOLE[condition, PSB$Queue]"

Expr = "21270B", type = "SomeImpl$Handle", deref = TRUE
=> interprets "LOOPHOLE[21270B, SomeImpl$Handle] ↑ "

Expr = "1017B", "SIGNAL" command invoked
=> interprets "LOOPHOLE[1017B, SIGNAL]"

-- DefList.doc

DefList.~ takes a list of mesa modules containing DIRECTORY clauses and spits out a single list of the interfaces referenced by those modules. Interfaces residing on the local disk will not be included in the list.

-- DeleteOldVersions.doc

The purpose of the DeleteOldVersions tool is to allow the user to easily delete all versions of a certain set of files, except for a specified number of the most recent versions. This is very similar to the "Delete" command when chatting with an IFS, but it works for both IFS and NS servers.

The tool has two subwindows: a FormSW and a LogSW. The LogSW is where the tool communicates to the user, and where the user types confirmation information.

The FormSW has the following items:

Host: String item name of host server

Directory: Directory name for files to be deleted (Sorry -- no patterns here for NS servers -- that's just the way NS servers work)

Files: Pattern for names of files to be deleted.

Keep: Number of most recent versions of each file to keep. (Note that this item may be changed during the running of the tool, so be careful)

Confirm: Boolean item. When set, requests confirmation for each delete. See confirmation information below. (Note that this item may be changed during the running of the tool, so be careful)

Connect, Password: Secondary credentials.

Go!: Starts the deletion process.

CONFIRMATION

Commands are confirmed by typing (into the LogSW) one of three letters in response to a prompt -- a filename followed by a question mark:

A -- abort (stop the tool)
Y -- yes (delete this file)
N -- no (don't delete this file)

-- DepressAndSpell.doc

DepressAndSpell is a combination of the Depress82 and SpellChecker programs. You can use this hack to spell check an interpress master from Star documents. The program will give you for each page in the interpress master of list of words it thinks is misspelled. This program needs SpellChecker.BitTable to run.

Interpress File: is the interpress file to be spell checked. This can be local or remote.

Private Dictionary: is the name of your current private dictionary. The private dictionary is used to hold words that are not in the SpellChecker standard dictionary. You can have more than one private dictionary, to create a new one or to load an old one, put the name of the dictionary file (default is Interpress.BitTable) in this field and hit ChangeDictionary!

Spell! will start the spell checker

Add! allow you to add words in the current selection to the dictionary. The added words will be appended to a .dictTxt file. For example, if the current dictionary is Interpress.BitTable, the text form of the added words will be in Interpress.dictTxt.

Delete! will delete the words in the current selection from the dictionary. It is advised to use this command with discretion since in may delete more entries than you anticipate.

ChangeDictionary! will cause the file in the Private Dictionary field to be loaded.

Sort will cause the output be be sorted and remove any redundant words.

Strip Plurals is a heuristic to reduce the number of erroneous matches by the spell checker. This will try a word that ends in "s" without the "s". It also tells the spell checker to ignore words with less than two characters.

Starting page = is the first page of the interpress document, default = 1.

MicasPerChar = is something from the Depress82 hacks, defaults to 222

-- DFDelete.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- DF.bisk.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- DFSubstitute.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- DirectoryMenu.doc

DirectoryMenu adds a menu to the root window the entries of which are the directories on the current volume. Selecting an item on the menu causes the directory name to be inserted at the current type-in point. The program keeps track of directories being created and deleted, and inserts or removes the corresponding items from the menu as appropriate.

-- DFTool.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- DirectoryMenu.doc

DirectoryMenu adds a menu to the root window the entries of which are the directories on the current volume. Selecting an item on the menu causes the directory name to be inserted at the current type-in point. The program keeps track of directories being created and deleted, and inserts or removes the corresponding items from the menu as appropriate.

XEROX

DF Software Reference Manual

Outline

Introduction

This document is based on Eric Schmidt's *DF Files Reference Manual*. It describes how to use the Klamath versions of the DF software. A companion document, *DF Release Tools Reference Manual*, describes the programs that are used by people who are responsible for doing software releases.

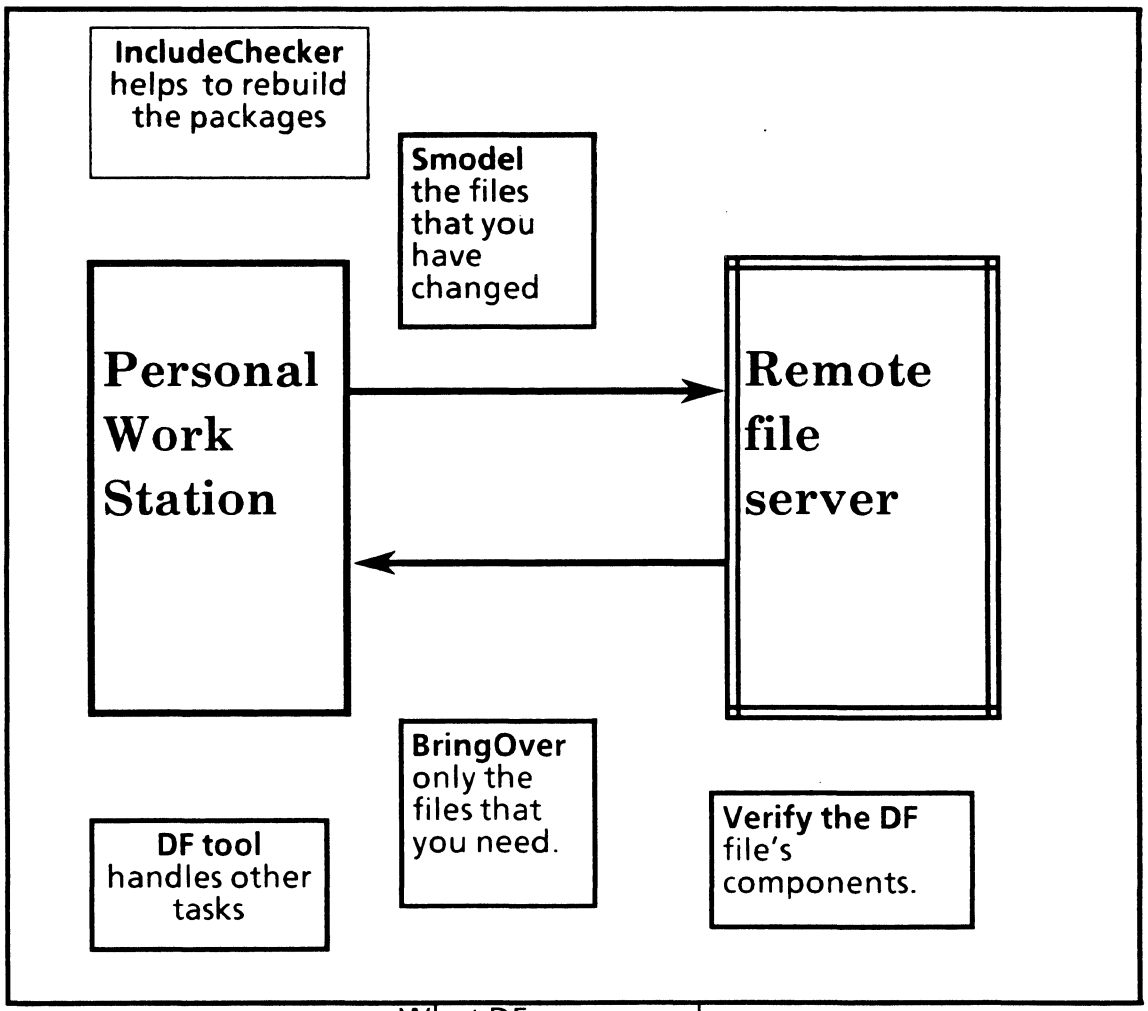
Why should one use the DF software?

The DF software helps the user keep track of files that you work on. These files may be program source and object files, or simple text files. Since it has the ability to describe the version and the location of files, the user needs to worry less about knowing were the files are located and knowing which versions of the files to use. A single DF file may describe all the required files in your program, thus you could simply use the name of the DF file to bring over all the files needed to work on a program from a remote file server to your local disk. After you modify and recompile some files, the DF program will store back only the files that were changed. This frees the user from remembering which files were changed. DF files may explicitly import and export files, in a manner similar to Mesa

programs or C/Mesa configurations. This allows careful sharing of programs between implementors. Having a software system described by a DF file allows one to use tools for managing files, verifying program consistency, and allowing programs be a part of a major software release. The DF software frees the user from bothering with the time consuming, yet important details of tracking program versions and locations. This manual introduces the user to the software and will also serve as a reference manual.

What are the DF programs?

The DF (Describe Files) programs comprise a general package for file management with explicit version control. These programs manipulate DF files, which are essentially lists of file names, fully qualified with remote location and create date. Each DF file typically corresponds to one software component.



What DF programs do.

Of the DF programs, these four are the most heavily used:

BringOver — retrieves the files listed in a DF file from their remote file servers, possibly overwriting different versions already on the local disk. It

insures that all files for a component, and the correct versions of those files, are on the local disk.

SModel — stores changed versions of files back on remote file servers and produces a new DF file containing references to the newest versions. Normally, the new DF file is also stored remotely for use by clients of the component.

VerifyDF — checks that a DF file is complete and consistent. That is, that all files needed to build the top-level object files of a component are listed in the DF file and are consistent in the Mesa compiler and binder sense.

DFTool — provides a window interface to the other DF programs.

The Klamath IncludeChecker can also check DF files and generate command files to rebuild their packages. These capabilities, which are not described in the *Mesa User's Guide*, are discussed in section 11.

The DF files system was initially used by people running Mesa on shared Dorados, (a high performance personal computer) who wanted to guarantee they had the correct version of files they needed and as an easy way to save changed versions of file without unnecessary copying. It is now being used to partially automate the Klamath and Cedar release processes within Xerox

1 Files

The DF programs are on the Klamath archive directory:
<APilot>11.0>DFFiles>Public>

and

The IncludeChecker is on the Klamath system test directory:
<AlphaMesa>11.0>.

If you are not in Xerox SDD, please refer to the release directories specific to your organization. If you using an earlier Mesa release such as Sierra (Mesa 10.0), please refer to an earlier version of this document.

2 An overview of DF files and their use

The DF file for a software component usually has three parts:

- A list of files exported by the component. These are interface or implementation files that are needed by clients; for example, `Space.bcd` and `Compiler.bcd`. DF file Exports (and Imports, described below) are analogous to Mesa module Exports (and Imports). A DF file normally exports itself; this self-reference causes SModel to store the DF file on a remote server whenever it changes

- The component's implementation: the list of files that comprise the component but are of interest only to implementors (e.g., implementation modules).

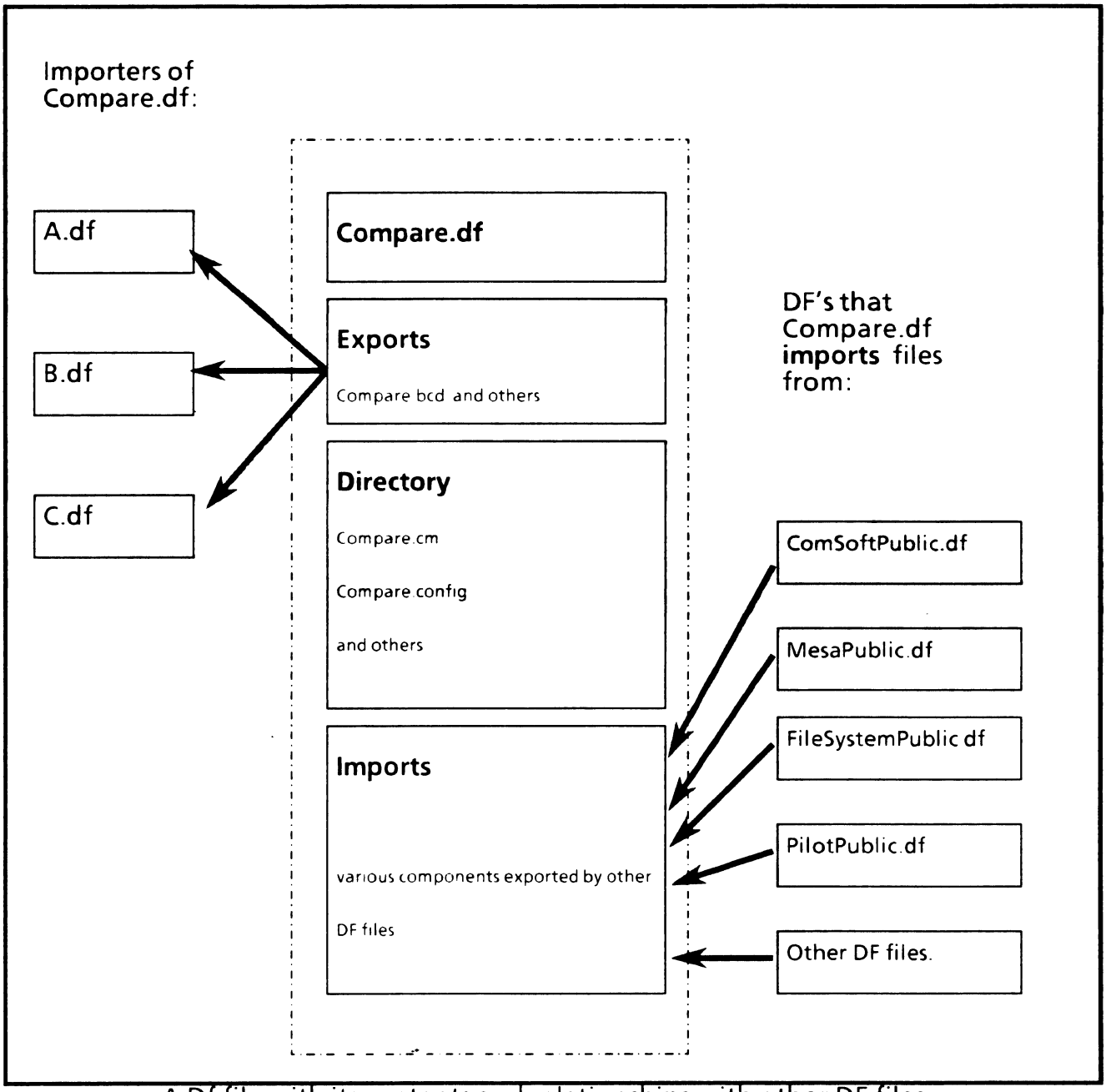
- The imported files needed to build the component (e.g., **Environment.bcd** and **String.bcd** for many programs). These are usually public interfaces exported by another component.

2.1 An example DF file

Probably the easiest way to understand DF files is to consider an example. The following is a DF file for the Compare utility.

```
-- Compare.df  Last edited by Joe on 8-Feb-83 13:36:58

Exports [Igor]<Emerson>DF>     ReleaseAs [Idun]<APilot>DF>
    Compare.df                                  22-Feb-83 13:59:40 PST

Exports [Igor]<Emerson>Compare>Public>  ReleaseAs [Idun]<APilot>Compare>Public>
    + Compare bcd!18                            22-Feb-83 13:53:16 PST
    Compare.symbols!7                           22-Feb-83 13:53:18 PST

Directory [Igor]<Emerson>Compare>Private>  ReleaseAs [Idun]<APilot>Compare>Private>
    Compare.cm!2                                16-Nov-82 10:16:29 PST
    Compare.config!4                            16-Nov-82 10:21:06 PST
    CompareControl bcd!13                       22-Feb-83 13:50:09 PST
    CompareControl.mesa!9                       22-Feb-83 13:49:53 PST
    CompareDefs.bcd!6                           16-Nov-83 11:48:21 PST
    CompareDefs.mesa!4                          16-Nov-83 10:16:47 PST
    CompareImpl.bcd!13                          18-Feb-83 14:55:08 PST
    CompareImpl.mesa!10                         18-Feb-83 14:55:00 PST
    CompareWindow.bcd!3                         23-Dec-83 13:52:41 PST
    CompareWindow.mesa!2                        23-Dec-83 13:51:53 PST

Imports [Igor]<Emerson>DF>ComSoftPublic.df Of #
    Using [Ascii.bcd, Format.bcd, Heap.bcd, String.bcd, Time.bcd]

Imports [Igor]<Emerson>DF>MesaPublic.df Of #
    Using [Environment.bcd, Inline.bcd]

Imports [Igor]<Emerson>DF>FileSystemPublic.df Of #
    Using [MSegment.bcd, MStream.bcd]

Imports [Igor]<Emerson>DF>PilotPublic.df Of #
    Using [Process.bcd, Runtime.bcd, Stream.bcd, System.bcd, UserTerminal.bcd]

Imports [Igor]<Emerson>DF>TajoPublic.df Of #
    Using [Exec.bcd, FileName.bcd, FileTransfer.bcd, FormSW.bcd, Put.bcd,
        Tool.bcd, ToolWindow.bcd, UserInput.bcd, Version.bcd, Window.bcd]
```

The files exported by the Compare package (including the DF file itself) are marked with the keyword **Exports**, other files that are part of Compare are marked with **Directory**, and imported files have the keyword **Imports**. The ReleaseAs clauses are used to tell a program



A Df file with its contents and relationships with other DF files.

called the ReleaseTool where to store the files during a release. Only the files that are part of the component (i.e. not Imports) have ReleaseAs clauses.

Imported files, such as **Exec.bcd** and **MFile.bcd**, are retrieved when the DF files is brought over. However, since the DF file does not "own" them, they will not be stored by SModel. The Imports clauses in this DF file have explicit Using lists. If no Using list is given, all exported files in the Import DF file are assumed. Having a Using list is generally a good idea, since it documents which files are needed and speeds up **BringOver**. Note that imported files are gotten indirectly, by pointing to another DF file (the one for the component that exports those files). An importer doesn't need to know

anything about where the imported files are stored, or even their versions, since all that information is in the imported DF file.

A file is specified by a full path name (remote host and directory), an optional file server version number, and an optional creation date. The create date is used to uniquely determine the correct version of the file, while the version number is used as a hint to reduce the time needed to locate that correct version. If the create date is omitted, the highest remote version is assumed. In most cases, however, the create date has been filled in by the DF program.

An Extremely Important note: If you are storing your files on *NS* servers, please be sure to use the *fully qualified names* . For example, use:
[Tundra:OSBU North:Xerox] instead of
[Tundra:].
This is particularly important for files that are imported by other DF's, since the users may be dispersed across NS domains.

There are two special "create dates": ">" and "#". If the newer remote version of the file should always be brought over, ">" is used. The "#" specifies any remote version of the file that has a *different* create date than the version on the local disk. These two "create dates" support the *loose binding* of imports. If one imports FileSystemPublic.df of ">", Compare.df will always retrieve the MSegment.bcd and MStream.bcd described by the FileSystemPublic.df *most recently* Smodel'ed by its implementor. In general, files that are part of a component (i.e. Exports and Directory files) have explicit create dates.

We recommend the use of the "#" type of create dates for *imports* because it allows one to bringover old DF's for maintainance updates. If the ">" create date are used, you may accidentally use newer versions of imports which *happen to be* on your local disk. Furthermore, the "#" create will also bringover the correct version of files under normal development. One typically imports files from a release directory which has an unambigious reference to a file, thus you would want the released version of the imported files, regardless of whether or not it is newer than the one on the local disk. The user should still consider the development practices in use and use the "right" mode. For example, you may not have an official release directory to import from, or perhaps you have some reason to avoid the use of older files.

The + in front of Compare.bcd indicates to the program VerifyDF that it is a top-level object file. A top level file may be of two types. First, it may be a .bcd or a .boot file. If so, VerifyDF will insure that all files needed to build Compare.bcd are listed in the DF file and are of the correct version. Otherwise it may be a file that is not a part of a component, such as documentation files. This prevents the program from doing unnecessary analysis .

Fine point:

> some DF files also have files marked with "*". The * is ignored by all DF programs except for the ReleaseTool; it indicates files that must be copied onto [somehost]<archiveDirectory> or [anotherhost]<systemTestDirectory> after a release.

Blank lines in a DF file are ignored, and lines are treated as comments if they begin with "--" or "//".

2.2 A typical development scenario using DF files

DF files have little inherent structure or semantics. There is no requirement, for example, that the files they describe be consistent in the Mesa compiler sense (this allows DF files to be used to back up arbitrary files on personal workstations). However, DF files can provide considerable assistance for development if they are used in a stylized manner.

To illustrate the use of DF programs in program development, assume that you had to fix a problem with Compare. The steps you would take normally include the following:

1. BringOver Compare.df. This insures that all files needed to build Compare, including imported files, are on the local disk in the correct version. BringOver will check to insure that you are using the most recent version of Compare.df.

2. Modify and test Compare. Since a DF file is just a text file with a fairly simple format, it can be edited whenever it is necessary, for example, to add new a module. Also, to assist in rebuilding its component, Compare.df, like many DF files, points to a command file (Compare.cm) that can be used to build the component from scratch; text can be selected from this command file and stuffed into the executive. It is also possible to run the IncludeChecker on a DF file to generate a command file for its reconstruction.

3. SModel Compare.df. This stores back changed files and updates Compare.df to reflect the new versions. In general, you do not have to think about what files have changed, you can just SModel the component.

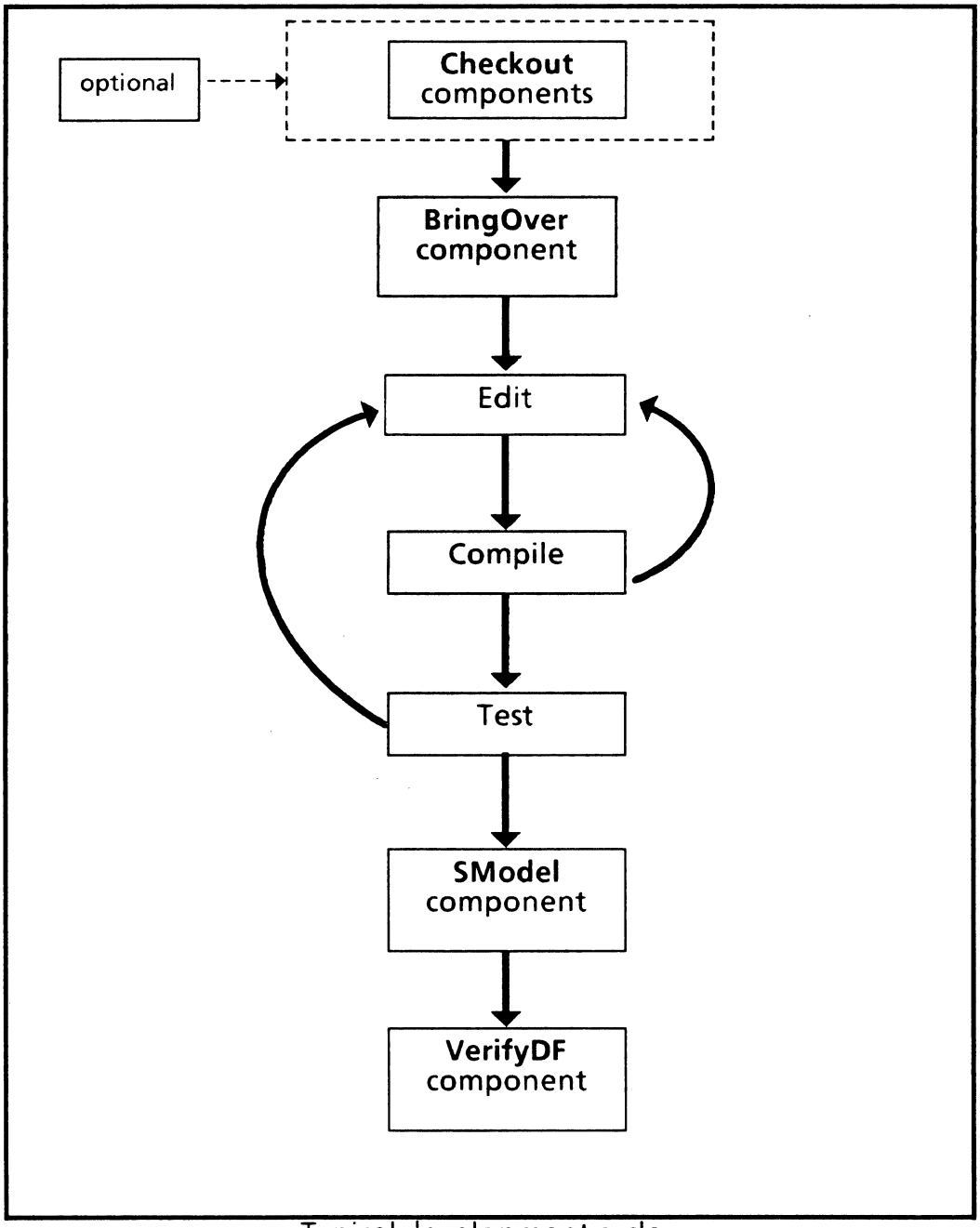
4. VerifyDF Compare.df. This verifies that Compare is complete and consistent. At this point, you can let users know about the new version of Compare.

2.3 Releases and the use of file server directories

In using the DF software, three directories are of special importance. These are the *working*,*integration*, and *archive* directories.

Each group of software developers has a separate working directory that holds the latest versions of their software. For example, this directory is [Rasp:OSBU North:Xerox]<Emerson> for the Mesa group. The Directory and Exports clauses in the group's DF files point to the working directory, and that directory is the source and target of most BringOver and SModel runs. Experience has shown that it is useful to set aside a subdirectory of the working directory, e.g. [Rasp:OSBU North:Xerox]<Emerson>DF>, as the location for the group's "working" DF files. This simplifies finding DF files and allows BringOver (when the `DefaultDFLoc:` entry of the `User.cm` is set) to insure that only the most recent versions of DF files are used.

The integration directory is shared by development groups. A component is stored onto the integration directory when it has been tested and verified (using VerifyDF), and its developers believe that it is ready for use by other groups. Components are stored onto the integration directory by using SModel's prerelease mode. The integration directory is used to communicate software between groups. A development group should only obtain (Import) software from another group that has been stored onto the integration directory;



Typical development cycle.

it should never use software or DF files from the private working directory of another group. The integration directory for the Mesa group and all of System Software, for example, is [Idun]<Int>. The integration directory also serves as a staging area for a release.

A *release* is a set of compatible software components that have been saved in a safe location. The ReleaseTool verifies that a set of DFfiles is globally consistent and complete, copies the files to the release directory, and generates new DF files that describe the release. The new DF files are fully bound: all Imports (and Includes, which are discussed in section 2.7) are specified with explicit create dates (there are no > or #'s). Only the

ReleaseTool stores files onto the release directory. The release directory is named by ReleaseAs clauses in each DF file. For example, the release directory for Klamath is [Idun]<APilot>. Other directories may follow as required by the user organization as shown in the diagram below.

Fine points:

It is possible to override the release location with the ReleaseTool.

The new DF files generated by the ReleaseTool also have *CameFrom* clauses in place of the original ReleaseAs clauses. A CameFrom clause for a file documents the location on the prerelease directory from which the file was copied. For example, the ReleaseTool will change

Exports [Idun]<Int>Compiler>Public> Compiler.bcd!27
ReleaseAs [Idun]<APilot>Compiler>Public> 29-Feb-83 11:16:37 PST

to be

Exports [Idun]<APilot>Compiler>Public> Compiler.bcd!1
CameFrom [Idun]<Int>Compiler>Public> 29-Feb-83 11:16:37 PST

2.4 Creating a new DF file from scratch

The easiest way to generate a DF file for a component is to do the following:

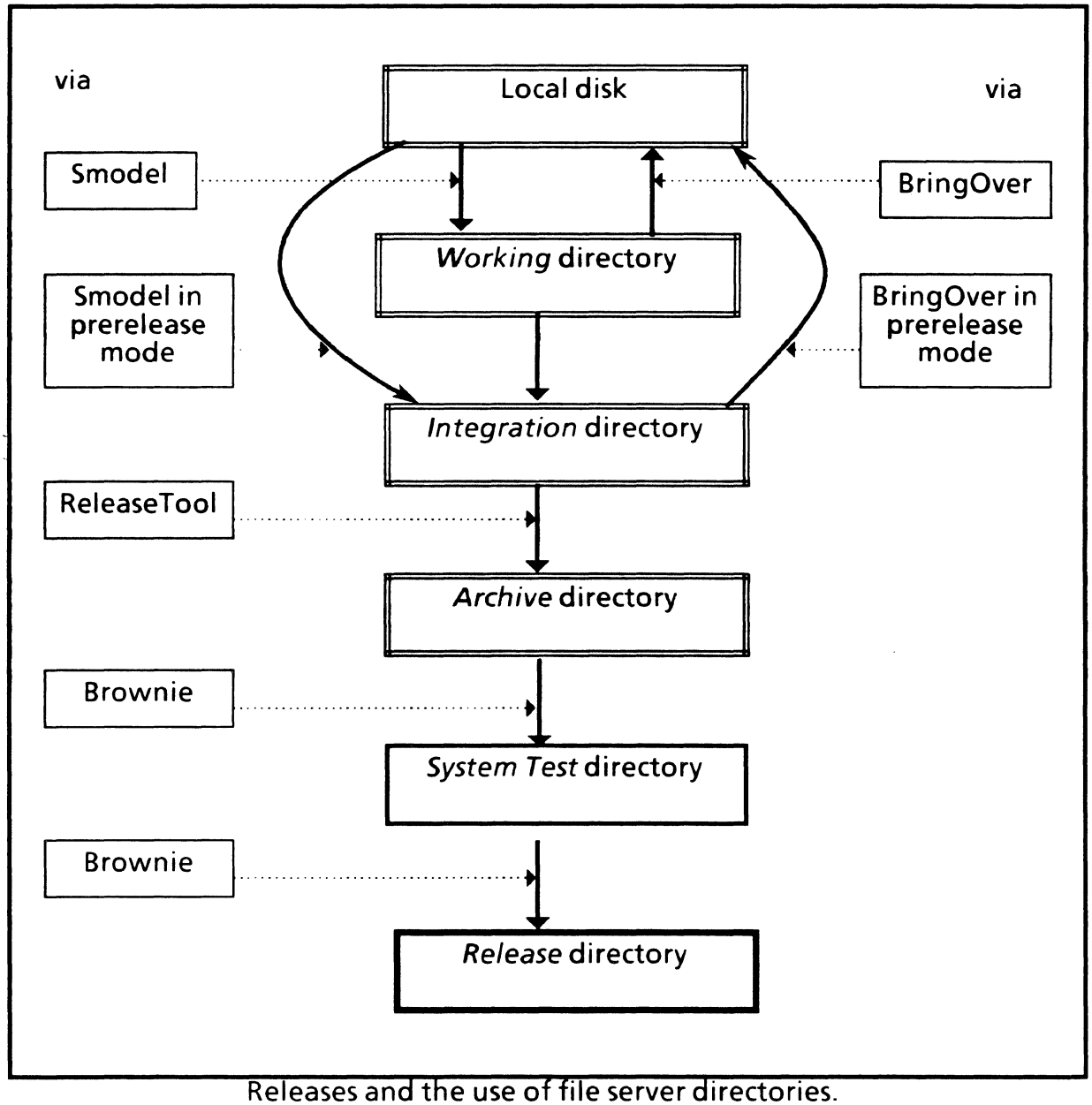
1. Get all of the component's files onto the local disk. Make sure that the local versions are the same as those on the remote file server (to keep from confusing yourself or the DF software).

2. Compose a skeleton DF file for the package that lists its files under the appropriate Directory and Exports lines. It is not necessary to fill in create dates since this will be done by SModel in step 3. Make sure that the remote locations and ReleaseAs locations are correct. Add any Imports that you can think of.

3. Run SModel on the skeleton DF file with the /n (don't store files remotely) switch. This will rewrite the DF file with the create dates filled in (and **will not store any files**).

4. Use VerifyDF to check the DF file. It will report any missing files, or files that have the wrong versions. Correct the DF file as necessary and repeat this step. The Find utility is often useful for locating the DF file that describes a needed import. The Find utility may be used to search over a number of DF files for the one that contains the files that you need. For example:
   > find System.bcd [Host]<Directory>DF>*.df
   will search over the DF files in the specified directory for the one that contains System.bcd. Ignore the DF files that Imports System.bcd, and look for the one that Exports it. The Exporting DF file would contain the create date, while the Importers would include System.bcd in the Using list.

fine point: If you are importing a file that is a part of a software release managed by DF software, you may be able to use a utility called DFetch. This program is still not part of the general release. This program will query a database using a file name as a key and will return the name of the DF file that contains it.



Releases and the use of file server directories.

5. When VerifyDF no longer complains, run SModel to store the DF file itself and the files it describes remotely Since you are running Smodel for the first time, use the /v switch to make the program verify that your files exist in the destination and store it there as necessary.

2.5 DF files and libjects

If more than one person is responsible for a software component, it is important to prevent simultaneous modification of both the individual files of the component *and* its DF file. This is because the DF file points to specific versions of the component's files. To deal with

this problem, each DF file should have a program librarian libject. There are libjects for each Klamath DF file maintained by the Mesa group.

When a component is to be worked on, its DF file is first checked out, typically using the "Access" program. After the component has been changed and tested, it is SModel'ed, which will check in the libject for each changed file, including the DF file itself. If another person attempts to work on the package at the same time, he will be unable to since the libject is already checked out. Typically, one checks out a libject for only the DF file.

If this methodology appears to be too restrictive for some large component, there are two possibilities: 1) break up the component into smaller pieces, each with its own DF file and libject, or 2) adopt a more complicated checkout and checkin scheme. The Mesa group's experience has shown that it is much simpler and less error prone to break up the DF file. If the component shouldn't be broken up, and it is necessary for more than one person to be modifying (different portions of) it at the same time, the following methodology can be followed by each maintainer:

1. BringOver the component's DF file. Do not check out the DF file at this time. Do check out libjects for component files that you will be changing.

2. After modifying and testing the component, but just before SModeling it, check out the DF file. Successfully checking out the DF file means that you currently have the right to change the description of the "truth" on the shared remote directory (i.e. the DF file).

3. Now BringOver the DF file and its components *again*. This will use the most recent DF file, which might be newer than the one you originally brought over (if other people were modifying the component simultaneously) BringOver might retrieve newer versions of files that *others* changed. If so, rebuild and retest your version of the component. You do not have to run BringOver again since you "hold the lock" on storing new versions of the component's files

4. SModel the DF file to store your changed files and the new DF file, and to release your "lock" on SModeling the DF file.

2.6 Use of IFSs and NS file servers

The DF software is able to retrieve and store files on both NS file servers and the PUP-based IFSs. To use a product file server, simply give its fully qualified clearinghouse name. For example,

```
Imports [Rasp:OSBU North:Xerox]<WComm>DF>RS232CPublic.df Of >
    Using [RS232CIO bcd]
```

2.7 Included DF files

Although a component is usually described by a single DF file, there are a few particularly large or complicated components that are more easily described by a set of DF files. An example is the Pilot kernel, which has so many files that it is convenient to have separate DF files for each major subconfiguration and for the public-, friends-, and private-level interfaces. Such a collection of DF files must have a "root" DF file that (directly or indirectly) *includes* the others. This is done with the **Includes** construct, which resembles

the Imports clause described above. For example, a fragment of the root DF file for the Pilot kernel, Pilot.df, is:

```
--Pilot.df  Last edited by Jimmy on 3-Jan-83 21:25:59

Exports [Igor]<Emerson>DF>     ReleaseAs [Idun]<APilot>DF>
    Pilot.df                                3-Jan-83 21:27:32 PST

-- Pilot kernel defs
Includes [Idun]<P>DF>PilotFriends.df Of >   ReleaseAs [Idun]<APilot>DF>
Includes [Idun]<P>DF>PilotPrivate.df Of >   ReleaseAs [Idun]<APilot>DF>
Includes [Idun]<P>DF>PilotPublic.df Of >    ReleaseAs [Idun]<APilot>DF>

-- Pilot kernel subconfigurations
Includes [Idun]<P>DF>Control.df Of >        ReleaseAs [Idun]<APilot>DF>
Includes [Idun]<P>DF>FileMgr.df Of >        ReleaseAs [Idun]<APilot>DF>
Includes [Idun]<P>DF>Filer.df Of >          ReleaseAs [Idun]<APilot>DF>
Includes [Idun]<P>DF>Swapper.df Of >        ReleaseAs [Idun]<APilot>DF>
```

Includes is treated as macro substitution: the effect is to replace the Includes clause with the entire contents of the included DF file. Whenever one of the DF programs such as BringOver is run on the root DF file, it is applied recursively to the included DF files

Note: There is a significant difference between Includes and Imports. The Imports clause is used when files are needed, but they are "owned" by another DF file. Although imported files are retrieved by BringOver, the DF programs do not otherwise recur on imported DF files. SModel, for example, will recursively store *included* DF files but not *imported* DF files.

2.8 ReadOnly files

If your component depends upon some files in a remote directory, but those files are not "owned" (described by) a DF file, you can't just Import them. However, you can document your component's dependence on those files, and have BringOver retrieve them when your DF file is brought over, by listing the files in your DF file and marking their directory ReadOnly. One needs this when you are importing components from implementors who are not using DF files. This practice should be discontinued once the implementors use DF files.

Here is an example,

```
ReadOnly Directory [Iris]<Smith>BTree>
    BTree.bcd                       12-Jan-83 10:17:22 PST
    BTree.mesa                       9-Jan-83 21:27:32 PST
    BTreeImpl.bcd                   17-Jan-83 15:56:19 PST
    BTreeImpl.mesa                  10-Jan-83 11:25:49 PST
```

(The keyword **Directory** after **ReadOnly** is optional). Readonly files are never stored by SModel. Since they are not owned by your DF file, they do not have a ReleaseAs clause to indicate where they are to be stored on a release.

3 User.cm

The DF software `User.cm` section is called **[DFTool]**. The following is a list of the `User.cm` fields used by the DF programs:

WorkingDFLoc: the remote working directory, e.g. [Rasp:OSBU North:Xerox]<Emerson>DF>, that holds your group's "working" DF files. It is used by BringOver when you retrieve a component. If the component's DF file is not local, BringOver will retrieve it from the **WorkingDFLoc:** If the DF file is on the local disk, BringOver will check that it is at least as new as the version on the **WorkingDFLoc:**. By default, the **WorkingDFLoc:** is empty, and BringOver does no checking.

IntegrationLoc: your group's prerelease location, e.g. [Idun]<Int>. By default, this entry is empty, and BringOver and SModel prerelease mode cannot be used.

CheckLibrarian: if **TRUE**, SModel will check, before storing each file, if it has a libject. If it doesn't, SModel simply stores the file. If it does have a libject, there are three possibilities: 1) If the file wasn't checked out, SModel won't store it. 2) If it was checked out, but not by you, SModel won't store it. 3) If it was checked out by you, SModel checks it back in and stores the file. Also, if **CheckLibrarian:** is **TRUE**, SModel's prerelease mode will warn you if any of the files that you are submitting to a release (storing onto the prerelease directory) have libjects checked out. The default value for **CheckLibrarian:** is **FALSE**.

LocalDFDir: the directory on your local disk, e.g. <>DF>, to which BringOver and VerifyDF will retrieve new DF files. Setting this entry helps to prevent DF files from being scattered all over your disk. The local DF directory should always be on your search path, since the DF software always looks files up (anywhere) on the search path. The default for **LocalDFDir:** is empty, and DF files are retrieved to the directory on the front of the search path.

fine point: The DF programs check that the LocalDFDir and LocalDir (see DFTool) are on the search path and a warning is given if they are not. For example, if they are not given in the search path, the **wrong** DF file may be used for a BringOver.

4 BringOver

BringOver runs in the Executive and takes commands from the command line. In the simplest case, to retrieve a DF file and its components, just type

```
>BringOver |Host|<Directory>DF>DFfile
```

BringOver works as follows: It reads the DF file one line at a time. It takes the remote file name listed in that line, strips off the directory information and looks to see if it is on the local disk. One of three things can happen:

- If the file is not on the local disk, BringOver will offer to retrieve it.
- If the file is on the local disk, BringOver looks at the version on the local disk. If the create date listed in the DF file differs from the create date of the local file, BringOver will try to retrieve the remote version. If this would retrieve an *older* version of the file over a newer version, BringOver will first ask for confirmation. This helps to support a "newer is usually better" file management methodology.
- If the create date is omitted from the DF file, BringOver will *always* try to retrieve the file. Again, if this would retrieve an *older* version of the file over a newer version, BringOver first asks for confirmation.

If you omit the file server version number (e.g. "!3"), BringOver will enumerate all the versions of that particular file looking for one with the correct create time. If there are no versions of the file you list in the DF file on the remote host in the directory you specify, BringOver will give you a warning message. If there are files with the same name and none of the create dates available match that listed in the DF file, BringOver will give you a warning and offer to retrieve the latest version.

After running BringOver you can be sure the files listed in the DF file are on your local disk, and that their create dates agree with the create dates listed in the DF file, or BringOver will have printed out error messages.

Normally BringOver will list each file to be retrieved and will ask for confirmation. (You may reply "y" or CR to confirm, "n" to skip retrieval of this file, "q" to stop BringOver altogether, and "a" to retrieve this file and subsequent files as if "y" were typed each time.) The /a switch can be given on BringOver's command line to suppress (most) requests for confirmation:

```
>BringOver /a [Host]<Directory>DF>DFfile
```

If you use the /a switch or reply "a", and an older version of a file would be retrieved over a newer one, BringOver will always stop and ask for explicit confirmation.

BringOver can read a local DF file as easily as a remote one:

```
>BringOver Compare.df
```

You will use a local copy of the DF file when you have done previous BringOver's and Smodel's. If so, you will have a local copy of the DF file that is identical to the remote one. That is because Smodel will modify the DF file and store it remotely, leaving a copy on your local disk.

As files are brought over, a property (called the *RemoteName* property) is added to their leader page recording of the retrieved file so that the Mesa Development Environment knows where the file came from. (FTP and the FileTool also set this property.) These remote locations can be printed out by the DFDisk program (described below in Section 9).

If the create date entry is a > rather than a normal date, BringOver will retrieve the file only if the version on the remote server is newer than the version on the local disk. If the file is not on the local disk, it will be retrieved. As an example,

```
BTree.mesa                >
```

will retrieve BTree.mesa from the remote server only if there is a newer version on the server or no local copies exist.

Similarly, if the create date entry is a #, BringOver will retrieve the file only if the highest version on the remote server is different than the version on the local disk. If the file is not on the local disk, it will be retrieved. For example,

```
BTree.mesa                #
```

An Includes clause, e.g.

```
Include [host]<path>Component.df Of <date>
```

will cause BringOver to invoke itself on Component.df at the point it encounters the Include statement. If the included file itself has an Include statement, BringOver will again invoke itself on the inner DF file, and so on, in a recursive fashion. Furthermore, the DF file itself is retrieved using the usual BringOver rules before the recursive call.

An Imports statement

```
Imports [host]<path>Package.df Of <date>
```

will cause BringOver to 1) retrieve Package.df to the local disk if necessary and 2) examine all exported files in Package.df and retrieve them if necessary. Of course Package.df may have Include or Imports statements, so this is a recursive algorithm.

Appending a Using clause to the Imports statement, analogous to the Mesa language construct, gives the user explicit control over the files to be retrieved. The Using list may be used to obtain files that are under both Exports and Directory headings; That is, files can be obtained with the Using clause whether they are exported or not. Although use of the Using clause is not required, it is strongly recommended.

```
Imports [host]<path>Package.df Of <date>
    Using [list of files, separated by commas]
```

Examples of Imports:

```
Imports [Igor]<Emerson>DF>TajoFriends.df Of >

Imports [Idun]<APilot>DF>CoPilot.df Of 24-Feb-83 11:14:26 PDT
    Using [CPSwapDefs.Bcd, CPSwap2.Bcd]
```

The files referred to by an Imports statement may themselves be exported by preceding the keyword **Imports** by **Exports**. This is useful when users of your package need to have files from some other package in order to, for example, compile their system.

4.1 BringOver modes

There are three special modes in which you can run BringOver.

4.1.1 BringOver only specified files mode

The switch /o will instruct BringOver to retrieve only the files listed on the command line after the /o. The DF file to be used is given last. This mode is often used when you are not working on a package, but you simply need some files that are described by its DF file. For example,

```
>BringOver /o MFile.bcd MStream.bcd MSegment.bcd FileSystemPublic.df
```

will examine and potentially retrieve only **`MFile.bcd`**, **`MStream.bcd`** and **`MSegment.bcd`** in FileSystemPublic.df.

4.1.2 BringOver "verify files exist" mode

The switch /v will cause BringOver to run in *verify files exist* mode, where it will check that the files listed in the DF file actually exist on the remote servers or the local disk. No files are retrieved in this mode. BringOver will inform the user if newer versions were found, and if so will write a new DF file listing the newer versions. Also, if any files were listed in the DF file without their file server version numbers (e.g. !5), BringOver will write a new DF file with those version numbers filled in. This mode is often used to "flesh out" a skeleton DF file with the correct create dates and file server version numbers. If the file is listed correctly and a local copy exists, BringOver will add the RemoteName property to its leader page. Note for large DF files the verify option takes a few minutes.

4.1.3 BringOver prerelease mode

BringOver's prerelease mode is useful for fixing an old version of a package that was submitted to a release. This mode is entered with the /z switch. It brings over the DF file for the package that is on the remote integration directory (named by the `IntegrationLoc:` entry in User.cm). Since this might overwrite newer versions on the local disk, BringOver asks for confirmation before doing any retrievals.

4.2 BringOver's command line

In general, the command line for BringOver has the form

```
>BringOver [/<global switches>] DFfile1[/<local switches>] ... DFfilen[/<local switches>]
```

The optional global switches control the retrieval of the following DF files. You can also set global switches by giving an empty DF file name.

BringOver's "*only file*" mode (/o) has a slightly different format: the files to be retrieved are listed after the global switch /o, and the DF file to be used is named last.

BringOver also recognizes commands, **localDir/c, localDFDir/c, WorkingDFLoc/c**, and **IntegrationLoc/c** that specify subdirectories for file retrieval and storage.

The command **localDir/c** gives the directory for looking up and retrieving files. For example, the command line

```
>BringOver localDir/c <>MyPackage> MyPackage.df
```

will retrieve MyPackage's files to the directory <>MyPackage> on the system volume.

The command **localDFDir/c** names the directory to which DF files themselves (not their contained files) should be retrieved. It overrides any `LocalDFDir:` entry in User.cm. If both **localDFDir/c** and **localDir/c** are specified, the local DF directory is used for DF files; all other files use the localDir/c directory. For those rare occasions when you don't want a package's DF files to go to the User.cm-specified local DF directory (e g. if you're fixing an old version of a package, perhaps one submitted to a release), you can use the **localDFDir/c** command to force the DF files to go to another directory. For example:

```
>BringOver localDir/c <>Old> localDFDir/c <>Old> [Idun]<Int>Stuff.df
```

The command **WorkingDFLoc /c** and **IntegrationLoc/c** overrides any WorkingDFLoc: and IntegrationLoc : entry in User.cm. For example:

```
>BringOver WorkingDFLoc/c [Idun]<P> IntegrationLoc/c [Idun]<Int> MyStuff.df/z
```

It helps developers that have more than one working directory; e.g. those doing both microcode and Pilot development.

4.2.1 BringOver switches

A switch specification is a letter identifying the switch, optionally preceded by a '-' or '~' to reverse the sense of that switch.

fine point: If you are using TTYTajo , please use the '-' rather than the '~'.

The valid switches are:

a	always retrieve without confirmation (unless an existing local file is newer)
b	get just "bcd" (derived) files: .bcd, .symbols, .boot, .signals, .press files
f	force retrieval of all files, disregarding any newer local files.
o	get only specified files: e.g. BringOver /o Exec.bcd Put.bcd Tajo.df
p	get just public (exported) files
r	get just readonly files: Imported (and ReadOnly) files
s	get just "source" files (inverse of /b)
u	only update existing local files (never get new files)
v	verify files exist in the right place and version, and fill in DF dates
w	get just "writable" (Exports or Directory) files (inverse of /r)
x	rename ".bcd" to ".archiveBcd" if an archiveBcd already exists
z	prerelease mode
<	suppress confirmation request if an *older* remote version is retrieved

The default setting for all switches is *off*. You can also change the default setting of any switch by using a global switch. Any switch given with no file name (i.e., just a slash and

switches) establishes the default setting for that switch. Unless overridden or reset, that default applies to all subsequent commands.

4.3 BringOver limitations

Each DF file read by BringOver must contain no more then 450 files. This applies to each Imported and Included DF file as well.

5 SModel

SModel (the name stands for "Simple Modeller") is used to store back new versions of files you've changed since the last time you ran BringOver on a DF file. For example, if you are working on the Compare program and you've already run BringOver on Compare.df, then

```
>SModel Compare.df
```

does the following: The files listed in Compare.df are checked on the local disk. If any have different create dates SModel will offer to store them on the remote servers specified in Compare.df. SModel then produces a new Compare.df file with the new create dates and remote file system version numbers (e.g. !4). The old DF file is saved by copying it to a "$" file, e.g. Compare.df$. Files listed under an "Imports" or a "Readonly" clause will be ignored.

If a file on the local disk is listed without any create date in the DF file, SModel will fill in the create date from the version on the local disk and then offer to store the file remotely. If the file listed in the DF file is followed by a > or #, it is ignored and will not be transferred.

If the DF file contains a reference to itself (e.g. Compare.df lists Compare.df), SModel will also store a new version of the DF file on the remote server. Since SModel has to write out a new DF file before it can store the DF file, SModel cannot put the file server version number (e.g. !5) on the DF file self-reference. However, BringOver will always get the correct version of the DF file since it will use the create date of the DF file instead.

Before storing a file, SModel first checks to see if that file is already on the remote directory (as the highest version); if so, it won't actually store the file. Also, if storing a file would write a version with a create date that is *older* than that of the current highest version, SModel will always ask for confirmation. This helps to support the "newer is usually better" methodology.

If the **CheckLibrarian:** entry in the User.cm is **TRUE**, then before storing a file, SModel will check if the file has a program librarian libject. If so, SModel won't store the file if it wasn't checked out, or if it wasn't checked out by you. (The old create date is left in the DF file, so that in most cases, you can simply check out the file [without retrieving the source] and rerun SModel.) One must be careful when doing this since someone else may have checked out the DF file and Smodel'ed already.

SModel invokes itself recursively on Included DF files. It does *not* invoke itself on Imported or ReadOnly DF files. If the Includes or Imports statement is not followed by an "Of <date>" clause, SModel will insert such a clause in the new DF file with <date> replaced by the create date of the file on the local disk.

5.1 SModel modes

There are three special modes in which you can run SModel.

5.1.1 SModel "verify files are remote" mode

New DF software users are often confused about the relationship of the entries in a DF file to the local and remote directories, and what SModel will do in certain cases. The easiest way to understand it is that SModel assumes 1) that the DF file was an accurate description of the remote directory at some point in the past, and 2) files with different create dates that it finds on the local disk are the "truth" and should be transferred. **This is one of the most important things to know. If you are having trouble with your DF files, always remember that the DF file describes the state of a remote directory**. However, assumption 1) allows SModel to assume that files with the same create date in the DF file and on the local disk *also* exist on the remote server. For this reason, and because remote enumerations are relatively slow, SModel does not check the remote server to see if in fact the files described by a DF file are actually there (unless it has already decided to store a file). So, SModel may not detect that certain files listed in a DF file are not present on a remote server, unless you use a special switch described below.

A common mistake is to assume that if you run SModel on a DF file successfully, and then simply change a Directory in the DF file, then all the files will be copied (again) to the new directory. This is *wrong*! After SModel has been run the first time, the create dates in the DF file are the same as those on the local disk. Since SModel just checks the create dates in the DF file against the local files, the second SModel invocation will not detect that any files need be transferred even though the Directory was changed.

To resolve these problems, SModel has a *verify files are remote* mode which is entered with the /v switch. In this mode, SModel not only applies the algorithm described above to store files, but if it decides a file doesn't need to be stored, it will look on the remote file server and check that in fact the file does not need to be stored. If the file is not on the remote directory, or the version listed in the DF file is not on the remote directory, then SModel will offer to store the file. In this way SModel /v will try to force the remote directory to agree with the DF file.

For example, the following will insure that all files listed in Compare.df are actually on the remote servers:

```
>SModel Compare.df/v
```

5.1.2 SModel "don't store files" mode

The /n switch has SModel do everything it normally does, except for storing files. The DF file is rewritten if there are different versions of files on the local disk, but those files, and the DF file itself, are not stored. This mode is useful for "fleshing out" a DF file with the versions of files that are on the local disk. **This is a dangerous thing to do. The DF file itself simply describes the version and locations of the files listed. If you use the "don't store files" mode, the Df file will be changed to include the create dates of the files on your local disk. Subsequent uses of the DF file will look for those files on the remote server, but they will not exist. Make sure you Smodel again using the "verify files are remote" mode to actually store the files remotely.**

5.1.3 SModel prerelease mode

In this mode, SModel stores a component on your prerelease directory (which is named by the **IntegrationLoc:** entry in User.cm). The actual remote directory for each file is gotten by concatenating the **IntegrationLoc:** directory with the *ReleaseAs subdirectory* for the file (Although this sounds strange, that is the correct location. The Exports or Directory subdirectory, for example, might point off to a temporary or personal directory.)

For example: the Integration directory of: [Idun]<Int> and the ReleaseAs directory of [Idun]<APilot>MyProg>Public will yield: [Idun]<Int>MyProg>Public.

SModel recurs on Included DF files and stores them out as well. Imports clauses are changed to point to the prerelease directory and SModel checks that the imported DF files already exist there. If the User.cm entry **CheckLibrarian:** is **TRUE**, SModel also checks to see if each file has a libject that is checked out: if so, it gives a warning.

5.2 SModel's command line

The SModel command line has the form

>SModel [/<global switches>] $DFfile_1$[/<local switches>] ... $DFfile_n$[/<local switches>]

Global switches are optional and control the store of subsequent DF files. You can also set global switches by giving an empty DF file name.

The subcommands **WorkingDFLoc/c** and **IntegrationLoc/c** work identically to the same commands in BringOver.

Secondary connect credentials can be given on the command line; e.g.

```
>SModel Conn/c Dir Passwd MyComponent.df/z
```

5.2.1 SModel switches

A switch specification is a letter, optionally preceded by a '-' or '~' to reverse the sense of that switch. The switches recognized by SModel are:

a	store always: without confirmation
f	flip CameFrom clause to be ReleaseAs (*default*)
l	check with program librarian (*default*) It overrides User.cm
n	do not store files remotely
r	ignore ReadOnly or Imports designation and store files if different versions
t	process only top (outermost) DF file, not Included DF files
v	verify that files are really on the remote server and store if necessary
z	prerelease mode

The default setting for the /f and /l switches is *on*; all other switches are *off*. You can change the default setting of any switch by using a global switch. Any switch given with no file name (i.e., just a slash and switches) establishes the default setting for that switch. Unless overridden, that default applies to following commands.

Fine point: When the /f (flip CameFrom) switch is on, SModel will convert a CameFrom clause back to a ReleaseAs clause. This makes it easier to use a DF file that was generated

by the ReleaseTool as a starting point for a new "working" DF file after a release. This switch generally has only a minor effect on the use of DF files.

5.2 SModel limitations

Each DF file processed by SModel must contain no more then 450 files. This applies to each Included DF file as well. This number may change. Contact the implementors if there is a question.

6 VerifyDF

VerifyDF attempts to answer the question: Does this DF file have entries for all the files I need to rebuild my program, and are these files consistent? VerifyDF scans a DF file looking for "end result" bcd and boot files. These are the files marked with a "+" before their names (a DF file can have more than one "end result"). VerifyDF will analyze **each** of these files to determine what files were needed to build it and will compare the needed files against entries in the DF file. If a needed file is not in the DF file, VerifyDF will give an error message. Also, VerifyDF will give an error message if a needed file is listed in a different version.

After checking the "end result" files, VerifyDF recursively analyzes the files they need. This process continues until all files in the closure of dependencies, except for imported (and missing) files, have been analyzed.

For example, to verify Compare.df, type

```
>VerifyDF Compare.df
```

Any files that are missing from the DF file are listed with the create dates and remote location (gotten from the RemoteName leader page property) of files on the local disk. This can help, for example, to identify DF files from which some of those files should be imported. VerifyDF also prints out files listed in the DF file that appear to be unnecessary. These might include such files as command files and signals listings, but they might also include imports that are no longer necessary. If those files are actually necessary, such as the command files, you can suppress these warnings by marking these files with a '+'. for example:

```
Directory [MyHost]<MyDir>Private          ReleaseAs [RelesaseHost]<RelDir>Private>

  +Source.MyStuff!1        14-Dec-60 16:23:27 PDT
  +ReBuildMyStuff.cm!1     14-Dec-60 16:25:53 PDT
```

VerifyDF also checks for certain common mistakes, such as files on a directory that are released onto a directory with a different >Public, >Friends, or >Private suffix. For example,

```
Directory [Igor]<Emerson>Compare>Public  ReleaseAs [Idun]<Apilot>Compare>Private
```

is probably a mistake, since Public is not the same as Private.

VerifyDF will look on remote file servers for the correct versions of files if they are not local. So, the files described by a DF file do not have to be on the local disk for VerifyDF to do its job. However, since this remote checking must currently be done with a pseudo page-level access protocol, it can be relatively slow. The DF file itself also does not have to be on the local disk. For example, the following can be used to check a remote version of Compare.df:

```
>VerifyDF [Igor]<Emerson>DF>Compare.df
```

When processing a DF file, VerifyDF may have to retrieve imported or included DF files from a remote server. Unless the /t (fetch to temporary files) switch is off, these DF files will be retrieved to temporary files. This avoids cluttering your disk with DF files you may not want.

6.1 VerifyDF's command line

The VerifyDF command line has the form

```
>VerifyDF [/<global switches>] DFfile1[/<local switches>] ... DFfileN[/<local switches>]
```

Global switches are optional and control the verification of subsequent DF files. You can also set global switches by giving an empty DF file name.

6.1.1 VerifyDF switches

A switch specification is a letter, optionally preceded by a '-' or '~' to reverse the sense of that switch. The valid switches are:

f print "flattened" DF file (all Imports and Includes structure removed)
n check that all files seem necessary (*default*) You would probably want this when using the /f switch
t retrieve DF files to temporary files (*default*)

The default setting for the /n and /t switches is *on*, while the /f switch is *off*. You can change the default setting of any switch by using a global switch. Any switch given with no file name (i.e., just a slash and switches) establishes the default setting for that switch. Unless overridden, that default applies to following commands.

6.2 VerifyDF limitations

The total number of files that VerifyDF can check, *including* those from imported and included DF files, is 1000.

7 DFDelete

When you have finished working on a DF file and have SModel'ed its files out to their remote locations, you can free up space on your local disk by running DFDelete on the DF file. This program scans a DF file (and the ones it Includes), and generates a command in Line.cm that can be used to delete the files described by the DF file. Deleting these files is safe because you can be certain, after running VerifyDF and SModel on a DF file, that all needed files have been stored remotely.

DFDelete will *not* add to the delete command any file on the local disk that has a create date *different* than that listed for it in the DF file.

7.1 DFDelete's command line

The DFDelete command line has the form

>DFDelete [/<global switches>] DFfile$_1$[/<local switches>] ... DFfile$_n$[/<local switches>]

As usual, global switches are optional and control the deletion of following DF files. You can also set global switches by giving an empty DF file name.

7.1.1 DFDelete switches

DFDelete has only one switch which can be preceded by a '-' or '~' to reverse its sense:

r also delete Imported and ReadOnly files

7.2 DFDelete limitations

Each DF file processed by DFDelete must contain no more then 450 files. This applies to each Included DF file as well.

8 DFSubstitute

Although a DF file is just a text file that can be edited by the user, it is still awkward to make simple repetitive changes to large numbers of DF files. The program DFSubstitute can be used to simplify this task. It can:

- change hosts or directories,
- move an Imported file (e.g. Heap.bcd) from one DF file (PilotPublic.df) to another (ComSoftPublic.df), and
- insert or delete Imported files.

DFSubstitute makes changes to a set of DF files according to commands in a substitution script file. The commands are executed in order from first to last for each line in a DF file. This means that later commands can take advantage of the substitutions made by previous commands. Included DF files are processed in the usual bottom-up recursive fashion. The rewritten DF files are not stored remotely by DFSubstitute; you must use SModel to do that.

The four DFSubstitute commands are:

- Rename [RHS] <loc$_1$> To <loc$_2$>

Rename changes the remote location on the left hand side (Directory, Imports, Includes, or ReadOnly) of matching DF file lines If RHS is specified, matching right hand sides (ReleaseAs or CameFrom) are changed. Each <loc> can be a host (e.g. [Igor]), a directory

(e.g. <Emerson>Tajo>), a file name (e.g. **MFile.bcd**), or a combination of all three. For example, to rename [Igor] to [Idun] in all left hand sides, use the following:

```
Rename [Igor] To [Idun]
```

To change all r.h.s. references to [Igor]<Ramona> to be [Idun]<Int>, use

```
Rename RHS [Igor]<Ramona> To [Idun]<Int>
```

It is also possible to change just a subdirectory, e.g.

```
Rename CWF>Public To Other>Public
```

To change the location of just one file, use a command like the following:

```
Rename [Igor]<Emerson>Tajo>Private>NSFileTransfersA.bcd
    To [Igor]<Emerson>NSFileTransfer>Friends>NSFileTransfersA.bcd
```

- Move Import <name> From <DF file$_1$> To <DF file$_2$>

This moves an import from the Using list of one DF file to another. If no Using list files remain the first Imports line is entirely deleted. For example,

```
Move Import Heap.bcd From Pilot.df To ComSoftPublic.df
```

- Delete Import <name> From <DF file>

This just removes the specified import from the Using list of an imported DF file. If no **Using list** files remain the entire Imports line is deleted. For example, to remove all importations of Space.bcd from PilotPublic.df, use

```
Delete Import Space.bcd From PilotPublic df
```

- Insert Import <name> From <DF file>

This simply adds an import to the DF file's Using list, e.g.

```
Insert Import Environment.bcd From MesaPublic df
```

8.1 DFSubstitute's command line

The DFSubstitute command line has the form

```
>DFSubstitute ScriptFile DFfile1 ... DFfilen
```

The first file is a substitution script (default extension ".script") that specifies the changes to be made to the following DF files. DFSubstitute has no switches.

8.2 DFSubstitute limitations

Each DF file processed by DFSubstitute must contain no more then 600 files. This applies to each Included DF file as well.

Important!
If there are *spaces* embedded in a token, please quote them. For example:

```
Rename "|Walter:Very Nice:Music|<Carlos>" To "|Wendy:Very Nice:Music|<Carlos>"
```

9 DFDisk

DFDisk produces a file "Disk.df" that describes the current search path. With the exception of "$" files and a few kinds of log files, it lists all files on the search path with the create date and remote location found on the local disk. The remote location is taken from the RemoteName property in each file's leader page. DFDisk is most useful when you are trying to find the remote location for files, or when you are trying to save all your files before reformatting the volume. It can also tell you about new files that should be recorded in a DF file, since the RemoteName property for these files will not have been set and they will be listed under the remote "location" [Unknown]<Unknown>.

9.1 DFDisk's command line

DFDisk has no switches, and its command line is simply

```
>DFDisk
```

9.2 DFDisk limitations

The maximum number of files that DFDisk can process is 1000.

10 DFTool

The DFTool provides a window interface to the other DF programs. It supports BringOver, SModel, VerifyDF, DFDelete, DFDisk, DFSubstitute, as well as program librarian CheckOut and Query. Since the different commands share several DF implementation modules, fewer resources are used by this tool than by the separate DF programs. The price you pay for this is that only one command can be run at a time as opposed to having multiple Executive windows the run the DF programs from. There is currently no command line interface to the different commands.

The DFTool communicates through four subwindows: a message, form, command, and TTY subwindow. The TTY subwindow is used to log the progress of each command, and for interaction with the user (e.g. for passwords and for file transfer confirmations). There is also an Options window which is used to set infrequently modified parameters.

fine point: A picture of the tool will be supplied here eventually.

10.1 Form subwindow

The fields that can be used as arguments to a command are listed in the form subwindow. The first row has a five Booleans that correspond to the most widely used DF program command line switches. The next four rows are string items that provide parameters for the DF commands.

Booleans:

Confirm means ask for confirmation before retrieving or storing a file. It has the same effect as /-a on the BringOver and SModel command lines. The default is **TRUE**.

GetOnlyExports means retrieve only files marked as exports when bringing over a DF file. This is the same as the BringOver /p switch. The default is **FALSE**.

VerifyFilesExist means check that files exist in the right place and version. It is the same as the BringOver /v switch. The default is **FALSE**.

DontStoreRemotely means never store files remotely. This is equivalent to SModel's /n switch. The default is **FALSE**.

VerifyRemoteOnStore when doing an **SModel!**, means verify that files are really on the remote server and store if necessary. This is the same as the SModel /v switch.The default is **FALSE**.

Fill-ins:

DF Files: are the names of the DF files to be used for any commands. If a DF file's name contains spaces (e.g. a fully qualified DF file name on an NS server), the name must be enclosed in double-quotes (").

Files: is a list of files (separated by spaces) for the next command to act upon. These might be, for example, the specific files that **BringOver!** should retrieve from a DF file (BringOver only files mode).

LocalDir: means that **BringOver!** should do all retrievals to this directory. This is equivalent to BringOver's localDir/c command. If the directory is not a complete path name, i.e. it does not begin with <, it is assumed to have a <> prepended.

Checkout Reason: is given to the program librarian when files are checked out.

10.2 DFTool command subwindow

The fields in the command subwindow are the following:

CheckOut!	checks out the libjects for the DF files listed in the **DF Files:** line and the files listed in the **Files:** line. The **CheckOut Reason:** is passed to the program librarian.
Query!	displays information regarding the libjects for the DF files listed in the **DF Files:** line and the files listed in the **Files:** line.
BringOver!	invokes the BringOver algorithm on the DF files listed in the **DF Files:** entry. If any files are also listed in the **Files:** line, BringOver will enter only files mode and retrieve just those files.
SModel!	stores back the DF files and its components within of the DF files listed in the **DF Files:** entry.
VerifyDF!	verifies the completeness and consistency of each DF file named in the **DF Files:** entry.
DFDelete!	invokes the DFDelete algorithm on each DF file listed in the **DF Files:** line.
DFDisk!	generates a file "Disk.df" that describes all the files on the current search path.
DFSubstitute!	modifies DF files listed in the **DF Files:** entry according to the substitution script file named in the **Files:** entry..
Options!	creates an options window for the DFTool if one does not already exist.

10.3 DFTool Options window

The Options window is created by the **Options!** command. It contains a string item and Booleans that govern the DFTool's operation, but which are typically changed only infrequently. The string item, **LocalDFDir:**, is initialized from the **User.cm LocalDFDir:** entry. The Booleans correspond directly to the command line switches for each DF program. After changing the options, invoke **Apply!** to invoke those changes. The **Abort** command will restore the options to what they were before the **Options!** command was invoked. Both **Apply!** and **Abort!** perform the appropriate actions and then destroy the Options window.

10.4 DFTool User.cm fields

The DFTool uses the same |DFTool| section in the User.cm as the other DF programs. Besides the fields described above in Section 3, the standard **InitialState:**, **TinyPlace:**, and **WindowBox:** entries can be set.

11 The IncludeChecker and DF files

If you are a DF software user (i.e. if you have a |DFTool| section in your User.cm), the released Klamath IncludeChecker can process a DF file as well as a lists of files. The

IncludeChecker has a **df/c** command that is used to specify the DF file to check. For example,

```
>IncludeChecker MyPackage.list/cio df/c MyPackage.df
```

will analyze the files described by MyPackage.df and generate an includes and includedBy listing in MyPackage.list and a rebuild command in Line.cm.

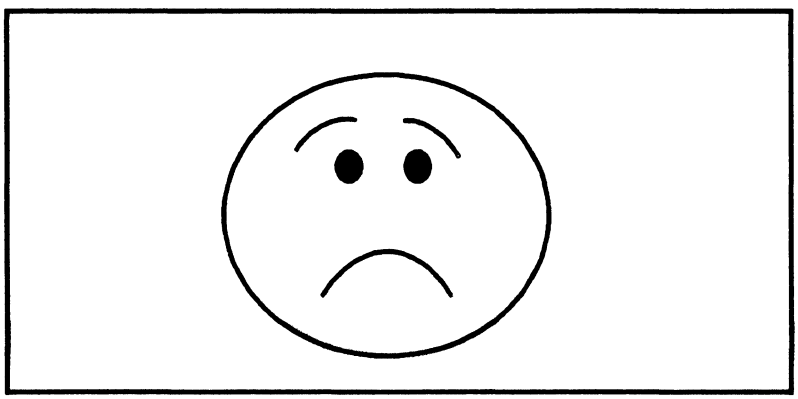
Each file listed in the DF file is looked up on the local disk. If there, that version is analyzed regardless of its create date. If the file is not local, the remote version is checked (the remote path is gotten from the DF file). Because the IncludeChecker believes that the local versions of files are the "truth" (the assumption is that the DF file was brought over and some changes were made to its files), it can be used to verify a component that has not yet been SModel'ed.

Note: *VerifyDF operates differently:* VerifyDF checks the particular snapshot of a package described by a DF file. Local versions of files with different create dates are ignored and the remote versions are used instead. This means, in general, that a DF file has to be SModel'ed before VerifyDF can be run on it.

The IncludeChecker also has a /r (examine DF imports) command line switch that may be specified when DF files are processed. When set, the IncludeChecker will also analyze imported files. If you believe that no imported file has changed since you brought over the DF file, you can use /-r to reduce the IncludeChecker's running time. The initial default for the /r switch is *on*. In general, you should not use /-r.

If you have a [DFTool] section in your User.cm, some additional parameters and an additional command appear in the IncludeChecker window. The **DF File:** entry names the DF file to be processed when the **Check DF!** command is invoked. The Boolean **Examine DF Imports** appears in the IncludeChecker Options window and has the same effect as the /r command line switch; it is initially **TRUE**.

12 Dealing with Problems



As much as we try to avoid them, problems still crop up. Here are some common problems and ways you could deal with them.

Network problems

Problem: You did a BringOver, modified files, and do a Smodel. Unfortunately, you lost a connection in the middle of a Smodel, so only some of the files were stored.

Solution: When the network is up, do a Smodel with the "verify files exists" mode to make sure **all** the files are stored back.

You forgot to to check out the Libject.

Problem: Shame on you. Smodel probably gave you a warning. Using the "don't check librarian" switch is probably dangerous since someone else may have checked out the DF. and has worked on the program.

Solution: You should try to check out the libject. There are two cases:

If you are denied access, go talk to the person who checked out the libject and try to coordinate the modifications. If you changes do not overlap, you are lucky.

If are you given access to the libject, you may still be in trouble since someone else may have done a checkout and a checkin during the period you were working on the component. Find out the checkin date of the libject and see if that occured during your after you did a BringOver.

The Librarian is down.

Problem: Smodel fails since the libject cannot be checked in.

Solution: You are safe since no one else can modify the files you have checked out. Frequently, you will only do a checkout of the DF file itself. In that case, the files listed in the DF file may have been stored back already, leaving only the DF file "un-stored". However, the DF file is already modified with the new create dates, including that of the DF file itself. Thus a subsequent Smodel (done when the librarian is up) will use the local DF file and will believe that all the required files are remotely stored. When the librarian is up, do a Smodel with the "verify files exists" mode to store the DF file back.

The programs tell me that it can't parse dates in my DF

Problem: Your time zone requires you to specify time relative to GMT. Some parts of the world require you to specify the time in the format of hh:mm:ss +N GMT. The released parser is not able to parse that correctly. This is a limitation in the Klamath version .

Solution: Please wait for the announcement of the newer, better, and more worldly DF software. The new parser should be able to handle various time formats.

I want to move the files pointed to by a DF from one location to another.

Problem: You want to change the "Directory" statements and move the files to their new destinations..

Solution: do a BringOver, run a DFSubstitute, and do a Smodel.

How do I stop the DF Tool?

Problem: You made filled in the wrong parameters, or perhaps the made some other mistakes, and you want to stop the operations.

Solution: Press the STOP key, and keep on trying until it stops.

Ethernet monitors

Filed as [McKinley:OSBU North]<PilotProse>11.0>Communication Tools>Ethernet monitors
last edited by AOF, October 18, 1984 3:45 PM

The tools described in this document are part of a collection of local ethernet monitoring facilities implemented by the same people that implemented the low level system software local area network code. Being such they have certain characteristics that are important for the casual user to understand.

First, they are non-released tools. That implies that they are unsupported, at least in the formal sense. They are, in fact, not only supported in an informal manner, but may be the subject of ARs. Users are encouraged to submit suggestions as to the applicability and correctness of the current tools and suggest enhancements and the like. The primary reson for the tools not being released is because of what you, the reader, are being subjected to at this very moment; the documentation. It will be found to be lacking.

Second, these tools are local network monitoring tools. They monitor traffic, usually in conjunction with the ethernet medium only, and they act as third party observers. Due to the architecture of the ethernet, the reliability of the information gathered is always in doubt. The user must be willing to accept the output as a hint, never believe he has observed the truth. They are monitors, i.e., they never insert traffic into the network.

1 Structure

All of the tools described in this document are instantiations of *spies*. They use a Friends level communication interface (**SpecialCommunication**) to get low level access to the packets observed by the host machine and to set that host' ethernet controller mode to *promiscuous*.

By running in promiscuous mode, the host machine is capable of observing all the traffic on the local ethernet. In doing so, it loses its capability to do address descrimination in microcode. Doing address descrimination in Mesa causes severe performance degradation. Users of these tools should be aware that the host machine's own communication performance may be decreased. All of the described tools will return the state of the host machine to normal when the tools are deactivated.

Multiple spy tools may be run at a time. Each tool will be afforded the opertunity to look at the packets and perform its own pre-defined function with regard to the packet. When a tool is deactivated (or stopped via a tool interface command), *all* spy tools that where running will be disabled.

Fine point: Care must be used when using tools that accept packets with bad ethernet recieve status. Since most spy tools are prepared to handle only well formed packets, tools that permit them to be inserted into the system must be run *last*. That tool will, it is assumed, not permit a bad packet to be inserted into the system or be observed by another spy proc.

1.1 Ethergraph

Ethergraph was designed to answer the question, "How much traffic is there on the local network?" It attempts to do this by displaying a realtime plot of the traffic, the plot being one pixel wide and having a height proportional to the average percent of medium bandwidth utilized during a user specified interval. All traffic is measured with no effort to characterize its type, source, destination or intent.

Reset! Resetting the display cases all previous information to be cleared and the **Time a left edge** to be set to the current system time.

Full deflection is: {100% | 10% | 1%}
The height of displayed rectangles is adjustable by factors of ten. If full deflection is set to 10% for instance, a display rectangle ten units high would represent 10% of the total ethernet badwidth.

Period: {seconds | minutes | hours}
The tool will observe traffic and display one rectangle per interval which is the average rate of traffic for that interval.

Source: {10 Mbit | 9.6 Kbit | 4.8 Kbit | 2.4 Kbit | 1.2 Kbit | 300 bps}
It was intended that this tool be capable of displaying traffic observed from a medium and plot the utilization of that medium proportional to its full capability. This has only been implemented for the 10 MBit ethernet.

Time at left edge: <dd-mmm-yy hh:mm:ss>
Whenever the display plots at the left edge of the screen, whether it be when the tool is first started, restarted, **Reset!** or wraps around from the right hand edge, the tool records the system time. This can be used to estimate the time of day that an interesting event took place in the display.

1.2 HostGraph

Once the amount of traffic present on the network is determined, it them becomes interesting to find out what machines are generating that traffic. **HostGraph** was written to do just that. It observes local traffic and displays the 48-bit host number of the machines transmitting on the ethernet. It has a limit of 20 addresses it will display. It normally selects the first 19 of them by being *first observed*. When the first 19 slots are filled and packets from some other machine is encountered, that is lumped into the 20th slot and noted with a broadcast host number. Once in the display table, the host number and the amount of traffic it transmists will be depicted, updating once per

second. The display is in the form of a horizontal bar chart. The amount of traffic observed in the last second is appended to the right side of the bar in black, while the previous observations are shown in grey-shade. This gives the illusion of moving from left to right and conveys some sort of feeling of the amount of traffic being inserted into the network by a particular machine. The amount of left to right growth is proportial to the bandwidth of the medium it consumed. When the bar reaches the right-hand side of the display, it is reset to the left side.

Once a machine is registered in the table of 20, it will remain there until the entire display is **Reset!**, the machine number is **Removed!**'d or the tool is deactivated.

A particular machine may be entered into the list, providing there is an empty slot, by use of the **Include!** command. Since the tool is always trying to keep the table full and will insert the number of the first machine not in the list when traffic is first incounted from that machine, the tool may have to be stopped

Stop|Go! — Bugging the **Stop|Go!** command will toggle the active state of the tool's spy proc. Since the tools always fills empty slots in its 20 element array with machines observed generating traffic on the ethernet, it is sometimes necessary to stop the spy process in order to **Remove!** then **Include!** a machine number of particular interest.

Reset! — Resetting the display cases all previous information to be cleared.

Remove! — Any machine number in the list may be removed from that list by entering the number in the **Host:** field, then bugging **Remove!**. Machine numbers may be **Removed!**'d when to tool's spy proc is active (see **Stop|Go!**). The slot in the table occupied then becomes empty, and if the spy proc's state is active, will be filled by the first machine not in the list generating traffic. If the host number has an improper format, cannot be found in the clearinghouse or is not in the current host number array, the tool will *blink* the display.

Include! — If the user wishes to observe a particular machine, that machine's host number may be entered into the display table manually by typing the machine's host number or clearinghouse name into the **Host:** field, then bugging **Include!**. If the **Host:** specification is incorrect, the name cannot be found in the clearinghouse or there is no empty slot in the display table, the tool will blink the display.

Host: — In order to use the **Remove!** or **Include!** commands, the user must first specifiy a host number. The specificaion is made in AddressTranslation format, which carries with it the implication that either numeric values as well as clearinghouse names are accepted. If numeric values are used, only the machine number need be present, i.e., ".25200001130.".

1.3 EtherIOGraph

This tool further characterizes the ethernet traffic by permitting the use to specifiy a particular machine to observe. The traffic being directed to that machine is then depicted as a realtime graph much like **EtherGraph**'s, but eminating down from a center axis. Traffic inserted into the network by the machine in question eminates upward from the center axis.

Reset! Resetting the display causes all previous information to be cleared and the **Time a left edge** to be set to the current system time.

Full deflection is:{50% | 5% | 0.5% | 0.05%}
The height of displayed rectangles is adjustable by factors of ten. If full deflection is set to 10% for instance, a display rectangle ten units high would represent 500 KBits per second (average) of ethernet traffic.

Period:{seconds | minutes | hours}
The tool will observe traffic and display one rectangle per interval which is the average rate of medium badwidth utilized during that interval.

Interpret Address! After the user has entered an appropriate value in the **Observer only** field, this command must be bugged to prompt the tool the interpret that field. If the interpretation fails, the tool will *blink* the display.

Observer only: <address translation>
This is the field interpreted by **Interpret Address!**. It must be in acceptable AddressTranslation format.

watching: {input | output | both}
It is possible (though there is little advantagein it) to observe on the traffic destined to or the traffic originating from the machine in question by selecting the desired value for this enumerated type.

Time at left edge: <dd-mmm-yy hh:mm:ss>
Whenever the display plots at the left edge of the screen, whether it be when the tool is first started, restarted, **Reset!** or wraps around from the right hand edge, the tool records the system time. This can be used to estimate the time of day that an interesting event took place in the display.

1.3 EtherMonitor

EtherMonitor is a very simple tool that collects statistics about the type of traffic observed on the local ethernet. The satistics are in the form of simple counters, with one excpetion, the average number of packets per seconds.

Caution: This tool modifies the logic of the ethernet driver such that it is willing to accept packets received with bad receive status. If this tool is run in conjunction with other peek tools, it should be the last one run so it has the opportunity to filter the bad packets from

the input queue. Other peek tools and the system in general are not prepared to handle packets of this nature.

Garbage! Bugging the **Garbage** command causes the tool to print out a recap of the running totals, including the duration of the statistics gathering session, the number of packets with error status and the average number of packets observed per second.

Broadcasts! Bugging this command will cause the tool to display the number of broadcast packets observed for each well-known packet type. It counts both XNS and Pup packets. Since broadcasts are received by all machines on the network, it is sometimes interesting to characterize the type of client that uses broadcasts.

Packet types! **Packet types** displays the number of packts observed of each level-2 packet type, whether broadcasted or directed. This is sometimes useful in determining the nature of the traffic on the network.

Sockets! Bugging the **Socket** command displays the number of XNS packets that have been broadcasted to *well-known sockets*. Like **Broadcasts** it is used to characterize the type of network client that uses broadcasts in an effort to minimize their use.

1.4 GatewayLoad

GatewayLoad is a tool used to monitor the load on a gateway, based on the source and destination networks of the packets the gateway is processing. The tool monitors both pup and ns packets.

The tool, when active, displays a window with two subwindows, one for displaying the load histogram and one containing the commands and parameters available to the user.

1.4.1 Graph subwindow

The tool's top subwindow is the graph window that displays a realtime histogram of the observed load.

After the **Start** command is selected the tool begins to collect packets whose immediate source or destination is the specified gateway. An entry is made in the table of networks (and displayed on the left side of the graph subwindow) based on the networks in the packet. The realtime histogram for that entry is then started. When the limited table fills up, any new network pairs observed are added to the histogram bar labeled in the window as all ones.

The grey section of each bar indicates the amount of traffic observed since the tool was last cleared - the black section indicates the amount of traffic observed in the last second. As each histogram bar reachs the right side of the window, it will wrap.

1.4.2 Command and parameter subwindow

Start

The **Start** command starts the monitoring process and the graphing of the histogram. The old histogram (if any) and the table of networks is not modified.

Stop

The **Stop** command stops the monitoring process and the histogram.

Clear

The **Clear** command clears the graph window and the table of networks. Since **Start** does not clear any of the old tool state (i.e., the monitoring process is stopped but the table and the histogram are left intact), **Start** may be used in conjunction with **Clear** to begin a new monitoring session.

Reset

The **Reset** command resets the histogram only. Because it does not clear the table of networks, it is useful for restarting the histogram after the user has modified the table with **Include** and/or **Remove**.

Remove

The **Remove** command takes the network pair specified in Net entry and deletes it from the table of networks. If this command is selected while the tool is **Start**'d, the entry will probably be quickly filled by the next packet arriving with a new network pair. Should the user desire to **Remove** an entry and replace it with a specific network pair, he should first **Stop** the tool, then **Remove** the unwanted pair and use the **Include** command to enter the network pair of his choice. A blink of the display indicates that hte specified network pair was not found in the table of networks.

Include

The **Include** command puts the network pair specified in **Net entry** into the first available slot in the networks table. Should there be no empty slots in the table or if the specified network pair is not in the proper format, the display will blink.

Histogram mode

The **Histogram mode** is an enumerated item that determines whether the load in bytes or the load in packets will be plotted.

Net entry

Net entry is the network pair that is to be removed or included. The format is *net-net* and the order of the nets does not matter.

Target host

Target host is the machine the user wishes to monitor. This parameter may either be either an actual host number or a clearinghouse name. A blink of the display indicates an invalid number or name. If this parameter is left blank, the tool will watch all hosts on the net (not very useful).

Flush old

The **Flush old** command clears the table of networks and the histogram of all network pairs that have not seen traffic for the number of seconds specified in the **seconds** parameter.

seconds

seconds is the number of seconds to be used when deleting old network pairs from the tool with the **Flush old** command.

Log

Log is a boolean item that is used to display a log file containing statistics gathered during the collection of the histogram information. The log reports the time started and the time of te currently displayed information. For each network pair, the log displays the number of bytes observed, the number of packets observed and how long it has been since traffic was last observed.

– – ExecCommands.doc

ExecCommands registers an arbitrary number of Exec commands that expand to text given in User.cm. The commands can take arguments that get substituted into the expansion. The expansion can also specify that some or all of the text is to be repeated, with different arguments being substituted within each repetition.

When ExecCommands is run, it registers an Exec command called ExecCommands.~. Invoking this command causes it to read User.cm and register any commands found there, as well as unregistering any commands previously registered. NOTE: Due to a bug in the Exec, RemoveCommand crashes if the command being removed has had its name changed. Therefore, don't use ChangeCommandName to muck with commands registered by ExecCommands; instead, change the command's name in User.cm and use ExecCommands.~ to unregister the old name and register the new one.

The commands to be registered are taken from the [ExecCommands] section of User.cm; as usual, this section may be volume – specific (via [CoPilot:ExecCommands], [CoCoPilot:ExecCommands], etc.). Each User.cm entry has the format:

command: [options] "expansion" [options]

That is, the entry consists of the Exec command name, a colon, and then some quoted text representing the expansion, optionally preceded or followed by other stuff. The "other stuff" specifies how many arguments the command requires. If you don't specify anything, then the command will accept any number of arguments. The options are:

<nn	The command will not accept more than nn – 1 arguments.
>nn	The command will require at least nn + 1 arguments.
=nn	The command will require exactly nn arguments.

If you specify an inconsistent set of options, ExecCommands will decide something reasonable. In general, if you provide bogus User.cm entries, ExecCommands will not complain but will try to do something reasonable without crashing.

Within the expansion, certain sequences of characters have special interpretations. All such special sequences begin with either \ or @:

\n, \r, \t, \b, \f, and \l, and their uppercase forms, are treated as in Mesa 10.0. \ followed by any other character yields that character. Thus \\ yields a single \, and so on. Note, however, that the Token interface doesn't know about \, so it will not let you use it to get a quote – mark into your string ("....\"...."). You can use apostrophes around the expansion instead of quotes if you want quotes inside the string. If you want both inside the string, you're out of luck.

@N yields the Nth argument, where the first argument is N = 0. N is a string of digits. If you want to have a digit immediately following an argument in the expansion, use @N@; the second @ is ignored other than to mark the end of the number N.

@M[xxxxx@N] , where M and N are strings of digits, causes the text xxxxx to be repeated N – M + 1 times, once for each argument from M through N. The iteration is performed all N – M + 1 times even if there aren't that many arguments given to the command. Within the text being repeated, the string @* represents the current argument of the iteration. For consistency, @*@ can also be used; the second @ is ignored as above. It is not permitted to have an iteration within an iteration.

@M[xxxxx@] is the same as @M[xxxxx@N], except N is assumed to be the number of the last argument, i.e., one less than the number of arguments given to the command when it was invoked.

@ followed by any other character yields that character, just like \. Thus @@ provides a single @ character, as does \@.

Within the expansion, tabs and spaces are ignored if they immediately follow a CR (explicit or via \n). If you really want white space you must use \ or @ to "quote" the first such character after the CR.

As an example, here are the ExecCommands from an User.cm:

```
[CoPilot:ExecCommands]
BigPrint: >0 "@0[Formatter @*/h – t – i\nPrint @*.press/s1\nDelete @*.press\n@]"
FinePrint: >0 "@0[Formatter @*/k – t – i\nPrint @*.press/s2\nDelete @*.press\n@]"
NotePrint: >0 "Print Laurel10/f@0[ <>Notes>@*/p1t12s2 – z@]"
NewProg: >0 "Open Library/W\n@0[Zap <Library>Progs>@*.bcd
          Copy <Library>Progs>@*.bcd ← <>Temp>@*.bcd
          Copy <Library>Source>@*.mesa ← <>Temp>@*.mesa\n@]Open Library"
OldProg: =1 "Copy <>Temp>@0.mesa ← <Library>Source>@0.mesa
          Copy <>Temp>@0.bcd ← <Library>Progs>@0.bcd"
PutHack: =1 "Ftp ServerName connect/c DirectoryName dir/c <DirectoryName>Tools> sto/c @0.bcd ↑
```

-- ExpandType.doc

ExpandType combines the debugger's ShowType with the editor's abbreviation expansion. The idea was stolen from Cedar.

What it does

ExpandType works just like the normal abbreviation expansion, except if it can't find the abbreviation in the dictionary, it does a ShowType! It uses the word right before the insertion point, the "." before it and the word before that. (e.g. if I type "Display.Black", then EXPAND, it looks up Display.Black, just like ShowType).

It does two kinds of expansion, ExpandTypeWithFields and ExpandType.

"ExpandTypeWithFields" is fancier. If the type being expanded is a PROCEDURE, it constructs a complete call to that procedure and inserts "‖" and "‖" around all the parameter TYPEs, like so:

```
I type "Display.Black", then EXPAND.
I get "Display.Black [window: ‖Display.Handle‖, box: ‖Window.Box‖]"
```

The editor automatically selects the first field for you. Now you just NEXT through the parameters, filling in values, or deleting defaulted ones. If the procedure has RETURN parameters, the expansion looks like this:

```
I type "Window.GetBox", then EXPAND.
I get "‖Window.Box‖ ← Window.GetBox [‖Window.Handle‖]"
```

This may not always be what you want, but what the heck...

"ExpandType" simply spits out the results of the ShowType:

```
I type "Display.Black", then EXPAND.
I get "Display.Black: PROCEDURE [window: Display.Handle, box: Window.Box];"
```

This is really just sortof an accelerator for ShowType; you can do a ShowType without going to the debugger.

One more little feature: if you ExpandTypeWithFields on a "TYPE = PROCEDURE...", it will spit out the definition with "<<>>" around it:

```
I type "TIP.NotifyProc", then EXPAND.
I get "<< = PROCEDURE [window: Window.Handle, results: TIP.Results]; >>"
```

How to run it

ExpandType registers two new ATOMs with the editor, "ExpandType" and "ExpandTypeWithFields". To use it, just edit your favorite TIP table expand productions, substituting "ExpandType" or "ExpandTypeWithFields" for "ExpandAbbreviation", then run ExpandType.bcd (you might have to reboot depending on what TIP table you edit). For example, if you use the EXPAND key to do expansions, then edit Tajo.TIP as follows:

```
EXPAND Down = > [IfShift,DefineAbbreviation,ExpandTypeWithFields];

or

EXPAND Down = > [IfShift,DefineAbbreviation,ExpandType];
```

If you like to use Shift – Space as your expansion keys, then edit Expand.TIP in a similar fashion. (If you're not familiar with Expand.TIP, see MesaDict.doc.) An example:

```
SELECT TRIGGER FROM

   Space Down = >
      SELECT ENABLE FROM
         LeftShift Down = > ExpandTypeWithFields;
         RightShift Down = > ExpandType;
         ENDCASE = > CHAR;

ENDCASE...
```

-- ExcludeHeapOwners.doc

Are you tired of using DebugHeap and finding nodes owned by modules like: NSStringImpl, NSNameImpl and SvPack (not the real node owners but, services)? Well, now there is a hack that can be given a list of module names to exclude as node owner and the clients that call them will be saved in their place.

To use this hack, bind your client boot file using HeapCheckPack.bcd. The instructions for using HeapCheckPack can be found in HeapCheckPack.doc

Boot your boot file with the '6 switch. When it has booted swat to CoPilot and, in the Executive, run ExcludeHeapOwners. The syntax is:

ExcludeHeapOwners module1 module2 module3 ... moduleN

Then, in the Debugger, Proceed. From that point on, each module excluded will have it's client saved as the node owner, instead of itself. If that client is also in the excluded list, then up the call stack we go looking for a non – excluded client to save as the node owner.

When trying to use DebugHeap on a heap that HeapCheckPack has created, it is necessary to extract the UNCOUNTED ZONE from the HeapCheckPack handle. To do this, use the following LOOPHOLE:

LOOPHOLE[<HeapCheckPack handle>, HeapCheckPack.UncountedZoneRep]↑.underlying

Restrictions:
ExclueHeapOwners REPLACES any previous list of modules with the new list.
Maximum number of modules to exclude must be < = 256.
Only works on heaps that HeapCheckPack intercepts.

-- FastMouse.doc

FastMouse is a hack to change the delta – mouse/ delta – cursor relationship. Thus, you can use it to let a small mouse movement sweep the cursor all the way across your display. It registers five ExecProcs.

"Threshold" refers to the delta mouse which must occur over 1/60 second before the standard mouse behavior changes. RaiseThreshold.~ sets threshold ← threshold + MAX[threshold/10, 1]. LowerThreshold.~ sets IF threshold > 0 THEN threshold ← threshold – MAX[threshold/10, 1] LowerAmplification.~

"Amplification" = 10 times the change factor. Thus, standard behavior = 10. RaiseAmplification.~ sets IF ampxTen < LAST[NATURAL] THEN ampxTen ← ampxTen + MAX[ampxTen/10, 1]. LowerAmplification.~ sets IF ampxTen > 0 THEN ampxTen ← ampxTen – MAX[ampxTen/10, 1].

"ShowMouseParms.~" lists the current Threshold and Amplification in the Herald window. I find Amplification = 1.8, Threshold = 14 to be optimal. Amplification and Threshold may be specified in User.cm as follows:

```
[FastMouse]
AmpxTen: 18
Threshold: 14
```

```
        dir/c <DirectoryName>Source> sto/c @0.mesa ↑
        dir/c <DirectoryName>Doc> sto/c @0.doc"
ReplaceHack: =1 "Ftp ServerName dir/c <DirectoryName>Tools> sto/c @0.bcd del/c @0.bcd ↑
        dir/c <DirectoryName>Source> sto/c @0.mesa del/c @0.mesa ↑
        dir/c <DirectoryName>Doc> sto/c @0.doc del/c @0.doc"
Use: "SetSearchPath <>Temp@0[ @*@@] <>Mail <Library>Defs <Library>Progs <>"
Star: =1 "Use.~ Star@0>Defs Star@0>Misc"
Bye: =0 "Delete <>Notes>*$ <>Notes>* -TOC
        Ftp ServerName sto/>a <>Mail>*.mail <>Notes>* ↑
          localdir/c <>Mail>Phictionary> dir/c <DirectoryName>Phictionary> ↑
          sto/>a Archive.mail Scores Words.used ↑
          localdir/c <> dir/c <DirectoryName>CMs> ↑
          sto/>a MakeLib.cm User.cm Default.dict ↑
          localdir/c <Library>Progs> dir/c <DirectoryName>Pilot>Tools ↑
          sto/>a TinyWindow.icons ↑
          localdir/c <>TIP> dir/c <DirectoryName>TIP> ↑
          sto/>a ExtendEdit.tip FormSW.tip KeyHacks.tip KeyJump.tip TinyPictures.tip
        Chat ServerName/h <>Mail>ChatCommands/f
        Poly"

[CoCopilot:ExecCommands]
TempSnarf: >0 "Snarf SourceDir/c Temp DestDir/c <>@0[ @*@@]"
TipSnarf: >0 "Snarf SourceDir/c TIP DestDir/c TIP@0[ @*.tip@]"
```

= = = STATE VECTOR POOL DISPLAY = = =

The state vector pool table (ho ho) describes the number of available state vectors for each process priority level. No processes at a particular priority level may run until a state vector becomes available for its level (by some faulted process having its fault satisfied and being restarted, thus freeing its state vector). A typical table displayed by this command is:

```
priority nStateVectors
    0    1
    1    5
    2    8
    3    5
    4    3
    5    5
    6    1
    7    1
```

= = = RUN TABLE DISPLAY = = =

The run table describes the properties of "runs". A run is a portion of a mapped space whose backing storage is contiguous (e.g. is a single run on the disk). There is at least one run for each mapped space, and typically, there is only one – – since the backing file for a mapped space is typically contiguous.

The run table contains Among other things, a run table entry contains data about the starting page and count of the run, a pointer to the swap unit data, flags indicating whether this run begins and/or ends a map unit, its Space.Usage and Space.Class, and the transferProc ID (backing storage implementation) that the memory is mapped to. A typical table displayed by this command (in nonverbose mode) is:

```
66 runs in run table.
  run  run swapU @swapU begin end  usage  xProc
  page count type  data  mUnit mUnit   class ID .
   540   3 irr    0 begin end   0 2   3
   543  33 irr    2 begin end   0 2   3
   578  20 irr   14 begin end   0 2   3
   648   1 unit  self begin end  30 6   3
   768  85 unit  self begin end   0 2   3
   853  10 unit  self begin end   0 2   3
   909  11 irr   22 begin end   0 2   3
   920   3 unit  self begin end   0 2   3
   ...
End of VM.
```

In verbose mode, the data for the backing storage implementation is also displayed (7 words).

A Start Key and Count may be specified for the run table. The run containing the specified page will be displayed, and the Count – 1 runs following it. Next! and Prev! may be used to continue this incremental display.

= = = SWAP UNIT TABLE DISPLAY = = =

The swap unit table describes the properties of swap units. Swap units are subdivisions of Pilot mapped spaces. The swap unit information displayed is: the swap unit's starting page and page length, availability for operations (avail/busy), access (write/read), swappability (swaps/pin'd), and life (alive/dead). The run table information is also displayed to show the run boundaries. A typical table displayed by this command is:

```
     swapU swapU
      page count  ...swap unit state...
   540   3 irr    0 begin end   0 2   3
       540   2 avail write swaps  dead
       542   1 avail write pin'd  dead
  ...
  3369   4 unit  self begin end  30 6   3
     3369   4 avail  read swaps  dead
  3373   8 unif  348 begin end  30 6   3
     3373   2 avail write swaps alive
     3375   2  busy write swaps alive
     3377   2 avail write swaps alive
     3379   2 avail write swaps alive
  ...
End of VM.
```

This way, the user get both functions, one is LeftShift – Space (my normal expand key), the other is RightShift – Space (used less frequently).

One small drawback to this: ExpandType will not work in FormSWs. (Although if you use Shift – Space for ExpandType expanding, you can leave the EXPAND key for old – fashioned expanding and that will still work in FormSWs).

Errata

There's an annoying bug that's in the current expand code. If you backspace, then type some more, then EXPAND, it often doesn't work or it does the wrong thing. I fixed that in ExpandType.

-- Fetch.doc

The Fetch Tool and DFetch Service

The Fetch tool locates or retrieves a released file. The user need not know the file's fully qualified name or which DF it is in. The Fetch tool lists the fully qualified path names for the file itself and its containing DF. If asked about alloc.bcd for Mesa 11.0b, for example, the Fetch tool would list:

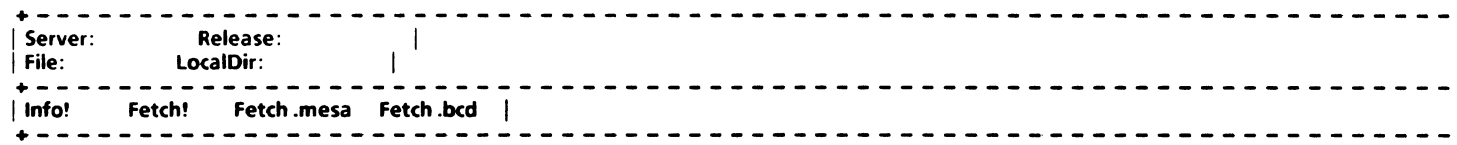
```
Retrieving information for alloc.bcd ... done. 1 match found.
[Host]<Directory>11.1>MesaBasics>Alloc.bcd!2  18-Jul-83 12:07:18 PDT
[Host]<Directory>11.1>DF>MesaBasics.df
```

For most queries and retrievals, the Fetch tool is as fast or faster than the file tool.

The Fetch tool, which is Courier-based, works in conjunction with a DFetch Service. The DFetch Service looks up the requested file in a release cross-reference (.xref) file created by the release statistics program RStats.

Using Fetch

The Fetch window looks like:



The server field identifies a machine running a DFetch service. The release field identifies the release of interest. The file field can contain either a short file name, a short name without extension, or a fully qualified file name. The LocalDir field identifies the local directory to which the file is fetched.

The Server and Release fields can be initialized in your user.cm, such as:

```
[Fetch]
Server: Gilroy
Release: Mesa11.0
LocalDir: <>Temp
```

The Info command lists the fully qualified names of all files in a given release with a given short name (and their containing DF's). If the file extension is omitted from the short name, all file names with that prefix are listed.

One or both of the "Fetch .mesa" and "Fetch .bcd" Booleans must be set if no extension is given in a fetch command.

In addition to running in a tool window, the Fetch tools commands are available in a menu on the root window and on the debugger window. If the Fetch tool is tiny, menu comands output their results to the default output sink (usually the Herald window).

Running a DFetch Service

A DFetch service can be run on either an integration machine or a workstation.
After a release, RStats /d can be used to create the cross-reference listing (i.e., the .xref file) used by DFetch services. The releases available to a DFetch service must be specified in the server's user.cm before the DFetch service is run:

```
[DFetchService]
Releases: Mesa11.0 Services8.0
```

Because the cross-reference file is not compressed, it is quite large. For Mesa 11.1, for example, it is ~1800 pages. A future implementation will use a compressed data file.

– – FaultBrowser.doc

FaultBrowser is a CoPilot UserProc which displays processor data and Pilot tables relating to the processing of page, frame, and write faults. It is useful in analyzing fault – related problems, and also gives information about recently used spaces and their properties. It also displays information about the current contents of real memory and its referenced and dirty status.

FaultBrowser displays five kinds of information:

* processor fault queues
* the processor ready queue
* the state vector heap
* Pilot's run table (mapped spaces)
* Pilot's swap unit table
* data about individual mapped pages

Some of this data is included in the information that CoPilot prints, but FaultBrowser typically prints more complete information, and in a tabular form.

= = = FORM SUBWINDOW = = =

The fields in the form subwindow operate as follows:

Table: {} This enumerated item is used to specify what kind of data FaultBrowser will display next, chosen from the following possibilities: the processor fault queues, the processor ready queue, the state vector heap, Pilot's run table (mapped spaces), Pilot's swap unit table, and data about individual mapped pages. More information on the significance of these tables are given in the next section.

Display Table! This command displays the entire table specified by the Table enumerated item.

For some types of information, FaultBrowser allows the user to display table item(s) related to a particular key. For example, one can ask for the mapped spaces which include a specified page. For tables which have a notion of sequence, a key may be accompanied by a count of items to be displayed, and the next and previous entry may be displayed. Some commands do not apply to some tables, and will respond with "not implemented".

Start Key = This numeric field specifies the key of the (first) item to be displayed.

Count = This numeric field specifies the number of table items to be displayed.

Display Table Items! This command displays those table items specified by Start Key and Count.

Next! This command displays the table item following the most – recently – displayed one.

Prev! This command displays the table item preceeding the most – recently – displayed one.

verbose This boolean field specifies that additional data, typically of low interest, should be included in displayed data.

= = = FAULT QUEUES DISPLAY = = =

The fault queues table describes the processes which are have taken a page, frame, or write fault and are waiting for the fault to be serviced. A typical line displayed by this command is:

PSB: 210B, priority: 1, L: 54410B, waiting page fault (being processed),
pointer: 1717000B ↑, page: 3636B

For page faults, the line shows Pilot's progress in the several stages of processing the fault (e.g. "being processed").

= = = READY QUEUE DISPLAY = = =

The ready queue table describes the processes which are "ready to run". However, these processes may still be blocked by the unavailability of a state vector for their priority level (see display of state vector pool). A typical table displayed by this command is:

PSB: 113B, priority: 3, L: 11470B, ready.
PSB: 104B, priority: 2, L: 3230B, ready.

- - FileSizes.doc

FileSizes runs in the executive. You give it command line arguments which are filenames, wildcards allowed of course. It produces a list of those files, along with their sizes, sorted by the sizes. FileSizes is intended to be useful in cleaning up the trash files that seem to accumulate on disks.

The normal usage of FileSizes is simply "FileSizes filename ...". There are, however, a number of switches available. Firstly, there is the "/A" switch, which makes the files come out in ascending order instead of descending order. Then there's the "nnn/I" switch, which tells FileSizes to ignore any files smaller than or equal to nnn pages. Then there are the /L and /R switches, which tell FileSizes to look at locally - created files only, or remotely - created files only, respectively. Lastly, there is the "volume/V" switch. This tells FileSizes to list all the files on the specified volume. A /V with no volume specified uses the current volume.

Here is a summary of the switches (also available via "help FileSizes"):

/A - - - - - - - - sort in ascending order instead of descending
/L - - - - - - - - locally - created files only
/R - - - - - - - - remotely - created files only
nnn/I - - - - - ignore files smaller than or equal to nnn pages
volume/V - - list all the files on the specified volume

Examples:

FileSizes /r *.mesa	- - list all Mesa files created remotely
FileSizes /L Tajo/V	- - list all locally - created files on the Tajo volume
FileSizes 50/i /V	- - list all files on the current volume which
	- - are larger than 50 pages

A Start Key may be specified for the swap unit table. The run containing the specified page will be displayed, and the Count – 1 swap units following it. Next! is implemented. Prev! is ignored.

= = = PAGE DISPLAY = = =

The page displays shows the properties of individual pages of mapped spaces. The page information displayed is either "vacant", indicating that the page is swapped out, or its cleanliness (clean/dirty) and referenced status (ref'd/unrefd). The run table and swap unit table information is also displayed to show the run and swap unit boundaries boundaries. Runs of adjacent pages with identical properties are displayed in a single line; each line shows the starting page and number of pages of the run of like pages. A typical table displayed by this command is:

```
    swapU swapU
     page count ...swap unit state...
...
      549   1 avail write swaps alive
        549   1 dirty ref'd
      550   5 avail write pin'd alive
        550   1 dirty ref'd
        552   3 clean unrefd
        554   1 dirty ref'd
      555   1 avail write swaps alive
        555   1 dirty ref'd
      556   1 avail write swaps alive
        556   1  vacant
...
End of VM.
```

A Start Key may be specified for the page display. The specified page will be displayed, and the Count – 1 pages following it. Next and Prev! are ignored.

-- Flash.doc

"Flash" flashes the screen and breedles the tone generator every two seconds until you hit STOP.

"Flash once" flashes and breedles only once and then quits. (The "once" can be any string, actually.)

If you load Flash in background priority, it does a "Flash once", so you can run it as the last thing in your Background.bcd command line to notify you when background loading has finished. (See Background.doc.) Another use for Flash is as a simple alarm; in the Exec, type "Wait 17:00;Flash". (You'll have to create another Exec window to do anything else in the Exec before 5pm.)

-- FileMover.doc

FileMover is a tool to transfer files between directories in the Mesa Development Environment. There are three string items to fill in: "From Directory", "To Directory", and "File List". When the "Move!" command is invoked, the files in the File List are moved from the From Directory to the To Directory.

The File List may contain wildcards and command files (*'s and @'s). Any command files in the File List must be findable on the current search path.

Neither the From Directory nor the To Directory need to be on the current search path. But they both must be on the currently running volume; i.e., no inter-volume moves are allowed. Thus, both directories must begin with either "<>" or "<volumeName>".

FileMover will refuse to move a file into the To Directory if a file of the same name already exists there, unless the "Overwrite" Boolean item is turned on (it is initially off).

Normally, FileMover runs in the Notifier process. Turning on the "ForkProcess" Boolean item causes FileMover to fork a separate process to do the file moving. The user may set the priority of the new process from the "Priority" enumerated item, which only appears when ForkProcess is on.

As each file is moved, a message is displayed in the bottom subwindow of the tool. If anything goes wrong during a move, an error message is displayed, and FileMover continues with the next file in the list (unless the error is global, e.g., directory full, in which case FileMover aborts).

FileMover will abort cleanly between files in the File List if the user hits ABORT with the cursor anywhere inside its window.

Floppy – Delete!
Deletes files specified in the Source from the floppy disk. If for any reason a file cannot be deleted, that file is skipped and processing continues with the rest of the files in the list. NOTE: the more files on a floppy, the longer it takes to do a delete. BE PATIENT!

Options!
This command opens the OptionSW.

OptionSW - - -

Boolean options:

Type
Display file type.

Length
Display file length in BYTES.

Create
Display time of file creation.

Write
Display latest write access time.

Sorted
When doing Floppy – List!, sort the files first. This option makes Floppy – List! take longer, but the listing is more IFS – like.

Count
Interpret To/Count = number item in FormSW as a count of pages rather than an upper boundary.

Number options:

File Type =
CARDINAL specifying File.Type Default is LAST[CARDINAL].

Format: Files =
CARDINAL specifying maximum number of files allowed on a floppy. See Format!

Command options:

Apply!
Apply changes to options.

Abort!
Don't apply changes to options.

Tool - - -

The tool can be safely loaded, deactivated, and unloaded, doing all the right things. It even registers a NoteSW for ToolDriver.

Notes & Cautions - - -

Pseudo – Directories

These are fun! They imitate the IFS directory structure. Actually, the Source string is just concatanated onto the end of the Directory string just before wildcard expansion is performed, so entering:

Directory: foo>bar>*>bletch
Source: dufus

... is the same as ...

Directory:
Source: foo>bar>*>bletch>dufus

Page ranges

This is kinda tricky. Lets use the example given in the XDE User's Guide for page ranges. To do the equivalent of

> Floppy Write HugeFile[0!2000]

-- FindSource.doc

Environments: CoPilot|Tajo
Description: FindSource searches for and displays the implementation or defintion of interface items such as procedures, signals, and exported variables etc.

If you're examining a program and you want to look at the implementation or definition of FooDefs.Action, you can use FindSource to load the file in a window scrolled to the correct line position.

How to Use FindSource

FindSource creates a FindSource menu containing six commands that is present in all loaded FileWindows. All six commands cause filewindows to display the source when found, as explained below. FindSource takes its arguments from the current selection. These arguments are strings of the form "Interface.Item". This asks FindSource to look up " Item:" in the files "Interface*.mesa". Case is significant. All of the commands either find a match and load a window or blink the display and print out a message to the default typeout (usually the Herald.) The commands are as follows:

LoadImpl, LoadDef: Loads the file into the window through which the menu command was invoked.
OpenImpl, OpenDef: Creates a new filewindow loads the file into that new filewindow.
ShowImpl, ShowDef: Loads the file into the window chosen according to the same heuristics used by the Debugger's "s" command during "Display Stack" mode.

The commands that end in "Impl" find the implementation of the selected item; the commands ending in "Def" look up the definition of the item.

How does it work?

FindSource enumerates likely mesa source files on the current search path and does string searches for the interface item desired. The common suffixes for DEFINITIONS files are stripped before the "*.mesa" is appended for enumerating files. The suffixes skipped are "Ops", "Defs", "Internal", "Extra", "Forgot", "Priv", and "Private". The suffixes are stripped in that order so that CaretForgotDefs.XXX will result in an enumeration of Caret*.mesa to search for " XXX:". In order for a search for an implementation to succeed, the implementing source file must EXPORT the Interface.

How to Install FindSource

FindSource is can be run in CoPilot or Tajo. It can be started in the Initialize section of the User.cm, thus making it available immediately.

For convienience, augment the [Symbiote] section(s) of your User.cm to contain a new command in the Symbiote Menu. For example:

```
[CoPilot:Symbiote]
Menu:Break Create Edit Find Load Position Reset Save ShowImpl Split Store Time Wrap
```

Limitations

FindSource can't find the implementation or a definition unless the source file containing is on the current search path, and begins with the same prefix as that for the interface in the argument to FindSource.

- - FloppyRecoveryTool.doc

This tool is designed to allow users of floppies to recover as much information as possible from any damaged floppy disk. In achieving this aim it presents the user with a file list of the files it thinks are on the disk and then allows the user to attempt to retrieve any he/she wishes to a file created on hard disk.

The tool presents the user with a tool window and allows the user 3 commands: -

Check Floppy -

This prints out the file list of all files on the disk either by name or by their order in the file list. Additional information is given such as location of the file(only printed with the number option), the files ID, the files type numerically and the size of the file.

Choice of which type of list to be printed is made via bugging either "Names" or "Numbers" and then bugging the "Check Floppy" command. The "verbose" option tries to tell the user as much information as possible about what is wrong the floppy being processed. The tool works in such a way that firstly it tries to conventionally find the File list via reading Sector nine of the disk and finding its address. If this fails the tool will search the data sectors for the file list. On failing to find the File List the tool will check each data sector and recreate the file list in virtual memory. This will be saved until such time as the floppy drive is opened. Should the us
file list from scratch, the Verbose switch must be set. This will overide any file lists already created at the expense of time.

*** It should be noted floppies with no file list or those checked with the verbose switch may take some time approx 15 mins before any response is seen.

Retrieve by Name -

This facility copies as much, as possible, of the file specified by "source" to a hard disk file named by "destination".

Retrieve by Name -

This facility copies as much, as possible, of the file specified by "number" to a hard disk file named by "destination". The number is that of the position of the file in the file list, on a check floppy by number the number of the file follows the "#" character.

**** PLEASE NOTE *****

Firstly the tool is not comprehensively tested as a good supply of incorrect floppies with a variety of bugs was difficult to come by, as such anybody using the tool is really a beta test.

********** Known Bugs, features, and stuff that will change (maybe) **********

The drive number should always be ZERO or the tool will crash .

Any destination file should not exist or it will overwrite the current version.

The tool may according to the type of errors on the floppy disk take some considerable time to either reconstruct the file List or tell the user that the disk cannot be recovered. In these circumstances some time may be saved by setting the Verbose switch though even under this condition the tool may take up to twenty minutes.

- - FloppyFileTool.doc

ABSTRACT:
The FloppyFileTool is meant to replace the Executive Floppy.~ command. It is a full fledged tool with an interface that is very similar to the FileTool. The FloppyFileTool contains all of the functionality of Floppy.~, plus two local file commands. In addition, the FloppyFileTool allows the user to set up pseudo - directories (like IFS directories) on the floppy which are parsed and understood in the same way as the FileTool. Full wildcard (* and #) patterns are supported for both directory names and file names. Only the Format! command runs in the Notifier, all others are FORKed.

FormSW - - -

Directory:
Name of the pseudo - directory. Wildcards are allowed. This string item ALWAYS refers to the floppy, NEVER to the local filesystem. Used by Retrieve!, Store!, Floppy - List! and Floppy - Delete!

Source:
A list of files (separated by spaces) for the next command to act upon. File names may include "expansion characters" (wildcards). This list applies both to the floppy and the local filesystem. EXCEPTION: For the Format! command, this string item is used as the name of the floppy volume. Used by Retrieve!, Store!, Local - List!, Floppy - List!, Format!, Local - Delete!, and Floppy - Delete!

Dest'n:
The file name for the destination of a transfer. This single string item applies to both the local and floppy filesystems. Used by Retrieve! and Store!

LocalDir:
All references to the local disk will only occur within this directory. This string MUST begin with the volume name <volume> or the abbreviation <>. Wildcards are allowed. Used by Retrieve!, Store!, Local - List!, and Local - Delete!

Pages: From =
This number item is used for specifying the lower bound of a subrange (IN PAGES) of a file (see To/Count =). It defaults to 0. Used by Retrieve! and Store!

To/Count =
This number item is used for specifying the upper bound of a subrange (IN PAGES) of a file (see Pages: From =). It defaults to 0. When defaulted, the range is understood to mean "... to the end of the file." When the boolean option Count is selected (TRUE), this number is interpreted as a COUNT of pages, rather than a boundary (see OptionSW, Count). Used by Retrieve! and Store!

CommandSW - - -

NOTE: All commands EXCEPT Info - Disk! and Format! can be aborted with the STOP key.

Retrieve!
Transfers the file name specified in Source from the floppy to the local disk. If Dest'n is blank, the file name of the copy made on the local disk is the source file names stripped of FLOPPY directory qualifiers.

Store!
Transfers the file name specified in Source from the local disk to the floppy.

Local - List!
Lists all files on the local disk corresponding to the file list in Source. The information listed is controlled by the OptionSW selections.

Floppy - List!
Lists all files on the floppy corresponding to the name file list in Source. The information listed is controlled by the OptionSW selections.

Info - Disk!
Gives information about the floppy. Indentical to the Floppy.~ Info command.

Format!
Prepares a floppy for storage. The name of the floppy volume is specified in the Source field. The maximum number of files allowed on the floppy is specified in the OptionSW (see OptionSW, Format: Files =). The default number is 64. Format! will ask for confirmation if there appears to be valid data on the floppy.

Local - Delete!
Deletes files specified in Source from the local disk. If for any reason a file cannot be deleted, that file is skipped and processing continues with the rest of the files in the list.

There can be any number of loaded files specified.

After setting up your User.cm and retrieving all the Font file, just type into the Exec: "Run FontMonster.bcd" and after the User.cm is parsed by FontMonster, fonts should start appearing in tools and a new menu will be on the root windows menu.

... you do the following ...

Step 1:

FormSW
Source: HugeFile
Dest'n: HugeFile[0!2000] - - or HugeFile1, or HugeFileA, etc. if you don't care about being compatible with Floppy.~
Pages: From = 0 To/Count = 2000

Step 2:

Bug Options!, set Count option to TRUE.

Step 3:

Bug Store!

Voila! To finish up (ie. do Floppy Write HugeFile[2000]) do

Step 4:

FormSW
Source: HugeFile
Dest'n: HugeFile[2000] - - or HugeFile2, or HugeFileB, etc. if you don't care about being compatible with Floppy.~
Pages: From = 2000 To/Count =
- - notice that To/Count is "nil" which will cause the default to be used

Step 5:

Bug Options!, set Count option to FALSE.

Step 6:

Bug Store!

To retrieve the file, set up the FormSW like ...

Source: HugeFile[0!2000] - - or HugeFile1, or HugeFileA, etc.
Dest'n: HugeFile
Pages: From = 0 To/Count = 0

... and bug Retrieve!, then do ...

Source: HugeFile[2000] - - or HugeFile2, or HugeFileB, etc.
Dest'n: HugeFile
Pages: From = 0 To/Count = 0

Indicator - - -

The indicator does not represent any interesting information. It is just there for show.

User.cm - -

- - A sample user.cm entry follows:

[FloppyFileTool]
WindowBox: [x:512, y: 404, w: 512, h: 252]
SetOptions: Length Create

- - SetOptions: may be any of Type, Length, Create, Write, Sorted, Count.

NOTES & CAUTIONS

Certain floppy operations will appear to take a long time. This is because it is often necessary to scan the whole floppy "directory."

DO NOT DEACTIVATE THE TOOL WHILE IT IS RUNNING (IE. WHILE YIN/YANG IS SHOWING). THE TOOL WILL CRASH IF YOU DO.

-- FormSWLayoutTool.doc

The FormSWLayoutTool allows a Tajo user to graphically layout a Tajo FormSW. After laying out the form, the user can generate the Mesa source file using the FormSWLayoutTool. This source can then be compiled and executed and the resulting Tool window will have the user's Form subwindow.

Description of the FormSWLayoutTool:

The layout tool has three subwindows: a message subwindow, a form subwindow, and a blank subwindow used for drawing a form subwindow. The layout tool has the following items on its form subwindow:

FormType: {bool, command, enum, longNum, source, string, tag}
This enumerated item is used the select the type of form item that you want to put on your form.

Tag:
In the Tag: field is the tag for the next item that you want to put on your form.

Zone:
The Zone: field is provided so that the user can specify a name for the zone from which storage will be allocated.

AlignX
This boolean allows forces that items that are layed out to be on columns defined by the width of the character '0.

Usebox
If this boolean is selected, the generated tool will have the same window box as the current size of the layout tool.

Anyfont
If this boolean is selected, the layout tool will generate source for a tool which will have proportion spacing on its form subwindow given any system font.

Root:
This is the root name of the program, file, and tool that will be generated.

Do It!
This will cause the layout tool to generate Mesa source for the form subwindow.

Clear!
This will cause the bottom subwindow to clear itself

Set Defaults!
This will bring up an option sheet which will allow you to set the defaults for the property sheets of the different form items.

Save!
This will save the contents of the bottom subwindow in an intermediate format. The file generated will have the root name in the Root: field with the extension ".by"

Load!
This will load a .by file into the layout tool. The .by extension is automatically

Plagiarize!
This will allow the user to select a form subwindow from a existing tool. Just bug Plagiarize! and the cursor will change form. Move the cursor over the form subwindow you want to copy and hit Point. If you want to abort, hit the Adjust button.

Method of Operation:

The layout tool has modes of operation: Initial layout and editing. You are in 'initial layout' mode if there is text in the Tag: field of the tool's form subwindow, and in 'editing' mode if there is no text.
When in the layout mode, you can place items in the bottom subwindow by moving the cursor into the bottom subwindow and hitting Point where you want the tag to be located. Upon entering the bottom subwindow, the cursor will become a brush which can be placed anywhere in the subwindow. All you can do in layout mode is to place form items in their initial places. As you edit the Tag: field and the FormType: {} field the brush takes on new looks to match the information in these two fields.
To get into editing mode, just delete the text in the Tag: field. When the cursor is moved into the bottom subwindow, it will remain an arrow. You can now select items to be manipulated by hitting Point over that item. Selected items are displayed by inverting the bits on the item. There can only be one item at a time which is selected. Once an item is selected, the following function are on the following keys:
DELETE: will cause the current selection to be deleted.
UNDO: will bring back the last deleted item
MOVE: will allow you to move the selected item
STOP: will abort a move when in the middle of a move

- - FontMonster.doc

FontMonster allows you to tailor your environment by allowing you to use different fonts in different text windows. FontMonster provides a menu on the root window that has a list of fonts. If an item on the menu is selected, the cursor is changed into a little face. Moving the cursor over a text subwindow and hitting Point will cause that subwindow to have the selected font. If Adjust is hit, FontMonster will abort the users choice. The first item on the menu is "System", which is the system's default font. You specify the rest of the entries in the menu through the User.cm (Explained below).

FontMonster also allows you to assign fonts to specific text subwindow in specific tools. You can have a different font in a tools Message, File, TTY, or Text subwindows. Fonts for loaded files and empty windows can also be specified. Fonts are specified through the User.cm file.

To use this program:

1. Retrieve FontMonster.bcd.
2. Retrieve the desired font files. (Explained below)
3. Set up User.cm (Also explained below)
4. Run FontMonster.bcd

Getting the desired font files:

Suppose you want to use 4 fonts: CREAM10.STRIKE, HELVETICA10.STRIKE, HELVETICA12B.STRIKE, and TIMESROMAN8I.STRIKE. First, retrieve the font files from the diskette. (See a list of font files that are on the diskette).

Setting up the User.cm:

**** A sample User.cm ****

```
[Chat]
WindowBox: [x: 512, y: 100, w: 512, h: 400]
TTYFont: CREAM10

[CommandCentral]
WindowBox: [x: 0, y: 404, w: 512, h: 374]
MessageFont: TIMESROMAN8I
FileFont: HELVETICA10

[FileWindow]
EmptyFont: TIMESROMAN8I

[Executive]
WindowBox: [x: 512, y: 448, w: 512, h: 330]
InitialState: active
TTYFont: TIMESROMAN12B
DefaultFont: CREAM10  - - if a symbiote is on the exec, it will get this font.

[FontMonster]
FontList: CREAM10.STRIKE HELVETICA10.STRIKE HELVETICA12B.STRIKE TIMESROMAN8I.STRIKE
MenuOnly: FALSE
Loaded: User.cm CREAM10
Loaded: ThingsToDo HELVETICA10
Loaded: Calendar TIMESROMAN8I
```

*** end of User.cm ***

Setting up your User.cm to use this program:

1. All fonts must be specified in the "FontList" part of "[FontMonster]". Each font file must include the proper filename extension, ie CREAM10.STRIKE vs. only CREAM10. These fonts will appear in FontMonster's menu.
2. MenuOnly is TRUE if you only want the FontMonster menu, FALSE if you want the below functions.
3. In each tool, you can specify font for the following types of subwindows, "TTYFont", "MessageFont", "FileFont", and "DefaultFont". If a tool has a text subwindow that is not a message, file, or tty subwindow, use the "DefaultFont" section. All fonts specified need only the root name of the file, that is, use CREAM10 instead of CREAM10.STRIKE.
4. In the "[FileWindow]" section, the "EmptyFont" specifies the font for empty windows.
5. In the "[FontMonster]" section, one can specify fonts for loaded file windows. The sections look as follows: "Loaded: <Filename> <Font>".

-- FSWindowTool.doc

FSWindowTool is a Star-like file tool which runs in Copilot or Tajo. It lets you examine and operate on Filing volumes which are either on your own machine (e.g., your OS 5.0 Star "user" volume) or on a public or private OS 5.0-compatible File Service. Access to Filing volumes on remote machines is limited by enforced access controls, while no access controls are enforced for local Filing volumes. This implies that you can only screw up your own files or files to which others have given you permissions.

You normally operate FSWindowTool in the pointing and opening mode as you do in Star, but you also have the option of entering the string name (or pathname) for the desired object when scrolling is less convenient or inappropriate. Thus, FSWindowTool combines the advantages of Star and the Mesa File Tool for accessing Filing volumes.

If you have used Star, you can probably figure out how to use most of this tool just by running it, so the following can be skimmed and referred to as needed, though you should probably read at least the current list of restrictions (at the end of this document).

Specifics:

For 11.1 -

Required software:
    RunFSWindowTool.bcd
    FSWindowTool.bcd
    Filing.bcd (optional - for local ops only)

In the executive of either Tajo or Copilot type:
RunFSWindowTool

This will cause Filing.bcd to be loaded first (a necessity), then FSWindowTool. If Filing.bcd is not around, FSWindowTool will still work, in a remote-only mode, i.e. it will not present you with the "local" option.

If you operate on your star User volume, be sure it is done at key stop 2 (if you have separate star/user volumes) or else reboot star in lieu of proceeding from the debugger. (I.e., the same restriction as with the StarFileTool).

Logon Window:

Initially you are in the logon window, and you have a choice of three options:
- o LOCAL is used to operate on LOCAL Filing volumes.
- o REMOTE is used to operate on REMOTE Filing volumes.
- o MESA is used to operate on the MESA development environment file system (MFile) on the current volume (in a very limited way -- i.e., naming directories as the destination of a copy or naming files as the source of a copy).

The user name and password will be retrieved from the Profile Tool on startup of the tool and will be updated each time you update your name and password in the Profile Tool. Changing you name and/or password has the effect of changing your identify for all the windows that make up this tool.

LOCAL mode:

When the LOCAL option is taken, you will be presented with a list of local logical volumes which are of equal or lower type than the current logical volume. If you really feel the need to open a volume of higher type (e.g., a debugger volume from a normal volume), you can enter the name of that volume after the prompt "Specified Volume:". You can then select one of the volumes provided to you or select the volume whose name you typed in and bug the command "Open&List!". This will create a new (directory) window which lists the first few files at the root of selected volume. The directory window is described in more detail below.

MESA mode:

-- ForkLogin.doc

ForkLogin.bcd creates a Form window which asks you for your login name and password. The login name is initialized to the value from the Profile, which is initially taken from User.cm. If you NEXT out of the password field, the Apply! command is invoked automatically. After either Apply! or Abort! is invoked, the window disappears.

The ForkLogin window is created only if either the User or the Password currently registered is empty. The ForkLogin window will appear later any time these credentials are changed so that one or both is empty. Thus the window will be brought up the first time you enter CoPilot after booting another volume, or if another program (such as the hack DMT) smashes your password, etc. You can specify where the window is to appear by putting a WindowBox parameter in the ForkLogin section of your User.cm. The default is to fill the screen.

If you include this in CoPilot's "Initialize" line, the login window magically appears on top whenever you reboot CoPilot. The effect is roughly equivalent to including "//eventBooted: Login.~" in the old Alto User.cm. (You cannot include "Login.~" in the Initialize line; it causes deadlock.)

This is a much more secure way to log in than having a Login.bcd or Login.mesa file hanging around your CoPilot volume (or, worse, your IFS directory).

with descendent directories still open. Each directory window is completely independent.

- "Close All!": Destroys all active windows which originated, directly or indirectly, form the same "Open&List!" command on the Logon window. Tiny and inactive windows are not affected. "Close All!" has the same effect on all active windows in the same 'family' of windows. Therefore, any windows that you want to save around, as icons on your Xerox Development Environment 'desktop' should be made tiny or inactive before issuing the "Close All!" command. The analogy with Star icons is accurate. Tiny (or inactive) directory windows have the same functionality and efficiency as closed reference icons in Star. Making a directory window tiny is a good way to redisplay a window (necessary when other users are changing a remote window you are looking at) or for going back to the start of the directory.

- "Attributes!": This is similar to Star's property sheets except you get to see and change more of Filing attributes (see current restrictions below) with FSWindowTool's attribute window. Any attribute the tool lets you change will be reflected back into the file when you bug "Apply!" or be discarded if you say "Abort!". The "Pages" attribute is really Filing "subtreeSize" attribute so; for a directory that gives the size of the directory and all of its descendents. You can have attributes windows open for several files concurrently, unlike Star. A more complete description of "Attributes" windows is given later.

- "Free Space!": This command is only available for local directories and gives the amount of free space (in disk pages) on the local Filing volume. This may not be the same as the available space when star is running since temporary files have been deleted by opening the volume. Also this number will differ from that shown by the Mesa file system, since Filing reserves about 50 pages for its own use.

- "Next Page!": Scrolls to the next screen full of the directory (16 entries per screen full). If there is selection, the command instead positions the selected file at the top of the page. The selected file may be entered on the "Specified File:" line in order to scroll to a distant entry in a large directory.

- "Previous Page!": Scrolls to the previous screen full of the directory, or to the start of the file, which ever comes first. If there is selection, the command instead positions the selected file at the bottom of the page. The selected file may be entered on the "Specified File:" line in order to scroll to a distant entry in a large directory.

- "Copy!": Begins a copy operation, taking the selected file as the source. It is completed by the "Confirm!" (or "Cancel!") command, neither of which is available until an operation is in progress. Copy can be used to copy between file services, copy between a file service and your Star desktop, copy between the local Mesa file system and the Filing volume on your desktop or on a File Service. The destination must be a directory rather than a file. It is not currently possible to rename a file as it is being copied. You must bug "Confirm!" in the window where you select the destination directory. If you bug "Confirm!" without selecting a destination you are selecting the open directory as the destination (akin to selecting after all of the items in a Star directory). In this case, a second confirmation will be required. [If you specify the open window using the "." name described above, no second confirmation is required.] Copying between two Filing volumes (on the same or different machines) can be done for directories as well as for individual files, exactly as can be done with Star. Copying to the Mesa file system is not necessarily reversable since attribute information (e.g., File type) may not be preserved.
  When copied to the Mesa file system, Multi – segment Filing files (e.g., Star Documents) are stored in compressed (segment – encoded) form as a single file. You cannot copy a directory into the Mesa file system. You must either copy each individual file separately or else use the "Serialize!" command to produce a serialized representation for the directory. Copying a text file out of the Mesa file system will create a file of type NSAssignedTypes.tText

ITALICS: will the equivalent of a JFirst on a textsubwindow.
PROPS: will bring up a property sheet on the selected item. The properties of each item are the parameters needed in the FormSW.ClientItemsProcType for making the item. For most items it will not be necessary to edit the property sheets unless you want special identifiers for the variables. The exception are the enumerated items, which must be edited because the the choices in the enumerated items window. The syntax for the choices is as follows:

```
Choices : = TOKENLIST
TOKENLIST : = TOKENLIST CHOICEITEM | CHOICEITEM
CHOICEITEM : = TOKEN | [TOKEN, VALUE]
VALUE : = valid Mesa variable | a numeric constant
TOKEN : = WORD | "WORDLIST"
WORDLIST : = WORDLIST WORD | WORD
WORD : = a list of any characters except space and double quote.
(Note: if VALUE is a Mesa variable, the generated source won't compile
without the user editing the source)
```

Example:
IF in an enumerated item prop sheet:
Choices: Waitaminnit [buddy, 43] "fix the enums"

results in the Mesa code:

```
ARRAY[0..3) OF Enumerated ←[
   ["Waitaminnit", 0], ["buddy", 43]
   ["fix the enums", 2]];
```

which results in a form item:
Item: {Waitaminnit, buddy, fix the enums}

After working on the form, you can either hit DoIt! to generate the Mesa source, or hit Save! to save the form in a loadable intermediate format. The files will have the name in the Root: field followed by .mesa for a Mesa source or .by for the intermediate format.

The default option sheet allows the user to set the defaults for new items which are created. It also allows the user to modify the name of the data handle, the name of the scalar type, and to put some arbitrary text in the FormSW.ClientItemsProc. These look as follows:

[Global Things]

EnumType:	Default = "FormItems", is the name of the generated scalar
HandleName:	Default = "data", the name of the data handle.
ProcName:	Default = "MakeForm", name of the FormSW.ClientItemsProc.
StuffString	No default. This is an arbitrary string which is stuffed into the FormSW.ClientItemsProc after all the local declarations. One use of it may be for comments or for getting a data context from a window, ie. StuffString = " data: MyContext ← Context.Find[c, sw]"

Notes:
After working on a form for awhile, its good to use the Save! command so that if you crash, you won't have lost all your work.

Use the property sheets to make the form as close as you can to the final code, ie put in the real variable names you're going to use, rather than the defaults the tool provides.

Also, save the .by file that is generated by the Save! command after you finish the form. This way, the next person to work on the program will be able to load this file and easily modify your tool's looks. Once you start doing massive edits to the generated source, the ability to easily to modifications become very difficult.

Known bugs:
The FormSWLayoutTool will not check for valid Mesa identifier names. If you pass a bogus one the tool, odds are it will not compile.

The format of the intermediate .by file is very similar to the Mesa source, if you feel so inclined as to edit these, you deserve what you get.

Before deactivating the tool, it is wise to destroy all the property sheets by using the Close! command on the property sheet. Likewise, you should make sure the Defaults options sheet is also close. Not doing this will likely result in an address fault somewhere.

The following attributes require an explanation for how they are manipulated, the other attributes which can be changed can simply be changed like you would do on a Star property sheet:

- o Type. A menu can be brought up over the Type attribute. This gives the types FSWindowTool knows about. You can enter one of these or select any other long cardinal value. To allow a symbolic name to be displayed, FSWindowTool ignores anything entered in this field after an open parenthesis or a blank space.

- o Checksum. If the checksum attribute is ever wrong on a file, the filesystem won't let you look at the contents of your "damaged" file. You can set the checksum attribute to the unknown value (177777B) if you want to take your chances with the file.

- o Directory Orderings. The two fields "Ordering" and "Direction" make up the ordering attribute of a directory. You can sort a directory by any of the attributes available in the ordering menu.

- o Access Lists (Access List and Default Access List). The Access List and Default Access List attributes are changed using the "Change Access List!" and "Change Default Access List!" commands, respectively, as well as the "Same As Parent" and "Same As Access List" booleans, respectively. When the booleans are not set the corresponding access list is explicitly set on the file. When the booleans are set, the access lists displayed are the effective access lists which are computed from the default access list of the parent, or the access list of the file, respectively. The "Unify Descendent Access Lists" boolean makes all files which are descendents of the current directory have access lists which are computed from this window's default access list. Like everything else in the Attributes window, this does not take affect until the "Apply!" command is invoked. To change an access list, enter the name of a group or individual in the "Name" field on the "Access Entry" line. Specify the desired access and invoke the "Change Access List!" or "Change Default Access List!" commands. You can use this method to change existing entries, add new entries or to eliminate entries. The special string "(World)" can be used to indicate the group containing everyone. System administrators may request that you use a valid clearinghouse group (such as AllOSBU:OSBU South:Xerox) to restrict access; however the special "(World)" group is more easily evaluated by the file service.

- o Extended Attributes. The line labeled "Currently Set Extended Attributes" lists the extended attributes which are currently set at any given point in time. To see a particular extended attribute, find its name in the "Attribute" menu or else enter the attribute number in the "number" field. For attributes that are understood by FSWindowTool, the "Interpretation" field will be set to the appropriate value, otherwise you will have to choose the interpretation given to your extended attribute. Once you have chosen the attribute (by name or number) and the interpretation, use the "Show Extended Attribute!" command to display it on the line after "Currently Set Extended Attributes". Although you can only see one extended attribute at a time, all changes to extended attributes will be remembered until the "Apply!" command is invoked.

A non – obvious use of FSWindowTool:

- o If you create an interpress file using the Mesa "Print" command, you can store it out and change it's type to "interpress". This will let Star users print your file.

Outstanding (known) problems [Current Restrictions]:

1. Deactivating and reactivating an Attributes window will crash (why would anyone want to do this anyway?).

2. The tool uses the systemZone and hasn't been checked for storage leaks.

3. Cannot connect to non – english file services.

When the MESA option is taken, you will be presented with just the current logical volume and the "Specified Volume:" prompt. You may select that volume or else enter the name of an already opened (in the Pilot/Mesa sense) logical volume and once again bug "Open&List!". A directory window, similar to the one created in LOCAL mode is created. Only those functions necessary for interacting with Filing volumes are provided. Others such as directory enumeration and file deletion which can be accomplished using the Mesa executive or FileTool are not attempted by the FSWindowTool.

REMOTE mode:

When the REMOTE option is taken, you will be presented with a list of public File Services which are in the domain and organization indicated by the "Domain:" and "Organization:" options. The choices of domain and organization are initialized using the domain and organization of the profile tool, if set. You can select the name of a Domain or Organization from a pop up menu or type the names into the respective fields directly. Selecting from the pop up menu will reset the state of the tool automatically. Typing the name of a Domain or Organization requires setting the tool state manually using the "Set Domain/Organization" command. If both the Domain and Organization names are changed, bugging the "Set .." command will cause the organization to change first. You can also enter the name or network address of a private File service after the "Specified File:" prompt if you so desire. You may choose one of the supplied services or else select the one you typed in and then bug "Open&List!" to get a directory window to a remote file service.
Note: – – Mail Services are no longer linked with the File Service.

Directory Window:

When you open a File Service or local volume, you will enter a directory window. Here you are given a list of files and directories which are at the root (or top – level) of the file system. You will also be given the prompt "Specified File:" which can be used to enter the name of a file or a file ID (identified as 5 octal numbers separated by spaces) which is in this directory but which is not currently displayed on the screen. This prompt can also be used to enter a path name relative to the currently displayed directory. For the purpose of this tool, path names are strings using "/" as a separator between file names, and '! separating file names from version numbers. It is also possible to escape "/"s which are included in file or directory names. The single character "." is given special interpretation by FSWindowTool when it is entered as the "Specified File". It is used to indicate the directory whose listing is displayed in the window. Thus, you can copy an open directory, get attributes of an open window (including the root of a volume) or copy into an open window all by typing and then selecting the single character ".". You may also enter the file ID (only) of a file NOT in this directory, but on the same volume.

Although you might be tempted to use the scrollbar to scroll to other files in the directory, this will not work. FSWindowTool always shows 16 items at a time. The commands "Next Page!" and "Previous Page!" described below are used for scrolling. The scrollbar is only useful should you resize a directory window so that all sixteen items cannot be displayed at once. Here are the various commands, most of these operate on the currently selected file or path name:

- o "Open&List!": Opens the selected directory. Only directories can be opened. Directories are indicated by a " – " in the first column of the window. Opening a directory results in another directory window being created. Unlike star, each opened directory creates a distinct, overlapping window. This allows you to see more than one directory at a time and move files, for instance, from one directory to another. Opening a directory using a pathname suppresses the intervening windows. The effect of "Open&List!" is undone by the "Close!" or "Close All!" command. It is pefectly acceptable to open the same directory multiple times. In such a case, a change to one window will be immeidately made visible in the others.

- o "Close!": Destroys the current directory window. This has no effect on any other window. In particular, it is legal to close a directory

-- Gateway [illegible]

Please refer to the document, "Ethernet Monitors", in XDE Unsupported Software Description.

(the same as the Mesa FileTool) which is understood by Star and therefore can be converted using the OS 5 Star Converter Icon.

- o "Move!": Begins a move operation, taking the selected file as the source. It is completed by the "Confirm!" (or "Cancel!") command. When the source and destination are on different Filing volumes, it is equivalent to "Copy!" followed by "Delete!". "Move!" cannot be done to or from the Mesa file system.

- o "Serialize!": Similar to copy, except the result is the serialized representation of a file and, if a directory, all of its descendents. This command should be used to transfer arbitrary files or directories into the Mesa file system when done for archival purposes. The serialized representation records all of the attribute information of the original file(s) so that they can later be restored completely.

- o "Deserialize!": Similar to copy, except the source must be in the serialized representation of a file or directory. This undoes the effects of "Serialize!". Desktops stored on file services are stored in this format, so they can be deserialized using this command.

- o "Delete!": Deletes the selected file, unless it is a directory which is not empty in which case the deletion doesn't occur until the "Confirm!" command is executed.

- o "Confirm!": Only is made available when a "Copy!", "Move!", "Serialize!", "Deserialize!" or "Delete!" operation is in progress. For "Copy!", "Move!", "Serialize!" and "Deserialize!", the selected file is the destination directory. Even if the selected file is a typed – in file name, it must still be the name of a directory and not the actual file name – – renaming during copy is not supported.

- o Cancel: Aborts a "Copy!", "Move!", "Serialize!", "Deserialize!" or "Delete!" operation in progress.

Attributes Window:

An Attributes window is created by the "Attributes!" command in a directory window. It corresponds very much to the property sheet in Star except that all Filing attributes are or can be made visible. No change to the file is effected until the "Apply!" command is invoked. The "Abort!" command averts all changes. Attributes windows can be made tiny but will crash if deactivated and later reactivated. Unlike directory windows, making an attributes window tiny will not cause any of the fields to be reevaluated. Also, attributes windows are dependent on directory windows, so they are destroyed when the directory window to which it applies is closed. Almost all attributes can be examined. A few cannot but they can be deduced (isDirectory – – deduced by seeing directory attributes and sizeInPages – – deduced from the sizeInBytes attribute (the "Pages" value is really the subtreesize attribute. When set it indicates the sum of the sizeInBytes attributes or a file and all its descendents). A few attributes can not be seen completely. Position attributes and uninterpreted extended attributes are only displayed for 6 words. If necessary that value can be changed in the debugger by changing a global variable in FSWindowToolCImpl. Lastly, there are those attributes like Type and Extended Attribute Type which are only known by the filesystem as a number. Only the common values of these attributes are known by FSWindowTool. Consult the Filing Programmer's Manual in the Services 8.0 Programmer's Guide if you are not familiar with Filing attributes.

The following attributes can be seen and changed (subject to access control):
Name, Version, Type, Page Limit, Checksum, Access List, Default Access List, Directory Ordering, Uniquely Named Children and extended attributes whose interpretation is one of {boolean, string, cardinal, long cardinal, integer, long integer}.

The following attributes can be seen but not changed:
SizeInBytes, SubtreeSize, FileID, ParentID, CreatedOn/By, FiledOn/By, ReadOn/By, BackedUpOn, extended attributes whose interpretation is one of {none, name list} (interpretation {time} is not supported).

Of these attributes which cannot be changed, the following are not allowed to be changed by Filing: SizeInBytes, SubtreeSize, FileID, FiledOn/By, and ReadOn/By.

-- HeapCheckImpl.doc

HeapCheckImpl implements a layer on top of the Pilot Heap mechanism. This version is modified from the version of HeapCheckPack.mesa, which was in turn stolen from Star. This version is named HeapCheckImpl instead of HeapCheckPack to distinguish it from those modules, and was modified to correct a few bugs and make it work with the HeapCheckTool. HeapCheckImpl has several built in facilities for heap debugging that have not been tested in this version; see the section entitled 'Untested facilities' for more details.

General info

HeapCheckImpl implements a layer on top of the Pilot Heap mechanism that supports owner checking on all heaps and gathers statistics on heap usage for all heaps. HeapCheckImpl also keeps a linked list of all heaps. The HeapCheckTool provides a tool interface to this information.

Normally, variables of type UNCOUNTED ZONE are really of type HeapInternal.UncountedZoneRep. This record contains a pointer to the HeapImpl procedures that implement NEW and FREE, and a pointer to some data that describes the parameters of the heap. With HeapCheckImpl installed above HeapImpl, variables of type UNCOUNTED ZONE are actually of type HeapCheckImpl.UncountedZoneRep. The existing NEW and FREE procedures are replaced with calls to HeapCheckImpl.MakeNode and FreeNode, and Heap Create and Delete calls call HeapCheckImpl instead of HeapInternal. The HeapCheckImpl.UncountedZoneRep also contains a pointer to the real heap, which actually does the request Heap operation after HeapCheckImpl has gathered a few statistics.

Other tools that operate on heaps, such as DebugHeap, will not accept the HeapCheckImpl.UncountedZoneRep handle that a variable of UNCOUNTED ZONE points to. These tools want a HeapInternal.UncountedZoneRep, which can be obtained by using the HeapCheckTool or using the debugger to execute (<heapcheck handle>%(HeapCheckImpl.UncountedZoneRep)).underlying .

The global variable head in HeapCheckImpl is a linked list of all the normal and uniform zones. mdsHead points at the list of MDS zones. zoneCt is a count of the number of all heaps currently in existance.

Note: currently, HeapCheckImpl allocates a 26 word node from the system heap for each Heap that gets created. HeapCheckImpl should probably be changed to allocate these nodes from it's own zone.

Building HeapCheckImpl into a boot file

Augmenting the existing HeapImpl with HeapCheckImpl requires building HeapCheckImpl into the bootfile, and using the bootfile main config to export the Heap procedures that HeapCheckImpl replaces. As an example of how to do this, BWSPerf.config is included at the end of this file. This example is from the Basic Workstation, and also binds SpaceCheckPack, which provides debugging information about the Space mechanism.

Untested facilities

HeapCheckImpl provides facilities for 'protecting' nodes and breaking to the debugger if a node is freed by a module other than the one that allocated it. These facilities have not been tested in this version of HeapCheckImpl. By default, HeapCheckImpl does not do this checking. HeapCheckImpl exports HeapCheckDefs.mesa, which provides access to some of these facilities.

HeapCheckImpl does owner checking on all modules by default. HeapCheckImpl also has a variable called 'releaseMode', according to the notes at the top of the module, if releaseMode is true:

- -- Client sees zone handles that are Pilot objects created by HeapImpl
- -- Client calls to NEW/FREE go directly to HeapImpl; to Heap.Make/Free, thru HeapCheckPack
- -- NodeHeader records are not used; nodes have client - specified length.

If releaseMode is false (the default), everything goes through HeapCheckImpl.

Example bootfile main config with HeapCheckImpl and SpaceCheckPack:

-- BWSPerf.config


```
BWSPerf: CONFIGURATION
  IMPORTS
    STBOps,
```

-- FTPs.doc

FTPs is a version of FTP that runs in the exec with an additional switch for the list command, /s, that causes ONLY the subdirectories to be listed, e.g.

FTPs ServerName dir/c Mesa li/s '*

FTPs is registered with the exec when FTPs is started.

This is FTP which has all the normal FTP functions. So you should be able to delete FTP.bcd and replace it with FTPs.bcd. You may invoke it by typing FTP or FTPS as long as you do not have FTP.bcd on your disk.

END..

-- HackedTajoFont.doc

HackedTajoFont.strike is a somewhat -- modified version of TajoFont.strike, the standard font that comes with Tajo. HackedTajoFont differs in the following ways:

- The control characters are visible. Mostly they are a little up -- arrow followed by a little letter. Exceptions: escape is a little "ESC"; ↑G is a cute little bell shape; ↑A and ↑B are unchanged, except that...

- ↑A is moved one bit to the right.

- Percent (%) is slightly prettier.

- The single -- quote / apostrophe character (') is asymmetrical. Perhaps this looks nicer -- perhaps it doesn't. It was necessary because...

- The back -- quote character (`) was made visible. It's a reflection of '. For those of you who have not fully explored your keyboards, you can type ` with a shift -- minus.

- Lower -- case m is prettier. The standard m is incredibly ugly if you look at it closely.

- Delete shows up as a gray rectangle. A number of different terminals use this convention, so I figured why not.

Besides its purely aesthetic improvement, HackedTajoFont is useful for reading mail from outside of Xerox. Unix types in particular seem to derive sadistic pleasure from quoting things with ``".

To use HackedTajoFont, retrieve the file and put the following line in the [System] section of your User.cm:

Font: HackedTajoFont.strike

I want a new font
  One that looks like it should
One that shows my back -- quotes right
  And makes my mmmmm's look good.

One that won't make me nervous
  Wondering what I typed
One that let's me see what's
  In each and every byte...
  Sending you those bytes.

```
    SpecialTIP, SpecialWindow, Window,
    XFormat, XString,
    BeepFace, DisplayFace, CommunicationPrograms,
    HeadStartChain, KeyboardFace, MouseFace, PilotDiskFace, ProcessorFace,
    RealMemory, RuntimeInternal,
    RuntimePrograms, SA800Face, TemporarySetGMT
  EXPORTS ALL =
  BEGIN

  PilotAndCheck: CONFIG
    IMPORTS
      BeepFace, DisplayFace, Environment, HeadStartChain,
      Heap, Inline, KeyboardFace, KernelPrograms,
      MouseFace, PilotDiskFace, ProcessorFace, RealMemory,
      RuntimeInternal, SA800Face, TemporarySetGMT,
      RuntimePrograms, CommunicationPrograms,
      PilotClient
    EXPORTS
      ByteBlt, File, FileExtras, FloppyChannel, ObjAlloc, PhysicalVolume,
      Process, Runtime, Scavenger, Space, Stream, System, TemporaryBooting,
      UserTerminal, UserTerminalExtras, Volume, VolumeConversion, Zone,
      DebuggerSwap, DeviceCleanup, DeviceError, DiskChannel,
      DiskScheduler,
      ResidentHeap, Snapshot, SpecialBooting,
      SpecialFile, SpecialRuntime,
      SpecialSpace, SpecialSystem, SpecialVolume,
      VM, ZoneInternal, DiskDriversPerf, FileBasicsPerf, FileMgrPerf,
      ResMemPerf, SpacePerf, VMPerf, KernelFile, LoadState,
      Heap,
      RuntimeInternal, SpaceCheckDefs =
    BEGIN
    [ByteBlt, File, FileExtras, FloppyChannel, ObjAlloc, PhysicalVolume,
      Process, Runtime, Scavenger, Space1, Stream, System, TemporaryBooting,
      UserTerminal, UserTerminalExtras, Volume, VolumeConversion, Zone,
      DebuggerSwap, DeviceCleanup, DeviceError, DiskChannel, DiskScheduler,
      ResidentHeap, Snapshot, SpecialBooting,
      SpecialFile, SpecialRuntime,
      SpecialSpace, SpecialSystem, SpecialVolume,
      VM, ZoneInternal, DiskDriversPerf,
      FileBasicsPerf, FileMgrPerf, ResMemPerf,
      SpacePerf, VMPerf, KernelFile, LoadState,
      RuntimeInternal] _ PilotKernel[];

    [Heap1, KernelPrograms, SpecialHeap] _ HeapImpl[Heap1, Inline, Runtime, Space, System, VM, Zone,
    Environment];
    [Space2, SpaceCheckDefs] _ SpaceCheckPack[Runtime, Space1, System];
    Space _ Space2 THEN Space1;
    [HeapCheckDefs, Heap2] _ HeapCheckPack[Heap1, Inline, Runtime];
    Heap _ Heap2 THEN Heap1
    END;

STBStuff: CONFIGURATION
    IMPORTS
      Auth, File, Heap, NSFile, NSFileControl, NSFileExtra, NSVolumeControl,
      Process, ProcessorFace, Runtime, XFormat, XString, SpecialWindow,
      SpecialTIP, STBOps, System, UserTerminal, Volume, Window
    EXPORTS ALL
    CONTROL MinimalStarControlImpl = {
    STBControl;
    MinimalStarControlImpl};


  - - kernel stuff
  PilotAndCheck;
  SupervisorImpl;
  VMMapLogImpl;

  Loader;

  FileStreamImpl LINKS: FRAME;

  MemoryStreamImpl;

  STBStuff;

  END..
```

```
    Auth, NSFile, NSFileExtra, NSFileControl, NSVolumeControl,
    SpecialTIP, SpecialWindow, Window,
    XFormat, XString,
    BeepFace, DisplayFace, CommunicationPrograms,
    HeadStartChain, KeyboardFace, MouseFace, PilotDiskFace, ProcessorFace,
    RealMemory, RuntimeInternal,
    RuntimePrograms, SA800Face, TemporarySetGMT
  EXPORTS ALL =
  BEGIN

  PilotAndCheck: CONFIG
    IMPORTS
      BeepFace, DisplayFace, Environment, HeadStartChain,
      Heap, Inline, KeyboardFace, KernelPrograms,
      MouseFace, PilotDiskFace, ProcessorFace, RealMemory,
      RuntimeInternal, SA800Face, TemporarySetGMT,
      RuntimePrograms, CommunicationPrograms,
      PilotClient
    EXPORTS
      ByteBlt, File, FileExtras, FloppyChannel, ObjAlloc, PhysicalVolume,
      Process, Runtime, Scavenger, Space, Stream, System, TemporaryBooting,
      UserTerminal, UserTerminalExtras, Volume, VolumeConversion, Zone,
      DebuggerSwap, DeviceCleanup, DeviceError, DiskChannel,
      DiskScheduler,
      ResidentHeap, Snapshot, SpecialBooting,
      SpecialFile, SpecialRuntime,
      SpecialSpace, SpecialSystem, SpecialVolume,
      VM, ZoneInternal, DiskDriversPerf, FileBasicsPerf, FileMgrPerf,
      ResMemPerf, SpacePerf, VMPerf, KernelFile, LoadState,
      Heap,
      RuntimeInternal, SpaceCheckDefs =
    BEGIN
    [ByteBlt, File, FileExtras, FloppyChannel, ObjAlloc, PhysicalVolume,
      Process, Runtime, Scavenger, Space1, Stream, System, TemporaryBooting,
      UserTerminal, UserTerminalExtras, Volume, VolumeConversion, Zone,
      DebuggerSwap, DeviceCleanup, DeviceError, DiskChannel, DiskScheduler,
      ResidentHeap, Snapshot, SpecialBooting,
      SpecialFile, SpecialRuntime,
      SpecialSpace, SpecialSystem, SpecialVolume,
      VM, ZoneInternal, DiskDriversPerf,
      FileBasicsPerf, FileMgrPerf, ResMemPerf,
      SpacePerf, VMPerf, KernelFile, LoadState,
      RuntimeInternal] ← PilotKernel[];

    [Heap1, KernelPrograms, SpecialHeap] ← HeapImpl[Heap1, Inline, Runtime, Space, System, VM, Zone,
    Environment];
    [Space2, SpaceCheckDefs] ← SpaceCheckPack[Runtime, Space1, System];
    Space ← Space2 THEN Space1;
    [HeapCheckDefs, Heap2] ← HeapCheckImpl[Heap1, Inline, Runtime];
    Heap ← Heap2 THEN Heap1
    END;

STBStuff: CONFIGURATION
    IMPORTS
      Auth, File, Heap, NSFile, NSFileControl, NSFileExtra, NSVolumeControl,
      Process, ProcessorFace, Runtime, XFormat, XString, SpecialWindow,
      SpecialTIP, STBOps, System, UserTerminal, Volume, Window
    EXPORTS ALL
    CONTROL MinimalStarControlImpl = {
    STBControl;
    MinimalStarControlImpl};


  -- kernel stuff
  PilotAndCheck;
  SupervisorImpl;
  VMMapLogImpl;

  Loader;

  FileStreamImpl LINKS: FRAME;

  MemoryStreamImpl;

  STBStuff;
```

The * in the first column indicates that this heap was created since the state was saved. For heaps that have changed, the number of words and nodes are displayed in the form <value from saved state>/<current value>. In this example, when the state was saved, ContextImpl had 18 words allocated in 2 nodes; it now has 63 words in 7 zones. The last line indicates that Heap #34 was deleted since the state was saved. To list all the heaps at this point, turn 'Use Saved State' off.

Suggested use
- - - - - - - - - - - - -

HeapCheckTool can be used for performance analysis or space leak checking. To analyze the amount of space used by a particular operation,

1) Boot client with HeapCheckImpl as part of the bootfile. Get to state where you want to check operation.
2) Go to the debugger; start HeapCheckTool
3) Dump All Heaps for future reference
4) Save State!
5) Proceed back to client and perform operation
6) Interrupt to debugger and turn on 'Use Saved State'
7) Invoke 'Dump All Heaps' to see what has changed.

If you are checking for space leaks, you can use HeapCheckTool in conjunction with DebugHeap. Once HeapCheckTool has told which heaps are losing space, use DebugHeap on one of those heaps to see exactly what is being allocated. (Don't forget that DebugHeap wants the 'underlying' handle given by 'Dump All Heaps!' or 'Dump One Heap!' command.)

Note: currently, HeapCheckImpl allocates a 26 word node from the system heap for each Heap that gets created. HeapCheckImpl should probably be changed to allocate these nodes from it's own zone.

-- HeapCheckPack.doc

HeapCheckPack implements a layer on top of the Pilot Heap mechanism.

General info
- - - - - - - - - - - -

HeapCheckPack implements a layer on top of the Pilot Heap mechanism that supports owner checking on all heaps and gathers statistics on heap usage for all heaps. HeapCheckPack also keeps a linked list of all heaps. The HeapCheckTool provides a tool interface to this information.

Normally, variables of type UNCOUNTED ZONE are really of type HeapInternal.UncountedZoneRep. This record contains a pointer to the HeapImpl procedures that implement NEW and FREE, and a pointer to some data that describes the parameters of the heap. With HeapCheckPack installed above HeapImpl, variables of type UNCOUNTED ZONE are actually of type HeapCheckPack.UncountedZoneRep. The existing NEW and FREE procedures are replaced with calls to HeapCheckPack.MakeNode and FreeNode, and Heap Create and Delete calls call HeapCheckPack instead of HeapInternal. The HeapCheckPack.UncountedZoneRep also contains a pointer to the real heap, which actually does the request Heap operation after HeapCheckPack has gathered a few statistics.

Other tools that operate on heaps, such as DebugHeap, will not accept the HeapCheckPack.UncountedZoneRep handle that a variable of UNCOUNTED ZONE points to. These tools want a HeapInternal.UncountedZoneRep, which can be obtained by using the HeapCheckTool or using the debugger to execute (<heapcheck handle>%(HeapCheckPack.UncountedZoneRep)).underlying .

The global variable head in HeapCheckPack is a linked list of all the normal and uniform zones. mdsHead points at the list of MDS zones. zoneCt is a count of the number of all heaps currently in existance.

In addition, there is a global pointer to an array of global frame handles to exclude as node owner. This is to support ExcludeHeapOwners (another hack).

Note: currently, HeapCheckPack allocates a 26 word node from the system heap for each Heap that gets created. HeapCheckPack should probably be changed to allocate these nodes from it's own zone.

Building HeapCheckPack into a boot file
- -

Augmenting the existing HeapImpl with HeapCheckPack requires building HeapCheckPack into the bootfile, and using the bootfile main config to export the Heap procedures that HeapCheckPack replaces. As an example of how to do this, BWSPerf.config is included at the end of this file. This example provides debugging information about the Space mechanism.

Untested facilities
- - - - - - - - - - - - - - - - - -
HeapCheckPack provides facilities for 'protecting' nodes and breaking to the debugger if a node is freed by a module other than the one that allocated it. These facilities have not been tested in this version of HeapCheckPack; see HeapCheckPack.mesa for information on how to use these features. By default, HeapCheckPack does not do this checking. HeapCheckPack exports HeapCheckDefs.mesa, which provides access to some of these facilities.

HeapCheckPack does owner checking on all modules by default. HeapCheckPack also has a variable called 'releaseMode', according to the notes at the top of the module, if releaseMode is true:
- -- Client sees zone handles that are Pilot objects created by HeapImpl
- -- Client calls to NEW/FREE go directly to HeapImpl; to Heap.Make/Free, thru HeapCheckPack
- -- NodeHeader records are not used; nodes have client - specified length.

If releaseMode is false (the default), everything goes through HeapCheckPack.

Example bootfile main config with HeapCheckPack and SpaceCheckPack:

-- BWSPerf.config


```
BWSPerf: CONFIGURATION
  IMPORTS
    STBOps,
    Auth, NSFile, NSFileExtra, NSFileControl, NSVolumeControl,
```

- - LVSnap.doc

LVSnap registers an exec command that facilitates backup and recovery of your logical volumes. The command takes as an argument the name of the logical volume that you want to save. If you omit this name, it defaults to "<>". The logical volume must be open and readable. The program creates two command files: LVSnapStore.cm and LVSnapRetrieve.cm. LVSnapStore.cm is a command file containing an FTP command to store all of the files on your system volume to a file server. LVSnapRetrieve.cm contains exec commands to recreate your directory structure, and an FTP command to retrieve the files. You need to fill in the file server name in both files before using them.

Before running LVSnap, you should clean up your disk as much as possible by deleting files that don't need to be backed up. The following command line is an example.

Delete.~ *$ *.log *.errlog *.doc *.TIPC *.Mail - TOC *.press *.interpress *.ip temp.* *.temp foo.* *.foo

After running LVSnap, you should fill in the file server name in the two command files, and delete unnecessary lines from them. You should delete references to Debugee.outload and Debugger.outload, and you might be able to delete the entire section saving and restoring TIP files.

Now say "@LVSnapStore.cm" to store your files. You will need FTP.bcd on your disk to do this.

!!! Make sure that LVSnapRetrieve.cm is stored on a file server. !!!

To restore your disk, assuming a completely erased volume, you do the following:
1) install your bootfile
2) boot that volume
3) retrieve FTP.bcd from the release directory
4) retrieve LVSnapRetrieve.cm from wherever it is stored
5) type "@LVSnapRetrieve.cm"

The retrieve command file will retrieve only those snapped files which are newer or don't already exist. If you prefer to overwrite any files of the same names that exist on your disk, you should change "Ret/ua" to "Ret/c" in LVSnapRetrieve.cm.

After you have restored your disk, you should clean up the file server by deleting the stored copies. This is easily accomplished by deleting
[YourFileServer]<YourName>BackupXXX>*
(for the logical volume named XXX).

!!! WARNING !!!

I've noticed that occaisionally the enumeration fails in strange and mysterious ways, resulting in bogus .cm files. Re - executing the command doesn't help, but re - booting and re - trying or executing the command from another volume seems to solve the problem. IN ANY CASE, CHECK YOUR COMMAND FILES CAREFULLY FOR CORRECTNESS.

-- LReal.doc

LReal is a package that maintains numbers as 13 decimal digits of signed manitissa with 10 bits of a power of 10 exponent (– 512 to 511). Number are stored as opaque objects occupying 4 words (64 bits).

All of the routines maintain the numbers normalized, i.e. the first digit is non – zero. The assumed decimal point is after that first digit.

One can create "Special" numbers which will raise an ERROR if used in any arithmetic operation.

For the four arithmetic operations, typical timings (in microseconds) compared with the current Common Software 32 – bit IEEE floating point package:

	LReal	REAL
+ or –	500	800
*	800	1000
/	1500	1900

The special funtions compute in comparable times to those of RealFns. The accuracy of the transendental functions is about 1 in 10^11, which is about as good as we know how to do without greater precision arithmetic for intermediate results.

-- LogOut.doc

LogOut sets your password to the empty string. It registers the command "LogOut.~" with the Exec and is handy for including in the command file you run at night before you go home.

The total size and size distribution of the initial set of local frames is specified in the parameter file passed to MakeBoot. There are two important considerations.

(1) For small frames, the initial quantity and distribution should be such that there are no frames which are never used (but see . The Totals command will show such frames.

(2) For lárge frames, sufficient quantities should be alllocated to cover those sizes which are used very frequently by the system. For infrequently – used sizes, there probably should be none allocated initially.

– – – – – Studing Normally Hidden Local Frames – – – – –

Some initially – existing frames are difficult for LocalFrameTool to find – – it only knows about them if it can find someone using them or find them on the free list. To allow it the best chance of finding these frames before they are allocated for some use which hides them, use Othello's Set Debugger Pointers command and then boot with the "0" (zero) key switch (in additon to your normal boot switches). When you arrive in the debugger with "Key Stop 0", do a Totals command. LocalFrameTool will find all (well, maybe all but one or two) of the local frames and cache this information for the rest of the bot session. Note that Deactivating the tool will flush its database and lose this information.

= = = = = = = = = = = = = = = = = = = ANOMALIES = = = = = = = = = = = = = = = = =

This section describes various anomalies that may be observed in the LocalFrameTool's output.

Frame occurs multiple times!!!!!

This message will occur if a child process has been FORKed but has not run yet. In that case, a debugger List Processes will show a process in module Processes, PC: 37527B, in procedure ForkInternal. The message may occur multiple times from this cause. If this occurs in a circumstance other than this, please contact Dale Knutsen immediately.

Frames of Unknown Usage

There is one frame which appears to the tool to be valid but which MakeBoot failed to put on the free list and hence appears to be in use by an unknown party. This frame has size index 0 and local variables all zero.

Frames which are Never Used

When Pilot finds insufficient space for a requested frame in the page from which it is constructing small frames, it creates frames of smaller sizes in the remaining space if any will fit. Typically these are size index 0. Thus there may be a small excess of the first few frame sizes; if so, they will appear in the "never used" category.

– – – – – End of document – – – – –

The third and fourth lines specify the current usage of frames to be dumped. If you do not mark any usages to be dumped, the tool will respond with "No such frames".

Example output with parameters =, >, 68, words, existed initially, on process stack:

```
Initially existing frames:
size              global
index size address  caller  frame    pc .. local variables ..
  11   80  7140B  11004B  40324B  2621B  40325B    2B  33137B    1B
  18  224  11004B  21144B  40674B  5446B  40325B    2B  33137B    0B
  12   96  12610B  7140B  40324B  1665B  40400B  33405B   177B  40367B
```

In the output, "address" is the address of the local frame. It may be used with Copilot's Display Frame command. Only the first four words of the local variables are displayed, regardless of the actual size of the frame. More can be printed using Copilot's Display Frame or Octal Read commands. Note that "caller", "global frame" and "pc" are valid in procedure and coroutine activation records but are NOT valid if the frame is being used for a large argument or return record, a signal argument record, a swap message, a file message, or any other use in which the frame was obtained by an explicit ALLOC instruction.

In some future version of LocalFrameTool, the "verbose" switch may cause more of the local variables to be printed.

- - - - - Change! command - - - - -

This command changes the free list for a particular frame size to allow or block local frame promotion. It is used for experimenting with various combinations of promotion and non - promotion for different frame sizes. Since this command will seldom be used, it is initially out of sight when LocalFrameTool is instantiated. Scroll the command window upward to get to this command. The parameter items for this command appear in the tool window as follows:

Change! size index = 12 {to promote, to not promote}

For this command, you specify the frame size index for which you want to change the promotion state and the desired state. You can find the size index that corresponds to a particular size from the output of the Totals command.

= USING THE TOOL =

This section describes techniques for using LocalFrameTool to solve particular problems.

- - - - - Crash due to "Out of VM for Resident Memory" - - - - -

The typical cause of this crash is that the system is attempting to allocate local frames endlessly. Possible causes are (1) an unterminating recursive call; (2) in unusual circumstances, Pilot may get into a state where it attempts to send an infinite number of intra - process messages from its Filer and Swapper subsystems. Proceed as follows:

Do a Totals.

Examine the total quantity of Filer and Swapper messages. If there are less than 20 (say) Filer messages and less than 50 (say) Swapper messages, these are probably not the cause.

Examine the total pages of dynamically created large and small frames. In the current version of Pilot, there are a total of 50 pages available to hold these frames. If very many large frames get requested simultaneously, this limit can be exceeded. Dynamically loading single modules with large global frames could also cause this problem. These frames will appear in the totals under "unknown use".

If one frame size has a large total number of frames, do a Process Stacks command. You may find one process in an unterminating recursion.

- - - - - Studying Frame Faults on Large Frames - - - - -

The Process Stacks command will quickly show those procedures on the call stack which have "large" frames. Using CheckFrames.bcd is probably a more systematic way to attack general performance problems related to large frame faults.

- - - - - Determining the Intial Frame Allocation - - - - -

circumstances, Pilot may get into a state where it attempts to send an infinite number of these messages. In that case, the system will land in the debugger with the message "Out of VM for Resident Memory". The summary totals will typically show scores of these messages.

"unknown use" The number of frames whose current use is unknown. That is, they are known not to fall in any other of the usage categories. The most common actual use of these frames are as activation records of coroutines which are not currently on the stack of any process. Local frames which are allocated to contain large procedure argument and return records and signal small argument records will also appear here. In the current version of Pilot, frames of unknown use may contain the global frame of a dynamically – loaded configuration if it consists of a single module and has not been packaged. due to system overhead, the global frame address will be 4 larger than the address of the local frame containing it.

"free" The number of frames which are available for use.

The next column is a special subclass of the "free" category described above.

"never used" The number of frames which are available for use and have never been used at all. This indicates a too – generous allotment of frames of this size in the parameter file which was used to build the debugee boot file.

The next two columns indicate the creator of the frame.

"existed initially" The number of frames which were created by MakeBoot and were present in the system at boot time.

"dynamically allocated" The number of frames which were dynamically created by Pilot in response to a frame fault. For smaller frames, Pilot allocates them permanently. For larger frames, Pilot allocates them for transient use and reclaims their storage when they are free. An asterisk following a count indicates such a transiently created "large" frame.

"are of unknown origin" The number of frames which were created neither by MakeBoot nor by Pilot. There are normally no frames in this category, and in that case the category heading and the category totals do not appear in the summary output.

– – – – – Process Stacks! command – – – – –

This command prints the size (and optionally the identity) of the local frames on all process stacks. It may be used to look for procedures which have large local frames. Example output:

```
PSB  innermost frame size .. root frame size
 20:  28, 28, 20,  8
 65:  16
 66:  16,  8
    ...
101:  12, 12,  8,  8, 96, 80, 224(L:11004B),  8
102:  12
    ...
```

The PSB column gives the process index as displayed by Copilot and as accepted by its Set Process Context command. Following the PSB, the size of each local frame of that process is given, beginning with the innermost activation record and ending with the root frame of the process. For "large frames", the frame address is also printed. If LocalFrameTool's "verbose" switch is turned on, the address of all frames is printed. The frame addresses given may be used with Copilot's Display Frame command.

– – – – – Dump Frames! command – – – – –

This command dumps local frames satisfying user – specified criteria on size, origin, and current use. The parameter items for this command appear in the tool window as follows:

```
Dump Frames! of size < = > 123 {words, size index}
   which  existed initially  were created dynamically  are of unknown origin
   and which are  on process stack  unknown usage  free but were used  free and never used
   and which are  free but were used  free and never used
         on process stack  swapper messages  filer messages  used by unknown
```

The first line contains the command and parameters which specify the sizes of frames that will be dumped. You may specify dumping frames which are smaller, the same, or larger than a chosen size or size index, or any combination of these relations.

The second line specifies the origin of frames to be dumped. If you do not care about the frames' origin, turn all three items on. If you do not mark any origins to be dumped, the tool will respond with "No such frames".

When a local frame is requested but there are no free frames of that size (or any higher size for which promotion is allowed), the processor generates a frame fault. In response to the fault, Pilot supplies a new frame of the requested size. These dynamically – generated frames are classified by Pilot into "small" frames and "large" frames (see below). In the current version of Pilot, small frames are those of size less than about 100 words. This may change in future versions of Pilot.

Small Frame

When Pilot supplies a frame in response to a frame fault and that frame is relatively small, the frame is allocated permanently and exists and occupies resident storage for the duration of the boot session. Thus the quantity of small frames grows in response to system demand and should stabilize at the maximum required.

Large Frame

When Pilot supplies a frame in response to a frame fault and that frame is relatively large, the frame is allocated transiently. When the frame is freed, the storage is reclaimed. Thus requests for large frames may incur more computing costs if the number normally needed is less than those provided when the boot file was built. [Note: Future versions of Pilot are expected to have improved large frame management strategies.]

= COMMANDS =

This section describes LocalFrameTool's commands and their parameters

- - - - - Totals! command - - - - -

This command prints a summary of local frame origins and usage. Example output:

```
size     prom total |process swap file unknown   |never| existed dynamically
index size otes frames| stack  msg msg use  free| used|initially allocated
  0    8 prom  38 |  26   0  0  7   5| 0|  38     0
  1   12 prom  34 |  18   0  0  0  16| 0|  34     0
  2   16 prom  15 |   4   0  0  2   9| 0|  14     1
     ...
 11   80      4 |   1   0  0  0   3| 0|   2     2
 12   96 prom  2 |   1   0  0  0   1| 0|   2     0
     ...
 17  192 prom  1 |   0   0  0  1   0| 0|   1     0
 18  224       1 |   1   0  0  0   0| 0|   1     0
 21  764       1 |   0   0  0  0   1| 0|   0     1*
 Totals:
                      179 |  66   0  0 18  95| 0| 155    24
 Total pages of dynamically created large frames =  3
 Total pages of dynamically created small frames =  4
 Total pages of initially existing frames =  15
 * indicates transiently created "large frame"
```

The items in the summary are explained individually below.

"size index" The PrincOps code for the size of a local frame.

"size" The size in words of a local frame of the given size index. This size includes system overhead of four words per frame.

"promotes" In the summary, "prom" means that frame allocation requests for that size will be promoted; Blanks mean that promotion will not occur for that frame size.

"total frames" The total number of frames of that size.

The next five columns summarize current frame usage.

"process stack" The number of frames currently being used as activation records over all processes. This is the normal use of a local frame.

"swap msg" The number of frames currently being used as intra – process messages from the Pilot Swapper subsystem. In a correctly operating system, this number should be less than 50 (say). In unusual circumstances, Pilot may get into a state where it attempts to send an infinite number of these messages. In that case, the system will land in the debugger with the message "Out of VM for Resident Memory". The summary totals will typically show hundreds of these messages.

"file msg" The number of frames currently being used as intra – process messages from the Pilot Filer subsystem. In a correctly operating system, this number should be less than 20 (say). In unusual

– – LocalFrameTool.doc

= SUMMARY =

LocalFrameTool is a new performance and debugging tool for examining the origins and usage of local frames. This tool may be of interest if you

- o need to debug a crash due to "Out of VM for Resident Memory",
- o are interested in poor performance due to frame faults on "large" local frames,
- o build boot files,
- o are concerned with real memory usage,
- o are interested in studying the performance consequences of different schemes for frame promotion during local frame allocation,
- o wish to study normally hidden local frames e.g. frames of coroutines.

LocalFrameTool runs in Copilot and prints information about the debugee. The program has commands to

- o print a summary of local frame origins and usage
- o dump local frames satisfying user – specified criteria on size, origin, and current use,
- o print the size and identity of the local frames on all process stacks,
- o allow or block local frame promotion.

The tool and documentation are

LocalFrameTool.bcd
LocalFrameTool.doc

= ORGANIZATION =

The sequel is organized as follows: The next section gives a glossary of terms and explains their significance. Subsequent sections describe the available commands and their parameters, techniques for using the tool for particular jobs, and finally a section describing some anomalies and fine points which may be encountered in using the tool.

= GLOSSARY =

Local Frame
: A local frame is a block of storage whose allocation and deallocation is supported by the processor. The most common use for local frames is as procedure activation records. In executing a procedure call, the processor allocates a local frame to contain the local variables of the procedure. The frame is freed automatically when the procedure returns. Local frames are also used to contain large procedure argument and return records and signal small argument records In the current version of Pilot, local frames are also used to hold intra – process messages from the Pilot Filer and Swapper subsystems, and the global frame of a dynamically – loaded configuration if it consists of a single module and has not been packaged. In the current version of Pilot, all local frames are constructed from resident storage – – they do not swap.

Frame Size, Frame Size Index
: As specified in the Principles of Operation, local frames only come in a particular set of sizes, ranging from 8 to 4092 words. A frame size index is an index into the array of sizes PrincOps.frameSizeMap, and is the way that the size of a frame is specified to the processor.

Allocation Vector
: This is a processor data structure which keeps track of free local frames. It is organized as an array of lists of free frames, one list for each frame size.

Promotion
: When the program generates a request to allocate a local frame, the processor goes to the Allocation Vector's list of free frames of the required size. This list may be marked so that if it is empty, the processor will continue looking for a free frame in the list of free frames of the next higher frame size; if a free frame is found there, it will be allocated and the extra space "wasted" until the frame is freed. This is referred to as "frame promotion". Alternatively, if the list is not marked for promotion, the processor will immediately generate a frame fault, and Pilot will supply another local frame.

Frame Fault

-- LoadStateTool.doc

This tool is a Tajo window used to look at the load state, i.e. modules currently loaded. It runs in the client rather than debugger environment.

The tool window consists of a message subwindow, a form subwindow, and a log window. The form subwindow contains two string items: "GF#" and "Name", and four command items: "GF# -->Name", "Name -->GF#", "Name -->Exporter" and "Name -->Importer".

The command "GF# -->Name" will take the octal global frame number/handle and translate it into a Module Name and display it in the log window. If this global frame is a dynamically -- made copy of another frame then the name will be followed by a "*" char. If this global frame has not been started, then the name will be followed by a "~" char.

The command "Name -->GF#" is the reverse of "GF# -->Name". That is, it will look for all occurances of the module name specified in the field "Name".

The command "Name -->Exporter" will take the field "Name" and look for all modules that export that interface. This means all runtime loaded modules, not those in the boot file.

The command "Name -->Importer" will take the field "Name" and look for all modules that import that interface. All modules means all runtime loaded, and all boot file modules that the interfaces were exported.

Names may be wild carded using the same rules as the EXEC (i.e., * and #)

How is this useful? If you want to know what modules are loaded, you can specify "*" for the module name and hit "Name -->GF#". Or maybe, you want to know how an interface is implimented, but don't know where the code is. Find out who exports that interface using "Name -->Exporter". If you want to know who imports a particular interface use "Name -->Importer".

Note: The number displayed when loading a program thru the Executive is not the Global Frame Handle.

Shortcommings --
- Duplicate info will sometimes be displayed.
- This hack currently does not have an EXEC interface, and it does not recognize the ABORT key.
- If you try to convert a nonexistant Global Frame Number into a module name, RunTime.ValidateGlobalFrame will address fault.

Future enhancements --
- It would be nice to have it enumerate the EXPORTS and IMPORTS of a named module.

-- ListWindows.doc

ListWindows displays window information for each window that is not inactive in your current environment. It writes a table into the file Window.boxes containing the name of the window, the bitmap relative window box and the bitmap relative position of the tiny form of the window. It also indicates the state (Active or Tiny) of that window.

If the window is a FileWindow, then the name shown in the table is prefixed with an asterisk ('*).

Sample contents of Windows.boxes:

[Executive]
WindowBox: [x: 0, y: 542, w: 512, h: 206]
TinyPlace: [x: 196, y: 778]
InitialState: Active

[CommandCentral]
WindowBox: [x: 0, y: 542, w: 512, h: 206]
TinyPlace: [x: 196, y: 778]
InitialState: Tiny

[Hardy]
WindowBox: [x: 0, y: 30, w: 512, h: 512]
TinyPlace: [x: 196, y: 748]
InitialState: Tiny

[Debugger]
WindowBox: [x: 512, y: 30, w: 512, h: 512]
TinyPlace: [x: 708, y: 748]
InitialState: Tiny

[HeraldWindow]
WindowBox: [x: 0, y: 0, w:1024, h: 30]
TinyPlace: [x: 482, y: 778]
InitialState: Active

[Activity]
WindowBox: [x: 420, y: 0, w: 184, h: 30]
TinyPlace: [x: 482, y: 778]
InitialState: Active

-- ListHacks.doc

ListHacks produces a listing of all hacks on a directory. For each hack, it lists its:

name
location and create date
"author" (actually, the last writer)
documentation location and create date (if there is any)
up to 1000 characters of the documentation.

It brings up a Tool window with string items for the hack and hack documentation locations. For each Bcd found on the hack directory, a search is made for a corresponding ".doc" file on the documentation directory. It produces the listing on the file Hacks.list. ListHacks runs at background priority in a process separate from the notifier.

- - ListFiles.doc

ListFiles is a program to permit you to quickly locate the space hogs on your disk. It always deals with sizes in pages, so you don't have to try to parse huge byte sizes the way you do with FileTool, or search through gobs of useless UID's, read dates, etc., the way you do with FileStat. Command syntax is

ListFiles.~ template minSizeInPages

"template" is applied against the current search path, and all files whose size in pages is > = minSizeInPages are listed in the Exec window, followed by their size in pages.
If "template" is followed by any switch, then only the totals will be given - - usefull if you think a large number of small files are eating up space.

Examples:

(1) list all mail files of 100 pages or more:

Listfiles '*.mail 100

(2) List all files on the current search path, with their size in pages:

Listfiles '* 0

(3) Give only the total number of files and total pagecount in the <>mail>old> subdirectory:

Listfiles <>mail>old>'*/t 0

NOTE that you must quote asterisks and pound signs to keep them from being expanded by the Executive.

-- ListDrawers.doc

OBJECTIVE:
Given a file service and a group name, list all the drawers accessible by that group.

INVOKING THE PROGRAM:
After having retreived the files
ListDrawers.bcd
ServicesStubsConfig.bcd
invoke the program by
Run/v ServicesStubsConfig ListDrawers
{ignore the version mismatch messages}

METHOD OF OPERATION:
In the form subwindow of the tool, there are two entries.
FILE SERVICE: Fill in the name of the file service you wish to deal with. Default domain and orgainations will be provided by the tool. This name should be unique and non – ambiguous.
GROUP NAME: Fill in the name of the group you wish to search for. As an added feature, this entry will also accept individual names. Default domain and orgainations will be provided by the tool. This name should be unique and non – ambiguous.
DOIT: When all the data is entered, invoking this command will perform the objective specified above.

-- EraseVolume.doc

EraseVolume allows a user to invoke the erase volume command from Othello from Copilot (or any other volume). This hack has a window interfaces and an Executive interface.

If you type "EraseVolume Star" into the Exec, Star will be erased.
If you type "EraseVolume " into the Exec, a window interface of the tool willappear. There are two elements in this window, Volume: , the name of the volume to be erased, and Erase!, which starts the erasure.

Hints:
Do not erase a volume that is on anybody's search path. This is surely to cause problems. Below is a suggested way to get extra disk space.

```
-- CleanStar.cm
SetSearchPath <CoPilot>
CloseVolume Star
EraseVolume Star
OpenVolume Star/w
CreateDir <Star>Temp
SetSearchPath <CoPilot> <Star>Temp
```

```
    ENDLOOP;
```

Error is raised if any of the obvious termination conditions arise. Some day I may add a signal Stable that gets raised if the position in oldBoard is "uninteresting".

The NoteChangeProc is called with your client data, the Row and Col of the cell that chenged state, and whether or not it is NOW living.

Implementation details:
LifeImpl is basically a 64 state finite state machine. Consider the current 9x9 of interest:

```
  abc
1 xxx
2 xxx
3 xxx
```

the current state is defined by the cells in columns a and b. the input to the state machine is column c. The output is the tuple (newState, changed, living). New state is just columns b and c, changed and living are defined by the rules of Life. I keep a row cursor which is the row number of "row 3" in the diagram, and a column cursor which is the column number of "column c" in the diagram. Calculating the input is the bulk of the algorithm. The algorithm is linear in the number of living cells plus the number of rows with living cells in them. The constant of proportionality is roughly three (since I have to look at each living cell three times, once for the row before it, once for the row it's in, and once for the row following it), the fixed overhead per generation is quite low.

Note that life is completely symmetric so the choice of which are rows and which are columns, and which direction is "increasing" is completely arbitrary.

IV – The rules of "Life"

Life is played on a rectilinear grid. Each position has eight neighbors.

A cell dies if it has 0 or 1 neighbor (underpopulation)
A cell lives if it has 2 or 3 neighbors
A cell dies if it has 4 or more neighbors (overpopulation)

A new cell is born in any empty space with exactly three neighbors.

So for example the position "blinker":

```
   abcde
1 00000
2 01110
3 00000  (where 0 is a dead position and 1 is a live cell)
```

becomes

```
   bcd
0 000
1 010
2 010
3 010
4 000
```

a2 dies (underpopulation)
b2 lives (2 neighbors)
c2 dies (underpopulation)

c1, c3 are born.

etc.

V – Programming Notes

For those of you who might be interested in implementing Life with another user interface, here are some notes.

A Board (position) is a sequence of Rows in ascending order, ending with an empty Row. A row is a Row Entry followed by a sequence of Col Entry's in ascending order.

Generate takes a Board that is the current position (oldBoard) and a Board to put the new position into (newBoard), a procedure to call for each cell that changes state (noteChange), and client data to pass to noteChange. oldBoard[0] is the Row Entry of the first row, oldBoard[oldBoard.LENGTH – 1] is the Row Entry of the empty row at the end. newBoard[0] is where to put the Row Entry of the first Row of the new position and newBoard.LENGTH is the maximum number of entries that you can put into newBoard. newBoard[newMaxIndex] is the Row Entry of the empty Row at the end of the new position. cells is the number of living cells in newBoard.

the following loop can be used to generate new positions until a termination condition arises:

```
DO
   temp: BoardHandle;
   [maxIndex, cells] ←
      Life.Generate[
         oldBoard: DESCRIPTOR[board, maxIndex + 1],
         newBoard: DESCRIPTOR[nextBoard, boardSize],
         noteChange: NoteChange,
         cd: graphSW
         ! Life.Error = > {
            SELECT code FROM
               dead = > Put.Text[msgSW, "\NDead..."L];
               boardTooSmall = > Put.Text[msgSW, "\NToo many cells for board..."L];
               atEdge = > Put.Text[msgSW, "\NAt edge..."L];
               ENDCASE;
            GOTO exit}];
   temp ← board; board ← nextBoard; nextBoard ← temp;
   REPEAT
      exit = > {};
```

2) The position exceeds the storage allocated to it (free disk space).
3) The position exceed the boundaries of the board (currently 32767 x 32767)
4) The position dies completely.
5) The Life Window is deactivated.

The position CAN be edited dynamically.

III – Input Format for the "Load" command

These are the "official" Life Commands from Gosper's Life program at Stanford. Thanks to Don Woods for snarfing them for me. I suspect small glitches in it and have interpolated what I thought were missing characters. Local modifications follow the complete list.

Life (Virtual machine) Commands

P	Advance one generation
D,E	Enter input mode, D also clear screen.
	. Insert point at center and move center right.
	n. Do . n times.
	n<sp> Move center right n places.
	x,y<delim> Insert pt. at x,y (relative to center).
	x,yD Delete pt. at x,y.
	x,yC Move center to x,y.
	nD Delete n pt. right (or left).
	n<cr> Set x to 0 and add n to y.
	n← Go left n places.
	n↑,n<lf> Figure it out for yourself.
	nX,nY Move X (Y) n positions.
	n# Set generation no. to n.
	E Leave input mode.
I,O	Select file for input,output
W	Write current screen pattern onto output file.
C	CLOSE disk output
nR	Read n'th pattern from input file.
	Print generation no. and no. of points.
	Print scale and (if nonzero) shift factor.
S	n>0 sets scale to n. *n!0 set shift factor to n.
X,Y	Prints X(Y) – coordinate of focal pt.
nX,nY	Adds n to focal pt.
n↑X(!),n↑Y(!)	Set focal pt. to n.
	XORG←–XOFF, YORG←–YOFF, Print XORG and YORG
<cr>,<lf>,<ff>	No – ops
;,:	Comment, ignores text to <cr>
Q	QUIT Exit, return screen to normal
nP	if n>0, proceed n steps. otherwise |n| – k steps
nG	set current generation to n

Local modifications:

There is no "edit" mode, what I have implemented is the following subset:

Note: lower case letters are used as non – terminals, anywhere an uppercase letter is used, its lowercase equivalent is allowed.

X,.	Insert point at center and move center right
O,<sp>	Move center right
;	Comment, ignores text to <cr>
D	Clear the screen, reset center to middle of board
E,<eof>	Terminates inputting a pattern
R,<cr>,<lf>,<ff>	No – ops
n.	Do "." n times
n<sp>	Do "<sp>" n times
nD	delete point n to the right (or left)
n<cr>	center ← [row: center.row + n, col: 0];
n←	center.col ← center.col – n (**BUG** currently adds n instead)
n↑	center.row ← center.row – n (**BUG** currently adds n instead)
n<lf>	center.row ← center.row + n (**BUG** currently subtracts)
nX	center.col ← center.col + n
nY	center.row ← center.row + n
nG,n#	generation ← n
nS	scale ← n (currently only 1, 2, 4, 8, and 16 implemented, others go to next higher value, or 16)
x,yD	Delete point at (col, row) x,y
x,yC	set center to x,y
x,y.	Add point at x,y

- - Life.doc

I - Overview

This document is divided into five parts

I - Overview
II - The tool
III - Input Format for the "Load" command
IV - The rules of "Life"
V - Programming Notes

LifeWindow.bcd is an implementation of John Horton Conway's cellular automata simulation "Life". See Martin Gardner's "Mathematical Games" of sometime in the late 60's for a full explanation.

II - The tool

LifeWindow will bring up a scrollable graphic window for the cell display. You add cells with Point and delete them with Adjust. The "Life" menu attached to this window has six commands: Go, Next, Shrink, Magnify, Clear, and Load.

Go:
Run the simulation, updating the display as you go.

Next:
Display the next generation.

Shrink:
Reduce the cell size by a factor of two.

Magnify:
Increase the cell size by a factor of two.

Clear:
Erase the board and re - center it.

Load:
Load the board from the current selection. There are some interesting positions in >Source>Life>*.life. Select in the file starting at any "D" and ending just before the next one, then use this command.

You can bring up a property sheet (FormSW) by hitting [Control] otherwise known as [Prop's]. The property sheet is destroyed when the Life window is deactivated, or you hit [Prop's] again. This FormSW has the following items in it: Close, Dump, File Name, Generation, Cells, and Scale.

Close:
Closes the output file.

Dump:
If the output file is open, it appends the current pattern to it. If there is no output file open, and no name for one, it dumps the pattern to the msgSW, otherwise it opens the log file named in File Name, and outputs the current pattern to it.

File Name:
String item to specify the file name for Dump, if an output file is open, it will be the name used to open the file.

Generation:
Generation number, incremented for each new generation. Can be set manually.

Cells:
Number of living cells in the current generation.

Scale:
Set the size of the individual cells.

Generating will stop under the following conditions:

1) The [STOP] key is pressed while the cursor is in the Life Window.

-- KineticFractal.doc

KineticFractal is yet another DMT variant. This one's pretty strange. It has a couple of pop – up menus to play with. It's got sound effects!

- - KeyJump.doc

KeyJump.bcd implements nine new editor commands: (these work in the window the cursor is currently in)

COMMAND	ACTION
1. JumpLast	Jump to last page of text
2. JumpForwardHalf	Jump forward 1/2 page of text
3. JumpForwardFull	Jump backward full page of text
4. JumpBackHalf	Jump backward 1/2 page of text
5. JumpBackFull	Jump backward full page of text
6. ScrollForward	Continuous Forward Scroll
7. FastForward	Continuous Fast Forward Scroll
8. ScrollBackward	Continuous Backward Scroll
9. FastBackward	Continuous Fast Forward Scroll

These new commands can be programmed into any key action using TIP tables. To install this hack, do the following:

1. Retrieve KeyJump.bcd
2. Execute command: KeyJump
3. If the below commands seem to work on the wrong window, type into the Exec: KeyJump.~

The following key action do the commands:

Keys	Command
1. Shift JFIRST	JumpLast
2. MENU	JumpForwardHalf
4. Shift MENU	JumpForwardFull
3. SCROLLBAR	JumpBackHalf
5. Shift SCROLLBAR	JumpBackFull
6. RESERVED	ScrollForward
7. Shift RESERVED	FastForward
8. CLIENT1	ScrollBackward
9. Shift CLIENT1	FastBackward

KeyJump registers two Exec commands: KeyJump.~ and TogglePriority.~.

KeyJump.~ initializes the keys JFIRST, MENU, SCROLLBAR, RESERVED, and CLIENT1 to be real estate events rather than input focus events. Should the above keys act like input focus events, KeyJump.~ will re - initialize those keys.

TogglePriority.~ toggles the priority of the process that is doing the scrolling. Initiall the scrolling process is a Process.priorityBackground, and toggling will result in a Process.priorityNormal priority.

The Exec's Unload.~ command will cause the keys JFIRST, MENU, SCROLLBAR, RESERVED, and CLIENT1 to do nothing. This is not recommended because there is a space leak from the KeyJump's TIP table not being cleanly destroyed.

If Keyjump seems to be doing its actions to the wrong window (ie. actions go to the insertion point rather than where the window the cursor is in) type KeyJump.~ into the Exec and the key action should go to the correct place .

To program these commands to other key actions (For TIP wizards only):

The right hand sides of the SELECT arms should be: (in <>TIP>KeyJump.TIP)

1. {COORDS JumpLast}
2. {COORDS JumpForwardHalf}
3. {COORDS JumpForwardFull}
4. {COORDS JumpBackHalf}
5. {COORDS JumpBackFull}
6. {COORDS ScrollForward}

*7. { n COORDS ScrollForward}

8. {COORDS ScrollBackward}

*9. { n COORDS ScrollBackward}

* n is a NATURAL number greater or equal to 1. It corresponds to the number of lines that are scrolled at a time to give the illusion of faster scrolling. The TIP table provided by KeyJump has n = 2. Thus in KeyJump.TIP, the right hand side for FastForward is { 2 COORDS ScrollForward}.

The left hand sides can be whatever your heart desires.

Real estate events:

KeyHacks.bcd, when run converts the following keys to real – estate, rather than insertion – point events. (In other words, the window containing the cursor gets notified about transitions of these keys rather than the window with the insertion point.) Keys: SCROLLBAR, RESERVED, CLIENT1, ATTENTION, HELP. This is in addition to the system default real – estate events.

User.cm:

KeyHacks looks in the [KeyHacks] section of User.cm for two items.

ReformatTime: TRUE | FALSE | YES | NO

If this item is TRUE or YES, then the Time atom will use a different format: "83 – Jun – 02 6:23 pm" in place of " 2 – Jun – 83 18:23:02".

Extensions: mesa form/n cm config log ...

Each of these extensions will be tried when looking for a file for LoadFile, InsertFile, or NewWindow. The LoadFile and InsertFile items will do an automatic NEXT (or SKIP) if the extension that is used is followed by a /n (or /s).

Re – Parse:

KeyHacks provides the ability to dynamically alter KeyHacks.TIP. This is done via private interfaces and involves a space leak the size of a tip table. This command is usually used only when debugging a new KeyHacks.TIP and is not recommended. The command is "ReparseKeyHacks.~" and requires confirmation. The command also re – parses User.cm; this does not leak any space and is done prior to asking for confirmation about the TIP re – parse.

Sample TIP table to attach to textsw (run UserTip.bcd and have "textsw: TextSWKeyHacks.TIP" in the [TIP] section of User.cm):

```
– – TextSWKeyHacks.TIP

– – Copyright (C) 1984 by Xerox Corporation. All rights reserved.

– – LineDelete on Shift – ESC, LineSelect on Cmd – Shift – ESC, WordSelect on Cmd – ESC

COMPLETE Down WHILE LeftShift Down | RightShift Down | COMMAND Down = >
   SELECT ENABLE FROM
      COMMAND Up = > Line, GrowPrimary, Replace;
      LeftShift Down | RightShift Down = > Line, GrowPrimary;
      ENDCASE = > Word, GrowPrimary;

– – BecomeInputFocus on CTRL – Adjust

Adjust Down WHILE CONTROL Down = > BecomeInputFocus;

ENDCASE...
```

tied up waiting for it.

```
Escape Completion:
  Escape Complete :: = ExpandESC
```

ExpandESC is useful for Editing Mesa source files. The basic idea is to expand many of the Identifiers that would be typed in a Mesa program, by looking on the disk for filenames or looking up identifiers found in symbol tables much the way ShowType does. ExpandESC operates on the token to the left of the insertion point and it wants to expand either the "Interface" part or the "Item" part of "Interface.Item". Which strategy ExpandESC uses depends on whether there is a dot in front of the token you are expanding. If ExpandESC thinks you are expanding the "Interface" part, it looks on the searchpath for a file starting with the token preceeding the insertion point. If it thinks you are expanding the "Item" part, it looks for a file called "Interface.bcd" and then peruses it for an expansion for the token. ExpandESC expands file names as in the SimpleExec except that no extension is included. For example, if SpecialMFile.bcd, SpecialSpace.bcd, and SpecialVolume.bcd are on the disk, "Spec" + <complete> will expand to "Special", and "SpecialM" + <complete> will expand to "SpecialMFile", but not "SpecialMFile.bcd".

Note: First, ExpandESC only works in TextSWs. Second, since the SimpleExec uses the complete key for epanding, ExpandESC has no effect in the SimpleExec, unless it is assigned to a key other than the complete key. Third, ExpandESC will not expand interface items in Tajo unless you run SymbolPack.bcd/-s SymbolCache.bcd. It will, however, ask you to load it, if it thinks you haven't.

```
Changing window positions
  Change Window Position :: = Topper;
```

Topper brings a window that is obscured by another to the top, and sends windows that are on top to the bottom (Equivalent to clicking Point in the left or right section of the name frame).

```
Sending type-in to an Exec
  Exec Gets Focus :: = ClearFocus;
```

ClearFocus clears the current input focus. If there is no "backstop" focus (the default situation), then KeyHacks looks at all active and tiny tools trying to find an Executive; if it finds one, it is made the backstop focus. Thus the usual effect is that invoking this TIP atom sends subsequent type-in to your Exec, until you make a new selection.

```
Converting characters to/from octal codes
  Convert character :: = ConvertChar;
```

ConvertChar takes the current selection and converts it to and from a three digit octal code. If the current selection is a single character, it is replaced with the octal code for that character. If the current selection is three octal digits it is replaced with the character for that code. If the selection is longer than three characters, or contains a non octal digit, it is first shortened to the first character of the selection. If there is no selection, the character just before the insertion point is converted. If there is no character in front of the insertion point, or there is no insertion point, or the window is not a text sub-window, it is an error.

```
Changing the input focus
  Window Gets Focus :: = BecomeInputFocus;
```

The window receiving the input is made the focus for later input. Thus, if this atom is generated by a real-estate event, the window containing the cursor will be notified of subsequent insertion-point events. This is very similar to clicking Adjust in a window, except that it avoids the side effect of extending the highlighted selection if the window happens to contain it. Note: This atom is parsed by the TextSW tip manager, so it should not appear in KeyHacks.TIP. Either put it in TextSW.TIP or use UserTip.bcd to push an extra tip table onto textsw. (See sample below.)

```
Expanding a text selection
  Expand Selection :: = <Level>, GrowPrimary;
  Level :: = Word | Line | Document;
```

The selection is grown to both the left and right until it reaches a boundary of the indicated type. For example: "Line, GrowPrimary, Replace" would delete the lines that contain the current selection. If there is no selection, a selection is begun at the insertion point. Note: This atom is parsed by the TextSW tip manager, so it should not appear in KeyHacks.TIP. Either put it in TextSW.TIP or use UserTip.bcd to push an extra tip table onto textsw. (See sample below.)

```
Reducing the level of multiclicking in a text selection
  Reduce Selection Level :: = PrevSelEntity;
```

If there was a text selection at the word level, it is changed to be at the character level; i.e., subsequent Adjust actions will select characters instead of words. If the selection was in lines, it is changed to words; if it was the whole window, it is changed to lines. Note: This atom is parsed by the TextSW tip manager, so it should not appear in KeyHacks.TIP. Either put it in TextSW.TIP or use UserTip.bcd to push an extra tip table onto textsw.

-- KeyHacks.doc

KeyHacks is a program that implements a number of commonly desired "hacks" that you can put under keys. It uses KeyHacks.bcd as a TIP interpreter and KeyHacks.TIP as a tip table to specify the desired features. The features that you can put on keys are:

Menu commands, Screen/Display state, Load/Insert files, Time,
Creating fileWindows, Exec commands, Escape completion,
Changing window positions (top/bottom), Sending type-in to an Exec,
Converting characters to/from octal codes, Changing the input focus,
Expanding a text selection, and Reducing the level of multiclicking in
a text selection.

The last three features are available only in text subwindows, and are invoked not through KeyHacks.TIP but rather through TextSW.TIP. (If you don't want to edit TextSW.TIP, you can use the UserTip hack to push an extra tip table onto text subwindows.)

Menu commands:

Menu Command :: = Menu, <Menu Name>, <Item Name>
Menu Name and Item Name can be either strings or atoms.

Will invoke item <Item> in menu <Menu> on the window for this event (the window with the insertion point for most events, or the window the cursor is in for "real-estate" events [see below]). If the menu isn't on that window, KeyHacks will try to find a menu with that name on the rootWindow.

Screen state:

Screen State :: = <Optional state>, Screen
OptionalState :: = White | Black | Toggle | <empty>

Sets the background color of your screen. <empty> is the same as Toggle.

Display state:

Display State :: = <Optional state>, Display
OptionalState :: = On | Off | Toggle | <empty>

Sets the state of your display. <empty> is the same as Toggle.

Loading and Inserting files:

Load File :: = LoadFile
Insert File :: = InsertFile

LoadFile will attempt to load a file into a TextSW. It first tries to use the current selection as a file name, then the contents of the sub-window, then the start of line to the insertion point. It replaces the contents of the sub-window with the contents of the file. Note: It does not convert a scratch "empty" window into a file window, it just loads the text into the window. This can be useful for loading Mail forms into a mail send tool. InsertFile is similar except that it replaces the file name (rather than the whole window) with the contents of the file.

Creating new windows:

New File :: = <Tiny Place>, <Window Box>, NewWindow
Tiny Place :: = <xCoord>, X, <yCoord>, Y, Tiny,
Window Box :: = <xCoord>, X, <yCoord>, Y, <width>, W, <height>, H,
xCoord, yCoord, height, and width are all cardinals

NewWindow creates a new file window at <Window box>, with a (optional) Tiny Place. It will attempt to use the current selection as the name of a file to be loaded into that window. Note: The window box is not optional. If the tiny place is not specified, the system default will be used. If there is no selection, or the selection is not a legal filename, an empty file window will be created.

Time:

Time :: = Time

Time replaces the current selection with the current time. If there is no selection the time will be inserted. This function works in both Text and Form subwindows. If the [KeyHacks] section of User.cm includes "ReformatTime: TRUE" (or "...YES") then the time will be formatted as "83-Jun-02 6:23 pm" in place of " 2-Jun-83 18:23:02".

Exec Commands:

Exec Command :: = ExecCommand, <command>
command is a string or atom

ExecCommand executes <command> as if had been typed to an Exec. Any output goes to the indirect output sink (default is the Herald, but see IndirectOutput.doc). It is not possible to provide input to the command except in the command string. The command is run in a separate fork so the notifier is not

-- KeepLogin.doc

KeepLogin maintains the password in a location that doesn't get destroyed across world swaps. If it is running in your Tajo, CoPilot and CoCoPilot worlds, it will allow you to login only once as long as the boot button is not pushed. If you log in with no password it will as be cleared in the other worlds as well.

-- KalWindow.doc

KalWindow runs in a 256x256 window, at zero priority. Unlike the original Kal, this program does not allow the user to change its internal state counters. If you want to add any of that, happy hacking.

-- Kaleidoscope.doc

Kaleidoscope is a DMT or Poly alternative which paints a series of kaleidoscopes on your screen. To start, type into the exec: Kaleidoscope

To quit, hit abort or use the window menu.

-- JAccuse.doc

JAccuse lists all files for which XDE file system handles exist. It notes for each handle, the type of handle (MFile, MStream, or MSegment), the access (e.g. readOnly or readWrite), and the name of the program module that acquired the handle. For example,

```
JAccuse.archivebcd
   readOnly  MLoaderImpl
Binder.Log
   readOnly  CommandCentral (stream)
DFTool.log
   log     TextSourcesA (stream)
Debuggee.outload
   readWrite CPTeledebug
```

It is also possible to restrict JAccuse's attention to a specific set of files by naming them on the command line. In general, a file name pattern containing the usual '* and '# wildcards can be specified on the command line; e.g.

```
>JAccuse '*.log
   Binder.Log
      readOnly  CommandCentral (stream)
   Debug.log
      log     CPOutPack (stream)
```

are beats between the source and destination (printer) res – olutions.

Clipping – The bitmap is not clipped to page boundaries at creation. The printer's clipping of a bitmap which spills over a page edge may be unexpectedly ragged. This is because it works by rejecting whole characters and rectangles (the user might otherwise be un – aware of their presence).

Specifying Margin or Scale – because the program tries to recalculate other parameters after any change, you will have a wrestling match if you try to specify margins while the tool is holding scale and visa versa.

- - IPCamera.doc

The Bitmap To Printer Tool.

Purpose: To capture in an Interpress file a rectangular portion of a bitmap. Sources include the display in the same volume or debugged volume.

Window: The tool has 3 subwindows. The first (top) is a msgSW where the tool says some things. The second has to do with the Capture Phase, the third with the Layout Phase and the fourth with the Send Phase of the tool's operation.

Capture Phase - specify a bitmap rectangle:
[1] Choose the bitmap source.
(Only "local" and "client" are now implemented)
[2] Bug "Capture!";
If source is debugee, the entire client bitmap will first be painted over the debugger bitmap else the tool window will simply disappear.
Full screen crosshairs will appear as a cursor.
[3] Position the crosshairs wherever desired for any of the four corners of the rectangle, depressing the left mouse button at any time.
[4] Release the left mouse button to freeze one corner and create a second pair of crosshairs, again tracking the mouse.
[5] Depress and release the left button again to freeze the second crosshair pair. The four lines now define a rectangle without regard to which corners were clicked or in which order. (Bits under the crosshairs are included in the rectangle).
[6] Click the left button again to confirm the rectangle.

Clicking the right mouse button at any time in steps 2 - 6 will "punt" the capture of a new rectangle, leaving the old one in effect. The current bitmap dimensions are shown. If source is debugee, the tool gives no help in repainting the debugger bitmap.

Layout Phase - creating an Interpress file from the current bitmap:
In any order specify choices for the options described below before bugging "Layout!" command:
- name for the Interpress file, to be created in MFile space.
- unit of measurement for margins and scaling (larger units are more precise for setting scale, smaller ones for margins, see below).
- orientation (normal viewing direction of page)
- paper size (from which margins are relative)
- printer type (this lets the tool reorder the bits for the printer's 'easy' scan direction. This is necessary for larger bitmaps on pre - Services 8.0 printers and helpful for all printers)
- Margins versus Scale:
The user chooses either scale or margins to be the independent (specified) variable. If a fixed scaling is not requested the picture will be scaled as large as possible within the limits of the paper size and margins. If a fixed scale is chosen, the image will be scaled about the requested area's center and trimmed at paper edge if necessary. In either case the tool shows the value it computed for the other variable. One always has the option of constraining horizontal and vertical scales to be equal by turning on the "Matched H&V Scales" boolean. Scales which are integral divisors of the printer resolution may work better, e.g. 100, 75, 60, 50, etc. dots per inch for a 300 dot per inch printer (like the 8044/Raven). See remarks.

Remarks:
Sending to the Printer - Use the Print command in the Exec.
Band Overruns - (those dreaded white gaps through the entire image, parallel to the long edge of the paper) are more likely to occur when scalings of more bits/unit are used. They are also increased dramatically by using non - integer scales on Services 7.0 or earlier printers.
Scaling artifacts - will be evident in non - interger scales. These

- - Install.doc

Description:
Install is a utility that install boot files, much as Othello does. It will take given file, either local or remote, and install it as the logical volume boot file of the given volume. Install also lets you specify default boot switches to be put into the bootfile that is being installed. Install cannot be used to install a boot file on the volume on which it is running.

Syntax of command:

Install.~ VolumeName ← FileName/defaultSwitches

Example:

Install.~ Star ← [ServerName]<DirectoryName>VersionNumber>StarDLion.boot/%dwW{

is equivalent to the Othello sequence

Open connection to: Ibis
Fetch
  logical volume: Star
  file: <DirectoryName>VersionNumber>StarDLion.boot
Set boot file default switches
  logical volume: Star
  switches: %dwW{

-- HostGraph.doc

Please refer to the document, "Ethernet Monitors", in XDE Unsupported Software Description.

-- IndirectOutput.doc

IndirectOutput.bcd creates a small window and directs "indirect output" to it. Such output includes "Boot from" messages from the Herald's menu, lists displayed by Adobe Query, etc. Normally these messages are displayed in the Herald window, but if you keep the Herald "tiny" then IndirectOutput can be used to create a more visible sink.

The window defaults to [x: 512, y: 0, w: 452, h: 30], which is the right half of the herald region, minus 60 bits to leave room for a tiny Herald window in the upper right corner. This default can be overridden using the [IndirectOutput] section of User.cm.

-- MakeBoot.doc

Please refer to the documentation in XDE User's Guide.

1

NS Spy tool

This document describes the use of the NS Spy Tool, a tool used for watching traffic on ethernets. Since the tool was designed to watch **ns** packets, the user can filter and format up through level 2 **ns** packets. In addition, **pup** packets may be filtered and formatted up through level 1. Other types of packets may be observed only as raw (level 0) packets, with a minimum of formatting available.

Last edited on February 27, 1984.

1.1 References

[1] *Mesa User's Guide*, Version 10.0, January, 1983.

[2] *Internet Transport Protocols*, XSIS 028112, December 1981.

1.2 Definition of terms

The way a packet is interpreted is dependent on the level of filtering the user has defined. (See the Internet Transport Protocols[2])

level 0 packet — A *level 0 packet* refers to a well formed raw packet from the net. Only the encapsulation has any meaning; the rest of the packets is treated as *level 0 data*, regardless of what it may contain.

level 1 packet — A *level 1 packet* refers to a well formed **ns** or **pup** packet. Meaningful information is the encapsulation and the *level 1 header* information - for **ns** packets this is the IDP header. The remaining part of the packet is treated as *level 1 data*.

level 2 packet — A *level 2 packet* is a **ns** packet with a well known level 2 packet type. This includes spp, packet exchange, routing, echo, error, etc. Meaningful information is the encapsulation, the IDP header information and the *level 2 header* information corresponding to the packet type. The data portion is the remaining part of the packet.

1.3 Hardware

The tool runs on a Dandelion processor with 512 kB memory and a LF display.

1.4 Files

[Idun]<APilot110>11.0>NSTools>Friends>NSSpyTool.bcd.

1.5 User interface

The NS Spy tool has a main window that is divided into four subwindows, plus an auxillary window whose contents depend on the requested level of filtering. See Figure 1.

1.5.1 Main window

When the program is started, the main window is created, which contains the general tool information. This window can be deactivated, and will abort any spying in progress.

Note: If the auxillary window is active when the main window is deactivated, it also will be deactivated.

1.5.1.1 Herald subwindow

The herald subwindow contains the window's name, the *time stamp* of the tool and the network address of the host machine.

1.5.1.2 Message subwindow

The top subwindow is the message subwindow and is used for posting error messages.

1.5.1.3 Flipper subwindow

The flipper subwindow contains seven *flippers*, small alternating boxes with associated labels and counters. The intent of the flipper subwindow is to provide some low-level real-time feedback pertaining to the state of the spying process.

1.5.1.3.1 Flipper items

The Level 0, Level1, and Level 2 flippers indicate the packets observed before any user defined filtering is done.

level 0 This flipper shows the number of well-formed level 0 packets the tool has observed since the tool was last started. Since the tool is spying in promiscuous mode, this is *any* level 0 packets on the entire net (i.e. **pup**, **ns**, **translation**, etc.).

level 1 The level 1 flipper shows the shows the number of well-formed level 1 **pup** and **ns** packets that were observed since the tool was last started.

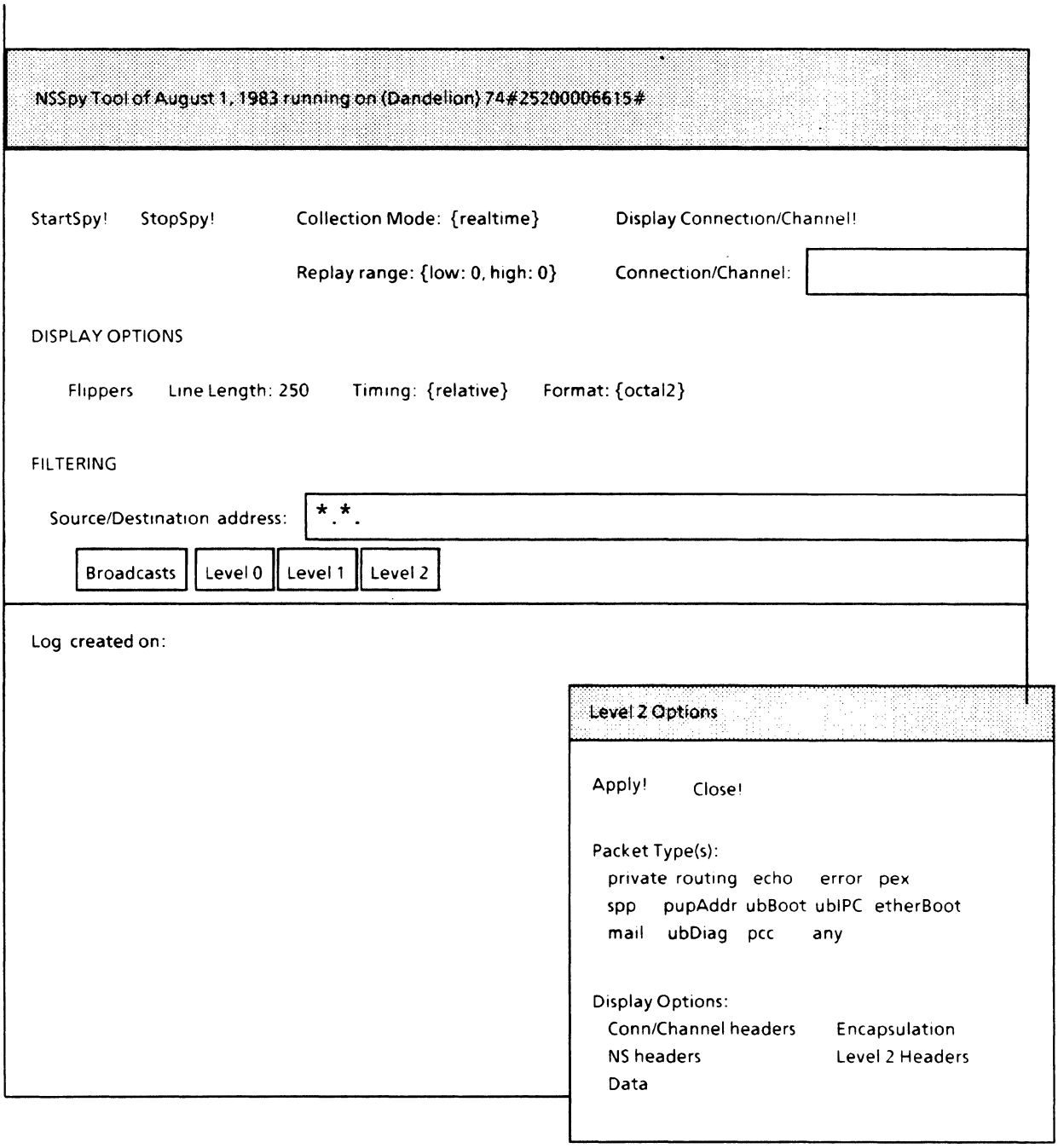


Figure 1

level 2 This flippper shows the number of well-formed level 2 **ns** packets the tool has observed since the tool was last started. The flipper reflects only well known packet types (such as the ones listed in the level 2 auxillary window in the Filtering section of the tool)..

queued **queued** is the number of buffers which have been copied onto on the input queue (after being run though the filters the user has chosen) since the tool was last started. This flipper will only be active when the collection mode is **buffered**.

displayed This item is the number of buffers that have been displayed since the tool was last started. It will only be active in **realtime** or **replay** mode. Note: The tool runs this flipper every time a packet is run through the display code, regardless of the actual display options that have been chosen. If the user has turned off all of the display options, this flipper will still be active in **replay** or **realtime** mode.

1.5.1.4 Control and filtering subwindow

This subwindow contains the operations and options used to run the spy tool.

1.5.1.4.1 Control items

The control section of this subwindow contains items used for controlling the spy tool.

StartSpy! **StartSpy** is a command item that starts the tool spying. It is used to start any of the three collection modes.

StopSpy! **StopSpy** is a command item that stops the spying activity and puts the tool into an idle state. When the collection mode is **realtime** or **buffered**, this command must be explicitly selected to stop the spying; if the collection mode is **replay**, the display will end when the tool has displayed all of the buffers that were collected. Should the user decide he does not wish to display all the collected buffers after starting **replay**, he may stop the display by selecting **StopSpy**. The tool may also be stopped by hitting the stop key. Note: Collecting packets in **realtime** mode is usually easier to stop by using the stop key rather than selecting the **StopSpy** command, as displaying these packets is a very high priority process.

Collection Mode This item enables the user to select the collection mode he wishes. **realtime** will collect packets directly from the net and display them immediately. Packets are not copied and saved; once displayed they cannot be displayed again. **buffered** will collect packets from the net and copy them to the disk for later display. The packets are stored in a large (half of the available volume space or 1000 disk pages, whichever is least) ring buffer. When the ring buffer fills up, the tool will continue to collect, overwriting the oldest packets in the ring with the new packets. **replay**, used after spying in **buffered** mode, will take the packets copied to the disk and display them. Once packets have been collected in **buffered** mode, they may then be **replay**ed any number of times, using different filters. The tool will "clean out" the disk space used for saving the packets whenever the tool is restarted in **realtime** mode or **buffered** mode.

ReplayRange The user may specify the range of the buffered packets he wishes to replay by using **ReplayRange**. **low** is the index of the first packet

he wishes to see; **high** is the index of the last packet he wishes to see.

Line length The length of the display line is specified by **Line length**. This is applicable only to the display of the data, as the header lines are shorter than the minimum line length, 250.

Timings **Timings** is an enumerated item that is used to specify the type of timings displayed in the connection or channel headers. **absolute** timings are absolute (normalized to 0). **relative** timings are relative to when the last packet was observed on the particular connection or channel.

Format The format (**octal1**, **octal2**, **hex**, **decimal**, **ascii**, **ebcdic**) in which the packets may be display is specified by the enumerated item **Format**. Selecting **octal2**, **hex** or **decimal** will display all fields, including data, in that format. Because the header fields are useful only in a numeric form, if **ascii, ebcdic,** to **octal1** is selected, only the data will be displayed in that format - the header fields will default to **octal2**.

Display Connection/Channel!

This command item will display state information about the connection or channel specified in the item below. It is important to note that this command is only applicable after the tool has collected packets in **buffered** mode, as this state information is not saved after displaying packets in **realtime** mode. The user will also not be able to display a channel or connection different than the filter type(s) specified when the packets were collected. For example, if the user specifed the level0 filter, collected a number of packets in **buffered** mode and then attempted to display the state information about a routing channel, an error message would be displayed, since the state information is built from the filters specified when collecting.

Connection/Channel

This string item allows the user to specify the connection or channel for which he wishes to see state information. . This item is specified by either entering the number of an SPP connection or by entering the name of a channel (e.g. routing, level0). If the channel does not exist, an error message will be displayed.

1.5.1.4.2 Filtering

The second half of the main subwindow contains the filtering items used to specify the packets the user wishes to collect.

source/destination address

The item that enables the user to specify the machine he wishes to watch is **source/destination address**. The user may specify an actual machine address in standard network address format, or a Clearinghouse name. The default address is the "all hosts" special

string. If the address cannot be translated, an error message will be displayed in the message subwindow and the tool will watch all hosts on the network.

The next four items are the protocol level filters. Only one may be selected at a time; should the user attempt to select a second item, the tool will turn off the previously selected one. When one of these filters is selected, an auxillary options window will be opened, showing the user the different filter and display options available for the protocol level chosen. For the details of these options windows, see §1.5.2.

Broadcasts This is a boolean item for specifying collection of broadcast packets only.

Level 0 This boolean item specifys that the user wishes to collect all level 0 packets. The portion of the packet beyond the encapsulation is treated as data.

Level 1 **Level 1** is a boolean item for specifying collection of level 1 **ns** or **pup** packets. One or both of these may be chose in the auxillary options window that is opened when this boolean item is selected. Any packet with a level 1 type that is not equal to that selected by the user will be ignored.

Level 2 This is a boolean item for specifying collection of certain level 2 **ns** packets. The packet type(s) to be collected are selected in the auxillary options window that is opened when this boolean item is chosen.

1.5.1.5 Log subwindow

The log subwindow contains the display of the formatted packets and other test information.

1.5.1.5.1 Sample log output

The following is a sample log of displayed packets, collected with the protocol level filter **Level 2** and a packet type filter of **any**. All of the display options booleans were turned on, enabling the user to view the entire packet.

```
NS Spy Tool 11.0a of 26-Jul-83 9:58:39 running on (Dandelion) 74.25200000031.
Log created on 28-Jul-83 8:49:41
267 pages available for buffering

Channel[pex, delta = 0B]
 Encapsulation[ethernet[#46021400022563B# ← #25200007561B#], type: ns]
 NS[checksum: 74423B, control: 3B, length: 64B, type: Pex,
156B#7777777777777777B#24B ← 3067B#25200012671B#15716B]
 PEX[id: 11325444200B, type: clearinghouseService(2B)]
  (20B)000003B 000003B 000000B 000000B 000000B 000002B 000002B 000000B
```

The first line of the displayed packet is the state information created by the tool. The timing used was **relative**, and since the was the first packet observed since the tool was last started, the delta (time since last packet was observed) is 0 milliseconds. The actual packet information starts on the second line, the display of the encapsulation, which

-- NSSpyTool.doc

Please refer to the document, "NS Spy tool", in XDE Unsupported Software Description.

-- NSSnarf.doc

NSSnarf copies NS files from the source volume onto the current volume. The default source volume is User and the default directory is the System Files Catalog. The user can specify the following local swithces:

c uses the current name as the unique abbreviation of a command name. The permissible commands are SourceDir and DestDir, which set the default values of the source directory and the destination directory, respectively. The name of the directory is the next name on the line.

s rename this file when copying it: the target name is the next name on the line.

u copy the file only if the source file is newer than the target file, or if the target file does not exist.

Example: to copy MyFile.mesa and MyOtherFile.mesa from logical volume Tajo, renaming MyOtherFile.mesa to Temp.mesa:

NSSnarf SourceDir/c <System> MyFile.mesa MyOtherFile.mesa/s Temp.mesa

IMPORTANT NOTES:

1. ServicesStubsConfig.bcd MUST BE LOADED BEFORE NSSnarf or you will end up in the debugger!

2. ServicesStubsConfig.bcd and NSSnarf.bcd must both be loaded with the "v" (ignore version mismatches) switch.

– – NSMailChecker.doc

NSMailChecker is a program which checks Network Services (i.e., Star) mailboxes. It can be run either to check for mail immediately or in the background to poll every five minutes for new mail.

NSMailChecker has four blanks to fill in: Name, Password, Domain, and Organization. These are pre – set to their currently set values in the environment but may be changed at any time; the changes will take effect the next time any command is invoked.

There are two commands which may be invoked, Start and CheckNow. Start starts a background polling process which checks for new mail approximately every 5 minutes. When Start is invoked, the command Start is replaced by Stop, which causes the background process to be stopped. CheckNow may be invoked either while the background process is running or when it's not, and causes an immediate check.

If the tool window is made tiny while the background process is running, the current status of the poller can be seen in the tiny window. The top line of the tiny window caption always says "NS Mail", and the bottom line says one of "Empty", "EXISTS", or "ERROR", depending on the state of the poll. If "EXISTS" appears, opening the window will reveal the number of messages currently in the mailbox. If "ERROR" appears, opening the window will reveal a more complete error message. If the window is made tiny when the poller is off, the bottom line says "Checker".

If NSMailChecker is STARTED in the tiny state (which may be accomplished either by saying "NSMailChecker/t" to the Exec or by including an "InitialState: Tiny" line in User.cm) it attempts to start the background poller immediately. Note that for this to work, you must have already logged in, so it probably won't work as part of your InitialCommand unless you use the WatchCredentials option described below.

NSMailChecker examines the [NS Mail Checker] section of User.cm. In addition to the usual fields (WindowBox, TinyPlace, and InitialState), you may also specify a NewMailTune to be played if the background poller finds new mail. In order to use the NewMailTune feature, you must have loaded and started Play.bcd.

If you specify "WatchCredentials: YES" (or "TRUE") in User.cm, then the form fields will be updated any time your profile data changes (e.g., via the Profile Tool or the exec's Login command). In particular, you can use this feature to make sure NSMailChecker doesn't hold onto a copy of your password after DMT or Poly smashes it.

If you like tiny window pictures better than textual tiny window captions, supply "YES" or "TRUE" as the value of the UseTinyPictures line. NSMailChecker comes with four built – in bitmaps (one for each of the three possible states of the poll, plus one for the "off" state) which attempt to look something like a Star inbasket. Just setting UseTinyPictures to TRUE gets you these default bitmaps.

If you prefer to supply your own tiny pictures, add them as the values of the EmptyBitmap, NotEmptyBitmap, ErrorBitmap, and/or OffBitmap lines. These should be in the same format that TinyWindowPictures uses, i.e., a list of 112 octal or decimal numbers separated by spaces. (Note that in order for this feature to work properly, NSMailChecker must NOT have an entry in your TinyWindow.icons file!)

NewMailTune, WatchCredentials, UseTinyPictures, and the bitmaps are read when the tool is activated, so to change them, just edit User.cm and deactivate/reactivate the tool.

Example User.cm section:

```
[NS Mail Checker]
WindowBox: [x: 563, y: 30, w: 463, h: 105]          – – this is the default
TinyPlace: [x: 964, y: 0]                           – – no default
NewMailTune: @200>ccggaaGGffeeddCC;                 – – default no tune
UseTinyPictures: YES                                – – default No
WatchCredentials: YES                               – – default No
NotEmptyBitmap: 0 0 0 0 0 177777B 177700B ...       – – default built in
```

NSMailChecker is intended to be a light – weight, low – overhead tool for simply checking for the existance of mail. Suggestions that it should perform information retrieval functions, such as display of message envelope information, will be cheerfully rejected.

8) A boolean named "Expand Pvt DLs" has been added to the NSMailSendTool and is intended to cause the members of any private Distribution Lists to be enumerated in the message header, so that the message may be answered more easily. Since private Distribution Lists are not currently supported, this boolean is only a placeholder.

9) The message "checking recipients ... " is no longer printed, as this is now part of the "sending ... " operation.

-- NSHardy.doc

FILES:

NSHardyPublic.df,
DistSvcSupport.df
NSHardy.doc
DistSvcSupport.bcd,
NSHardy.bcd,
NSHardyConfig.bcd,
NSMailSendTool.bcd

OVERVIEW:

NSHardy is a variant of Hardy which uses the Services 8.0 mail system instead of the Grapevine mail system. NSHardy.bcd may be run directly on top of the Mesa environment, as it has the necessary stubs and Services Common Software configuations already bound in. If other Services 8.0 Mail Tools are being used, NSHardyConfig.bcd is available to be run on top of DistSvcSupport.bcd (which exports the Services 8.0 Authentication, Clearinghouse, and Mail stubs) in order to save disk space.

The sharing of mail files with Grapevine Hardy is not supported; however, the tools may be run in parallel since the mail file extension used by NSHardy is ".nsMail" instead of ".mail". The differences between NSHardy and Hardy 11.0 are listed below.

DIFFERENCES:

1) NSHardy uses the User.cm section "[NSHardy]", which has the same format as the "[Hardy]" section.

2) The above .bcd files may have to be run with the "v" switch so that they are started despite version mismatches.

3) Only a single Mail Service is checked for a user's mail, since secondary mailboxes are not supported by the Services 8.0 Mail System.

4) Private Distribution Lists (files containing a list of mail recipients) are not currently implemented and are therefore not recognized as such in parsing the "To:" and "cc:" message header fields.

5) An "Append!" command has been added which inserts the current selection at the end of the mail file and creates a TOC entry for it, the result looking as if the new message were retrieved using "New Mail!". This can be used, for example, to extract a forwarded message so that it may be answered with the "Answer!" command, or to insert a comment into the mail file at an arbitrary location by setting the "Date:" field of the comment appropriately and then following the "Append!" with the "Sort!" command.

6) Messages which are not entirely readable by NSHardy, such as Star Documents, are left on the Mail Service so that the user may read them using, say, Star Mail. The parts of such a message that are readable, namely the header information and MailNote, are copied to the local mail file; a hint as to the type of the message is also added to the header of the local message. If the "Flush Remote" option is set to TRUE, then the message is marked on the server so that it will no longer show up as being new mail. In this case, the message is also marked with an "a" in NSHardy's TOC subwindow to indicate that there is an unreadable "attachment" to this message still on the server. Bugging "Delete!" has no immediate effect on the attachment of a message: the local part in the mail file is marked for deletion, but the attachment remains on the server intact. A subsequent "Expunge!" of that message will first attempt to delete the attachment from the user's mailbox; if this is successful, the message will then be expunged from the mail file (deactivating NSHardy or changing mail files has no effect on messages with attachments; they remain in the mail file, marked as deleted). Moving a message that has an attachment to another mail file will copy only the local part of the message -- the attachment remains with the original copy of the message (Note that at this point, the original copy of the message would be marked as deleted, and unless it were unmarked using "Undelete!", a subsequent expunge would destroy both the local copy and attachment of that message).

7) The NSMailSendTool provides for three ways of sending a message: as a MailNote, as a Text message, or as a MailNote with an attachment. A MailNote is the simplest way to exchange mail between the Star and Mesa environments. A Text message requires the Star user to convert the message before reading it, and its delivery incurs a slight amount more overhead than the MailNote, but it does allow for longer messages (a MailNote is limited to 8000 characters). Sending a message as a "MailNote with attachment" is intended as a way for the user to specify the type of a message; presently, there are no facilities provided by NSHardy either to the convert Mesa files to other formats or to forward attachments received by NSHardy.

- - NSChat.doc

NSChat allows you to chat to NS Remote System Administration, ITS and to NS Remote Executive. Type to the Executive:

NSChat <host>/<switch>

where the switches are as follows:

s	NS Remote System Administration
i	ITS
e	NS Remote Executive

A NSChat window will appear. If you have the LogIn boolean selected, you will be logged in automatically.

Options:
In the Options sheet you can select which type of service you want to talk to. Open the options sheet and select the desired host type (sa, exec, or its). Then hit Apply! on the option sheet.

Connecting:
Type the name of the host into the textsw of NSChat and hit the DOIT key (the one labeled MARGINS on the Dandelion keyboard). You should then be connected to the remote host.

NSChat has almost the exact user interface as that of the released Chat program. All the commands on the tools form subwindow behave analogous as those of Chat.

User.cm:
NSChat will read the following User.cm entries:

[NSChat]
LogIn: TRUE FALSE
HostType: sa any exec its

1. Retrieve MenuSymbiote.TIP and replace the one that's currently on your TIP> subdirectory. (i.e. Delete TIP>MenuSymbiote.TIP, then retrieve the new one with a Dest: TIP>MenuSymbiote.TIP).

2. Retrieve NewSymbiote.bcd.

3. Change the [FileWindow] Setup: entry as follows: replace the word "Menu" with "NewMenu". If you are using the standard system editor, replace the word "Edit" with "OldEdit". If you are using the Editor hack, you don't need to change "NewEdit".

You might want to change the Menu: entry to change your default FileWindow menu.

If you want to use the GlobalSymbiote tool, you should add a [GlobalSymbiote] section to your user.cm. See the section on the GlobalSymbiote tool for a sample user.cm entry.

3. Re – Boot. You must do this before the new MenuSymbiote.TIP table will take effect.

4. Run NewSymbiote.

I recommend running NewSymbiote from your User.cm InitialCommand line. If you use the Editor hack, you should run NewSymbiote first to make the Menu symbiote come out on top of the Editor symbiote.

NITTY GRITTY NOTES/RESTRICTIONS

– – Menu commands are executed on button up. You can abort a command by moving out of the symbiote before you button up.

– – NewSymbiote does not implement menu name qualification. This was not documented in the XDE User's Guide, but the system symbiote implementation allows you to qualify the name of a menu item ('FileWindow$Create') to cover the case where you have a command with the same name in more than one menu.

– – NewSymbiote matches symbiote items against menus by matching the item against all menu items and menu names and using the 'closest match'. The 'closest match' is defined as:

- all the characters of the symbiote item must be used up in the match (so we don't match 'Fine' with 'Find' if there is no exact match for 'Fine').
- if we match more than one menu item/name in this manner, take the one with the fewest unmatched characters. Thus: 'Fi' matches 'Find' rather than 'FileOps' because we have two unmatched characters with 'Find' verses five unmatched with 'FileOps'.

– – Note to those who use bounding characters to highlight symbiote items such as ' – Split – |Edit|': These examples still work, but ' – J.First – ' doesn't work. The rule: don't put bounding characters around something with embedded non – alphabetic characters. Read on if you care about why this is so.

The menu symbiote window is actually a TextSW. NewSymbiote relies on TextSW to resolve to the particular item you are pointing to. TextSW does so by simply doing a word select. This does not always get all of the item: if you have the item 'J.Last', pointing at 'Last' will only select that word, which wouldn't match according to the matching algorithm described above. So before NewSymbiote tries to match, it normalizes (or extends) the selection to include everything bounded by spaces (or the end or beginning of the symbiote line). In this example, the selection would extended left to include the 'J.'. However, this algorithm fails if you use other characters to visually set off menu items: ' – Split – ' or '|Edit|'. To support these kinds of items, if the search for a menu item fails with the normalized item, it will try again with the original selection. This will succeed for the ' – Split – ' example; if you pointed at 'Split', the search would first fail with ' – Split – ', but would suceed with just 'Split'. However, something like ' – J.First – ' would fail regardless; if you pointed at 'First', the search would fail with ' – J.First – ', and then would fail with 'First'. Thus the rule: don't put bounding characters around something with embedded non – alphabetic characters.

-- NewSymbiote.doc

NewSymbiote replaces the standard system menu symbiote facility. NewSymbiote adds the following capabilities to menu symbiotes:

* Menu item names can be abbreviated (i.e., 'Pos' instead of 'Position' or 'J.F' instead of 'J.First').

* Macintosh/Lisa – style popup menus are supported. If you put the name of a menu in a symbiote (such as "FileOps") and point down over that name, that menu pops up under the cursor. This saves you from having to search through the stack of menus.

* A "global symbiote" tool allows you to access commands on your root menus (such as the Inactive and ExecOps menus).

NewSymbiote works like the standard menu symbiote implementation as described in XDE User's Guide with the following changes:

Menu commands or menu names that have embedded spaces can be included in the symbiote by removing the space: "Show Type" becomes "ShowType". The standard symbiote implementation allows items with embedded spaces if they are surrounded with quotes. This is no longer supported.

NewSymbiote puts the NewSymbiote menu on the root menu. This menu should be used instead of the system Symbiote menu to attach or detach menu symbiotes to other tools. (Note that you can have a Menu: entry for any tool, such as CoPilot or even Life. You can only get these attached to a tool by using the "Attach New Menu" command in the NewSymbiote menu.)

Popup menus work slightly differently: point down (not chording) over the name of a menu bring up that menu. Select the command you want and point up; that command will be executed.

You can change the default FileWindow menu items by editing the [FileWindow] Menu: User.cm entry. Here's an example:

```
[FileWindow]
SetUp: Always NewMenu NewEdit
Menu: Cre  Load Edit Store Reset  Split J.F J.L J.S  FileW Text EditO  Dest TSave
```

GLOBAL SYMBIOTE TOOL

The GlobalSymbiote is a tool with no name stripe that looks like an standard symbiote, except that it isn't attached to a window. Items in this symbiote refer to the root menus (Inactive, ExecOps, etc.). The GlobalSymbiote behaves like an ordinary symbiote in all other respects, with the addition of the StuffToExec feature (see below).

NewSymbiote automatically starts up the GlobalSymbiote tool. To use it, you need to have a [GlobalSymbiote] user.cm entry. Here's an example entry:

```
[GlobalSymbiote]
WindowBox: [x: 512, y: 29, w: 512, h: 16]
Menu: Inactive ExecOps Profile Info Fetch.bcd Fetch.mesa Fetch.both
StuffToExec: TRUE
InitialState: Active
```

As with the [FileWindow] section, Menu: specifies the default menu items to put in the symbiote. In this example, 'Inactive' and 'ExecOps' popup those menus, 'Profile' brings up the Profile tool from the Inactive menu (if it's there), and the 'Fetch' items access the Fetch hack menu which also lives on the root window.

StuffToExec: If the entry StuffToExec: TRUE is present in the [GlobalSymbiote] section of User.cm, then if no menu or menu item match is found, then Exec.ProcessCommandLine is called with the symbiote item you selected. Typeout is to the default window, and to abort you must signal a global abort (hit STOP twice quickly). This feature is extremely useful for invoking arbitrary command files with the click of your mouse.

INSTALLATION

A NEW TIP>MenuSymbiote.TIP IS REQUIRED FOR THIS HACK.

-- MungeFileType.doc

MungeFileType is an Exec hack which changes the MFile.Type of a file to "binary". The only purpose of this hack is to work around a bug in the Command Central's "Run" command, which claims it is unable to find files whose MFile.Type is "text" (such as .TIP and .TIPC files).

Syntax: MungeFileType.~ <list of files>

-- MPClock.doc

The MPClock hack turns your maintenance panel into a digital clock. Repeating the command toggles the clock on and off.

-- MoveToDir.doc

MoveToDir moves or copies files from one directory to another. It is more convenient than issuing a sequence of "Rename" or "Copy" commands to acheive the same purpose.

To move files, the command syntax is:

```
MoveToDir.~ directoryName file1 file2 ... fileN
```

meaning: move the files "file1", "file2", etc. to directory "directoryName", equivalent to the command sequence:

```
Rename file1 directoryName>file1;
Rename file2 directoryName>file2;
...
Rename fileN directoryName>fileN;
```

Both the directory name and the file names are looked up on the current search path, so they may be either absolute (fully qualified) or relative pathnames.

To copy files, the command syntax is:

```
CopyToDir.~ directoryName file1 file2 ... fileN
```

meaning: copy the files "file1", "file2", etc. to directory "directoryName", equivalent to the command sequence:

```
Copy.~ directoryName>file1 ← file1;
Copy.~ directoryName>file2 ← file2;
...
Copy.~ directoryName>fileN ← fileN;
```

WARNINGS:

a) It doesn't work properly for files with absolute pathnames greater than 200 characters.

b) Because of bugs in the Exec's command lookup algorithm, if you want to use the Exec's "Copy.~" command after running MoveToDir, you must type the command name fully INCLUDING the ".~" at the end.

There is no way to send procedure signs. One possibility is to have "[" turn off character spacing, and "]" turn it back on. Thus, "[sk]" would send "s" and "k" run together. But this means I have to do a little parsing of the Send Random! selection.

There are some bugs in the cursor code that may cause old, dead insertion point cursors to be left around in the subwindows.

<number> | NITEMS | <itemVar> |
<itemExpr> + <itemExpr> | <itemExpr> - <itemExpr> |
(<itemExpr>)

<itemNum> :: = %<number>%

<itemVar> :: = %<identifier>%

<metaCommand> :: =
<interpretedConstruct><optionalBreakChars> |
<uninterpretedConstruct><optionalBreakChars> |
<metaCommand><optionalBreakChars><metaCommand>

<number>
- - digit - sequence representing a CARDINAL value; see also <itemNum>

<optionalBreakChars> :: =
<Empty> | <ignoredWhiteSpace> | <whiteSpace> | <breakChar> |
<optionalBreakChars><optionalBreakChars>

<optionalIgnoredWhiteSpace> :: = <Empty> | <ignoredWhiteSpace>

<optionalParam> :: = <Empty> | <optionalIgnoredWhiteSpace><metaCommand>

<paramList> :: = <optionalParam> | <paramList>,<optionalParam>

<relationOp> :: = > | > = | < | < = | # | =

<uninterpretedConstruct>
- - a <DigitSequence>; or an <identifier> NOT followed by "[" (i.e. vs. a <functionCall>)

<whiteSpace> :: = <SP> | <TAB> | <whiteSpace><whiteSpace> - - see also <ignoredWhiteSpace>

- -

- - MorseCode.doc

MorseCode is a Tajo tool that sends Morse code through the Dandelion speaker. MorseCode can send selected text, and can also send random code groups of characters chosen from the current selection. Character speed and spacing, weighting, and pitch are all variable.

MorseCode has four subwindows. Top to bottom, they are:
(labels are for documentation convenience only)

Message:	A message subwindow.
Command:	A form subwindow for commands and parameters.
Output:	an editable scratch subwindow used mostly for output
Input:	an editable scratch subwindow for user type – in to copy code.

Commands and parameters:

Send!

Send the current selection, which need not be in Output or Input subwindows. The selection may be of any length. MorseCode knows the code for the following characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890.,/? (and space)

Alphabetic characters may be in upper or lower case. The input focus is set to be at the end of the Input subwindow. You can type the received characters there, instead of using paper and pencil, if you wish. To interrupt sending, press the ABORT button.

Send Random!

Send random code groups, with characters chosen from the current selection (which must be < 200 characters long). Code groups are separated by word spaces, and are echoed in the Output subwindow after each group is sent. If you want to make some characters more likely than others, just repeat them in the selection: e.g. AAB will send A's twice as often as B's.

The input focus is set to the Input subwindow, as with Send!. MorseCode will keep sending until you press the ABORT button. Send Random! reseeds its random number generator each time, so you will not get the same code groups.

Insert All Chars!

Insert all the characters MorseCode knows how to send into the Output subwindow. This is convenient when using Send Random!.

Char WPM =

The speed at which individual characters are sent, in words per minute.

Spacing WPM =

The WPM rate used for spacing between characters (so you can send, say, 13 WPM characters at 5 WPM spacing).

Pitch =

Pitch of tone, in Hz.

Weighting =

Dah length to dit length ratio. 3 (the default) is standard.

Code Group Size =

Number of characters in a code group (1 to 10).

Send in background

Do the sending in a process running at background priority. This allows you to see your type – in immediately echoed in the Input subwindow. (If the process doing the sending is running at normal priority, echoing happens rather infrequently.) Turn this toggle off if other processes are interfering with the code sending.

Suggestions for recording code practice tapes:

I prop up my Dandelion keyboard on a few books, turn the speaker volume down fairly low, and slide a portable tape recorder underneath so its microphone is right next to the speaker. It's still necessary to be fairly quiet while recording.

Deficiencies:

All parameters are integers. This probably isn't a very important restriction except for the weighting.

RemoveSwitches[tokenList]
String – function – removes any switch settings (e.g. "/bnpu") from the listed tokens.

ReplaceSubstrings[string, old, new]
String – function – replaces each occurrence of old in string by new.

Same[string1, string2, caseMatters]
Boolean – function – yields TRUE iff string1 matches string2.
If "caseMatters" is present, capitalization must match.

Show[message]
String – function – displays message and yields empty string.

ShowCopy[string]
String – function – displays string and yields string as its result.
Debugging hint: Inserting ShowCopy calls may help explain why a meta – command performs as it does.

Sort[filenameList, option1, option2, option3, option4]
String – function – uniquely sorts the list of filenames, ignoring capitalization.
Option – order dictates major – to – minor sort order, and the valid options are:
host
dir – – (directory, including any sub – directories)
name
ext – – (extension, e.g. last occurrence of ".bcd" in filename)
The default sort order is: host > dir > name > ext > version – number.

SubstringFound[substring, string, caseMatters]
Boolean – function – yields TRUE iff substring occurs at least once in string.
If "caseMatters" is present, capitalization must match.

SuffixFound[suffix, string, caseMatters]
Boolean – function – yields TRUE iff string ends with suffix.
If "caseMatters" is present, capitalization must match.

Union[tokenListA, tokenListB]
String – function – yields sorted list of the tokens found in tokenListA and/or tokenListB.

– –
– –
BNF Definitions (Informal)
– –

<booleanExpr> ::=
<functionCall><paramList>] | – – the function must be Boolean – valued
<itemExpr><relationOp><itemExpr> |
NOT <booleanExpr> | <booleanExpr> AND <booleanExpr> | <booleanExpr> OR <booleanExpr> |
(<booleanExpr>)

<breakChar>
– – any character OTHER THAN a:
<Letter>, <Digit>, <SP>, <TAB>, <CR>, %, {, }, (,), [,], or <Comma>

<functionCall> ::= <identifier>[– – no white space between the <identifier> and [

<identifier> ::= <Letter> | <Letter><SequenceOfLettersAndOrDigits>
– – where the identifier is not a MetaCommands keyword, namely:
AFTER, AND, DO, EACH, ELSE, ENDLOOP, FOR, IF, NITEMS, NOT, NULL, OR, THEN, UNTIL, WHILE

<ignoredWhiteSpace> ::= <CR> | <CR><whiteSpace>

<interpretedConstruct> ::=
<functionCall><paramList>] | – – the function must be string – valued
[<optionalIgnoredWhiteSpace><HostName>] | – – i.e. so user can specify remote file names
IF <booleanExpr> THEN <metaCommand> ELSE NULL |
IF <booleanExpr> THEN <metaCommand> ELSE (<metaCommand>) | – – note the parentheses
<itemNum> |
FOR EACH <itemVar> DO <metaCommand> ENDLOOP |
FOR EACH <itemVar> AFTER <number> DO <metaCommand> ENDLOOP |
FOR <itemVar> ← <itemExpr>,<itemExpr> UNTIL <booleanExpr> DO <metaCommand> ENDLOOP |
FOR <itemVar> ← <itemExpr>,<itemExpr> WHILE <booleanExpr> DO <metaCommand> ENDLOOP |
<itemVar> |
{<FairlyArbitraryStringOfCharacters>} |
(<metaCommand>)

<itemExpr> ::=

- -
List of Built – In Functions
- -

AppendToTokens[suffix, tokenList]
String – function – appends suffix to each token in tokenList.

BlankOutExecControls[string]
String – function – replaces certain Executive – controlling characters by blanks, namely:
* # @ ↑ ' ? <TAB> <Semicolon> <CR> and also \\ or - -

CallExec[commandLine]
String – function – immediately feeds commandLine to the Executive, via Exec.ProcessCommandLine.
CallExec normally yields an empty string, i.e. when the Exec returns outcome = normal (or when commandLine is empty); otherwise, it yields the string, ABNORMAL – OUTCOME.

Empty[string]
Boolean – function – yields TRUE iff string has length = 0.

Error[message]
String – function – displays message and aborts execution.

Expand[commandLine]
String – function – performs wildcard expansion of tokens in commandLine.
Tokens beginning with "[" are expanded as remote – file names, if the host server permits.
Other tokens receive Executive – style expansion (for *, #, @, and ↑ characters).

FileExists[file]
Boolean – function – yields TRUE iff file exists.

FileGets[localFile, string, append]
String – function – writes string in the given local file and yields empty string.
If "append" is absent, the file is reset before string is written.

FileGetsCopy[localFile, string, append]
String – function – writes copy of string in the given local file and yields string as its result.
If "append" is absent, the file is reset before string is written.

FullDiff[tokenListA, tokenListB]
String – function – yields sorted list of tokens found in tokenListA or tokenListB but not both.

GetSwitches[string]
String – function – yields string's (rightmost) switch setting, e.g. /bnpu.
An empty string results if no "/" is present.
Otherwise, the result's first (and possibly its only) character is a "/".

Intersect[tokenListA, tokenListB]
String – function – yields sorted list of the tokens shared by tokenListA and tokenListB.

NewerFile[file1, file2]
Boolean – function – yields TRUE iff file1 has a newer create – date than file2.

NonEmpty[string]
Boolean – function – yields TRUE iff string has length > 0.

OlderFile[file1, file2]
Boolean – function – yields TRUE iff file1 has an older create – date than file2.

PartialDiff[tokenListA, tokenListB]
String – function – yields sorted list of tokens in tokenListA which are not also in tokenListB.

PrefixFound[prefix, string, caseMatters]
Boolean – function – yields TRUE iff string begins with prefix.
If "caseMatters" is present, capitalization must match.

PrependToTokens[prefix, tokenList]
String – function – prepends prefix to each token in tokenList.

PruneFilenames[filenameList, option1, option2, option3]
String – function – prunes each token in filenameList according to given options.
Option order is unimportant; the options are:
noHost – removes hosts (e.g. "[Igor]").
noDir – removes directories (e.g. "<Hacks>11.0>Tools>").
noName – removes names (e.g. "MetaCommands").
noExt – removes extensions (e.g. ".bcd").
Version – numbers (e.g. "!1") are automatically removed.

```
ELSE(Ftp Igor connect/c Hacks directory/c <Hacks>11.0>Source store/a>
  FOR EACH %fn% DO %fn%ENDLOOP; thinOutfiles.~)
```

vs. one whose closing parenthesis is placed differently (so files are always thinned out):

```
IF NITEMS < 1 THEN - - This command expects at least one arg;
ELSE(Ftp Igor connect/c Hacks directory/c <Hacks>11.0>Source store/a>
  FOR EACH %fn% DO %fn%ENDLOOP;) thinOutfiles.~
```

BUILT – IN FUNCTIONS

The hack offers quite a few built – in functions and is designed to readily accomodate more in the future. For that reason, it takes a hard – nosed attitude about constructs of the form:

```
<identifier>[
```

(Note the absence of white space between "<identifier>" and "[".) The hack's attitude is that <identifier> represents the name of a built – in function. If no function currently exists with that name, the hack assumes the user made a mistake.

The general form for calling a function is:

```
<identifier>[<metaComand>, <metaComand>, ..., <metaComand>]
```

This generality permits arbitrarily complex arguments involving loops, IF – tests, or more function calls, but ultimately the arguments resolve to string values. Different functions may, of course, place different restrictions on string – content. By the way, any leading white space (i.e. before an argument's <metaComand>) is ignored. Trailing white space, as usual, results in appending a single blank to the argument's string value (unless the white space starts on a carriage – return).

As in Mesa, arguments may be omitted or elided. A default string value is automatically supplied for each such argument, namely an empty (0 – length) string.

An alphabetic list of the current built – in functions is shown later, but a brief preview may be helpful. Many of the functions yield Boolean – valued results. Thus, there are functions which test:

- a string for a given prefix or suffix
- for empty or non – empty strings
- for a given substring anywhere within a string
- two strings for equivalent values or – if case matters – equal values
- whether a given file exists or not
- whether one file is older or newer than another

All other functions yield string – valued results, possibly empty ones. Among these are functions which:

- extract any switches occurring at the end of a string
- remove any switches occurring in a string's tokens
- prepend or append a given substring to all of a string's tokens
- replace occurrences of one substring by another, throughout a string
- perform wildcard expansion within a string
- blank out wildcards and other Executive – control characters in a string
- write a string in a given local file*
- prune out selected parts of filenames, e.g. for local – directory removal
- sort a list of filenames
- obtain the union or intersection of two filename lists
- obtain the full or partial difference between two filename lists
- display a string (without causing its execution)
- inform the Executive that an Error has occurred
- call the Executive to immediately execute a command string

* Hint: A function call of the form "Expand[@<filename>]" allows a meta – command to READ such a file.

One aspect of built – in functions can be a little tricky. Consider the following meta – command fragment:

```
Delete.~ *.errlog;
...
IF FileExists[foo.errlog] THEN...
```

If there is a file named foo.errlog, then the called function (FileExists) will detect that fact as soon as the meta – command is executed, i.e. BEFORE string "Delete.~ *.errlog;" is transmitted to the Executive (thus before foo.errlog is deleted).

In cases like that, where execution order is a problem, two remedies are available. First, an immediate call on the Executive can be arranged:

```
CallExec[Delete.~ *.errlog]
...
IF FileExists[foo.errlog] THEN...
```

Second, execution can be staged through the Executive's window by invoking an extra meta – command:

```
Delete.~ *.errlog;
...
extraMetaCommand.~
```

where that meta – command is defined to complete the intended task:

```
extraMetaCommand: "IF FileExists[foo.errlog] THEN..."
```

EVOKE

From time to time, the user may want to exercise a built – in function WITHOUT having to write a meta – command first. The hack registers the Evoke.~ command for that purpose. For example

```
>evoke Sort[$$$, ext]
```

would list filenames occurring in the current selection; where the list is sorted first by extension, then by host, directory, name, and version – number.

There are no restricted functions; all of MetaCommands' built – in functions are evokable. As with normal calls, arguments may be omitted or elided. Amusingly, perhaps, even the "]" can be omitted in an evoked call. Normal and evoked calls differ mainly, however, in the nature of their argument specifications. For a normal call, an argument can be an arbitrary <metaCommand>. For an evoked call, each argument is simply a <string> and cannot itself call a function, test for a given condition, or perform looping.

Hint: If you've forgotten the name of a function of interest, you can jog your memory by typing "Help Evoke" or "Help MetaCommands" to the Executive.

GRUBBY DETAILS

This section mentions some of the more obscure matters concerning MetaCommands, and it is followed by a list of all current built – in functions. The section concludes on the last page of this doc with an informal, but fairly definitive, BNF syntax description which is arranged alphabetically. (To track the syntax in top – down fashion, start at the "<metaCommand>" entry.) Some of the material below refers to BNF constructs on the last page.

The hack assumes no one will change any of its registered meta – command names (c.f. Executive command, ChangeCommandName). Users can always re – name their meta – commands by editing the command file.

When a <metaCommand> is evaluated, it yields a string containing a client – determined concatenation of uninterpreted constructs, break characters, and sub – strings resulting from interpreted constructs. Evaluation of an <uninterpretedConstruct> yields a string consisting of that construct; evaluation of a <breakChar> simply yields that character. Evaluating an <interpretedConstruct> yields a string result and also causes each <booleanExpr>, <paramList>, and <itemExpr> (along with sub – constructs) to be interpreted when encountered.

When evaluating an IF – THEN – ELSE construct, only one arm is interpreted; the other arm is ignored (skipped). As in Mesa, "x AND y" is treated equivalent to "IF x THEN y ELSE FALSE"; i.e. when x is FALSE, y is not evaluated. Similarly, "x OR y" is treated equivalent to "IF x THEN TRUE ELSE y"; i.e. when x is TRUE, y is not evaluated. Evaluation is basically left – to – right, and all binary operators have equal precedence (notably AND & OR).

A loop's <itemVar> can be assigned an arbitrary CARDINAL value in a FOR construct; however loop – exit occurs if that value falls outside the range [1..NITEMS]. Scope rule: An <itemVar> is considered defined only after being assigned its initial value in a FOR construct. The <itemVar> remains defined up to the corresponding ENDLOOP.

junkImpl.mesa$ ...deleted
junkImpl.mesa$$ ...deleted
junkImpl.errlog ...deleted

> SomeOtherCommand foo.drivel - - remainder of input line is now executed

In effect, then, the executed meta – command replaced its name by its expansion. (Note: An expansion can be any string acceptable to the Executive. Therefore, it is perfectly okay for a meta – command's expansion to include the name of another meta – command.)

The user will often want to defer designating an item of interest until the very last moment. Consider the following, rather limited, meta – command:

pressPrint: "OldPrint Auk/h %1%"

Suppose the command is executed via:

>pressPrint BingoImpl.mesa/L2

The result is Executive command line:

>OldPrint Auk/h BingoImpl.mesa/L2

In other words, the meta – command's peculiar "%1%" is replaced by the item "BingoImpl.mesa/L2" when expansion occurs. More generally, the hack interprets any construct of the form, %<number>%, as a reference to the <number>th item input to the meta – command. We call such constructs item – numbers.

Regarding input items: A. The items MAY include switches, as in "BingoImpl.mesa/L2" above, but that is not a requirement. B. The special item, $$$, is recognized and represents the current selection. For example, suppose the user selects a string such as "Test.mesa TestImplA.mesa TestImplB.mesa" and executes (while the string remains selected) the following Executive command:

>pressPrint $$$

The resultant expansion would be:

>OldPrint Auk/h Test.mesa TestImplA.mesa TestImplB.mesa

In short, the "$$$" item means "use the entire current selection in my place."

Let's consider another meta – command using item – numbers:

myPrettyPrint: "Formatter %2%;OldPrint %1%/H %2%/L2"

If executed as:

>myPrettyPrint Bud BingoImpl.mesa

the result is:

>Formatter BingoImpl.mesa
...
>OldPrint Bud/H BingoImpl.mesa/L2

The myPrettyPrint meta – command could be written in a variety of formats. For example, it might be written:

```
myPrettyPrint: "Formatter %2%;
        OldPrint %1%/H %2%/L2"
```

or:

```
myPrettyPrint: "
    Formatter %2%;OldPrint %1%/H %2%/L2"
```

In either case, the result would be the same as shown above, which points out something about white space (tabs, blanks, carriage – returns). White space beginning with a carriage – return is NOT passed through to the Executive. White space prior to, or not including, a carriage – return is converted to a single blank before transmittal to the Executive. (A partial reason: It wouldn't be nice to hand the Executive one or more tab characters every time the meta – command was executed.)

LOOPING

MetaCommands offers a general looping facility enabling access to an ARBITRARY number of items (such as the large number of items which may be represented by Executive command – string, "*.bcd"). There are four forms of loops, all having the same basic format:

FOR <loop control> DO <metaCommand> ENDLOOP

The simplest, most common form would be used to write a more – capable pressPrint command:

```
pressPrint: "OldPrint Auk/h FOR EACH %name% DO%name% ENDLOOP"
```

Suppose the user edits MetaCommands.cm in this fashion and then immediately executes:

```
>pressPrint *.config
```

The resultant output might read as:

```
Parsing [MetaCommands] section of MetaCommands.Cm...
25 meta – commands are now registered with the Executive

>OldPrint Auk/h That.config This.config Tother.config
```

The first two lines result because the hack detects that its command file has changed. Therefore, it de – registers all current meta – commands, parses the command file, and registers a fresh set of meta – commands. Incidentally, the same two lines are saved in a log file, MetaCommands.Log. (Hint: The log's main purpose is to preserve a copy of any error report in case the parse fails.)

After parsing the command file, the hack resumes execution. In the example above, we assume the Executive expands "*.config" into three items – "That.config This.config Tother.config" – each of which is assigned to %name% when loop – body "DO%name% ENDLOOP" is executed. (Constructs of the form "%<identifier>%" are called item – variables.)

Another form of loop would allow the user to write a more – capable myPrettyPrint command:

```
myPrettyPrint: "
        Formatter FOR EACH %i% AFTER 1 DO%i% ENDLOOP;
        OldPrint %1%/H FOR EACH %i% AFTER 1 DO%i%/L2 ENDLOOP"
```

The loop control, "EACH %i% AFTER 1" means %i% is to start with the 2nd input item, if available. More generally, "EACH <item – variable> AFTER <number>" means the item – variable starts with the <number + 1>th item, if available. Executing the myPrettyPrint command by means of

```
>myPrettyPrint Bud High.mesa Low.mesa InBetween.mesa
```

would yield the following (notice the replication of switch – setting "/L2"):

```
>Formatter High.mesa Low.mesa InBetween.mesa
...
>OldPrint Bud/H High.mesa/L2 Low.mesa/L2 InBetween.mesa/L2
```

The two remaining forms of loops provide increased flexibility at the cost of increased complexity. The interested reader may consult the "GRUBBY DETAILS" section for a BNF description, but here is a weird example illustrating both forms:

```
weird: "
    FOR %i% ← NITEMS, %i% – 1 UNTIL %i% < 3 DO – – major: %i%\N
      FOR %k% ← %i%, %k% – 2 WHILE %k% < (NITEMS – 2) OR %i% # 3 DO
        (– – minor: %k%\N)ENDLOOP
      ENDLOOP – – done"
```

Note that item – variables within loop – controls (FOR...DO) merely designate items – by numeric position – whereas item – variables within loop – bodies (DO...ENDLOOP) represent actual item content.

The weird example exposes several other points of interest: 1. The keyword, NITEMS, represents the number of items supplied by the Executive. 2. Loops within loops are permitted. 3. Parentheses may be used to achieve any desired grouping. 4. A limited arithmetic capability (+ or –) is available. 5. Logical operators (AND, OR, NOT) may be used. 6. Relational operators (<, < =, >, > =, =, #) may be used. 7. The hack has no comment facility, e.g. " – – done" will be passed through to the Executive.

IF – THEN – ELSE

Two forms of IF – THEN – ELSE are provided:

```
IF <Boolean expression> THEN <metaCommand> ELSE NULL
IF <Boolean expression> THEN <metaCommand> ELSE (<metaCommand>)
```

Note that ELSE is required, and in the second form notice the parentheses; they are mandatory. These requirements simplify analysis and clearly distinguish between a <metaCommand> such as:

```
IF NITEMS < 1 THEN – – This command expects at least one arg;
```

specialCheck – – For use by the mCompile & mBind meta – commands.

– – MetaCommands.doc

INTRODUCTION

The MetaCommands hack is intended to simplify the programmer's job. It does so by amplifying the power of the Mesa Environment's Executive tool. The magnification is particularly beneficial for repetitive, routine tasks, but it can also be exploited for exceptional, specialized tasks.

This hack is similar to ExecCommands. In large measure, ExecCommands provided the inspiration for MetaCommands. The MetaCommands hack departs from ExecCommands in three major respects: 1. Its syntax is reminiscent of Mesa. 2. It provides decision – making capability, IF – THEN – ELSE. 3. It offers a variety of built – in functions.

GETTING STARTED

The recommended approach to using the hack is: A. Read this doc, at least up to the "GRUBBY DETAILS" section. B. Obtain the hack itself and an initial command file:

MetaCommands.bcd ↑
MetaCommands.Cm (sample)

C. Examine MetaCommands.Cm, reading and heeding its comments. D. Start the hack and try out a few commands. E. Edit MetaCommands.Cm, customizing it to suit yourself.

FUNDAMENTALS

Running MetaCommands.bcd registers two Executive commands, MetaCommands.~ & Evoke.~ (discussed later). Starting the hack registers further commands, determined by the user. More specifically, these other commands, which we call meta – commands, are determined by the content of the [MetaCommands] section of file MetaCommands.cm. (If there is no such file, the hack looks in User.Cm.)

The [MetaCommands] section contains a list of entries, one per meta – command, each entry consisting of:

a name for the meta – command, terminated by a colon
an arbitrary amount of spacing (blanks or tabs), followed by an opening quote – mark (")
a declaration for the meta – command
a closing quote – mark, followed by a carriage – return

As usual, the entire section terminates if and when a double carriage – return occurs. The declaration, which we denote as <metaCommand>, may itself contain carriage – returns. It may also contain quote – mark pairs ("") representing a single quote – mark.

Regarding characters in a <metaCommand>: The ExecCommands' back – slash convention is obeyed. This, for example, allows use of \b or \B when a BACKSPACE must be conveyed to the Executive. Similarly, \n or \N can be used to convey a carriage – return. (Note that "\n\n" can be employed when the user wants to convey a double carriage – return.) The convention can also be used to "escape" characters (other than the quote – mark), i.e. characters which the hack would normally try to interpret for its own use. The following escapes are noteworthy:

\% \{ \} \[\] \(\) \<Comma> \<SP>

A "super – escape" is also available. I.e. a sequence of fairly arbitrary characters (quote – marks, double carriage – returns, and closing – braces being the exceptions) may be escaped by enclosing them in braces. For instance, given the super – escape

{ XrefByCaller[myFiles.list] XrefByCallee[myFiles.list]}

the hack would convey " XrefByCaller[myFiles.list] XrefByCallee[myFiles.list]" on to the Executive without trying to interpret that string.

In relation to the Executive, MetaCommands can be viewed as a highly sophisticated abbreviation expander. Take a trivial example. Suppose the following meta – command has been registered:

thinOutFiles: "Delete.~ *$ *.errlog"

When executed, the given meta – command prepends its result to the Executive's remaining command data. The Executive window's content might look as follows in a particular case (comments added):

>thinOutFiles; SomeOtherCommand foo.drivel – – user inputs this line

>Delete.~ *$ *.errlog – – this line results from executing meta – command "thinOutFiles"

```
-- MetaCommands.Cm - sample

-- Copyright (C) 1984 by Xerox Corporation. All rights reserved.


-- WARNING: These meta - commands assume you have obtained certain other hacks:
                SingOut.bcd BootVolumeName.bcd Play.bcd
    and have loaded the Play and SingOut hacks.  You might want to edit the
    InitialCommand entry in your User.Cm [System] section(s) along these lines:
            "Run.~ Play SingOut; MetaCommands;"

[MetaCommands]
new:        "
            FOR EACH %item% DO%item% ENDLOOP"
extract: "
            IF NITEMS # 2 THEN sos.~ The expected syntax is:
               \\extract.~ substringOfInterest $$$ {(i.e. current selection)}
            ELSE (CallExec[specialExtract.~ %1% ReplaceSubstrings[
               Sort[BlankOutExecControls[%2%]], $$$]])"
specialExtract: "
            Show[FOR EACH %t% AFTER 1 DO
               IF SubstringFound[%1%, %t%]THEN(%t% )ELSE NULL
               ENDLOOP]"
sos:        "
            {ChantOut.~[5, 2] '@160>>(aaa)% - (AaAaAa)% - (aaa)};error"
error:      "
            Error[]"
bootStar: "
            BootVolumeName System/IF NITEMS > 0 THEN%1%ELSE(Ldw\{) Y"
bootTajo: "
            BootVolumeName User/IF NITEMS > 0 THEN%1%ELSE NULL Y"
inStar: "
            IF NITEMS = 0 OR NOT SuffixFound[.boot, %1%] THEN
               sos.~ need 1st arg: [host]<directory>filename.boot
                  { (with optional 2nd arg giving boot switches)}
            ELSE (FileGets[reStar.my, %1%]
               reStar.~ IF NITEMS > 1 THEN %2% ELSE NULL)"
reStar:     "
            Install System __ Expand[@reStar.my];
            bootStar.~ IF NITEMS > 0 THEN %1% ELSE NULL"
shipToTajo: "
            IF NITEMS = 0 THEN sos.~ No files specified;
            ELSE(FileGets[toTajo.my, FOR EACH %f% DO%f% ENDLOOP, append]
               {SingOut '@105CCcAA(AaGgFF)(FGGgFFfDDd)(CCc<AA)AAA})"
transferToTajo: "
            specialTransfer.~ Sort[FOR EACH %f% DO%f% ENDLOOP
               IF FileExists[toTajo.my] THEN
                  Expand[@toTajo.my]CallExec[Delete.~ toTajo.my]
               ELSE NULL]"
specialTransfer: "
            OpenVolume.~ User/w;
            FOR EACH %f% DO Copy.~ <User>%f% __ %f%;ENDLOOP
            {SingOut <Bb>dDDDD(Cc<BbAa)BBb>CCcC'#C'#c'#DDD}; bootTajo.~"
mCompile: "
            IF Empty[CallExec[ShowCopy[
                  Delete.~ *.errlog *.mesa$$; specialCheck.~ *.mesa$$]]] THEN
               IF NITEMS > 0 THEN
                  IF NonEmpty[CallExec[
                        Compile / - b - ej - n - ps - uw - y FOR EACH %m% DO%m% ENDLOOP]] THEN
                     sos.~ Compilation error logs exist: Expand[*.errlog];
                  ELSE NULL
               ELSE NULL
               {SingOut DdddAAAAA}
            ELSE NULL"
mBind: "
            IF Empty[CallExec[ShowCopy[
                  Delete.~ *.errlog *.config$$; specialCheck.~ *.config$$]]] THEN
               IF NITEMS > 0 THEN
                  IF NonEmpty[CallExec[
                        Bind / - c - e - p - sw FOR EACH %m% DO%m% ENDLOOP]] THEN
                     sos.~ Binding error logs exist: Expand[*.errlog];
                  ELSE NULL
               ELSE NULL
               {SingOut DdddAAAAA}
            ELSE NULL"
specialCheck: "
```

```
        IF NITEMS > 0 THEN ShowCopy[sos.~ Please cease editing:
            \ReplaceSubstrings[FOR EACH %m% DO%m% ENDLOOP, $$]]
        ELSE NULL"


--Information about the meta-commands:

new -- For use after you have composed a fresh meta-command and want to try it.
        E.g. if the fresh command needs two args, you can type something like this:
            "new freshCommandName arg1 arg2"

extract -- For listing all filenames (or strings composed of filename characters)
        in the current selection which contain a specific string of interest.
        Steps: 1. Make the desired selection. 2. Place the cursor in the
        Executive window and tap the right-hand button. 3. Then to list all
        selected filenames containing the string "Impl" type:
            "extract impl $$$"

specialExtract -- For use solely by the meta-command, extract.

sos -- For alerting oneself to an error. E.g. try invoking the extract
        meta-command without any arguments.

error -- For informing the Executive that an error has been detected.

bootStar -- For booting the System volume using either specified switches or
        your favorite defaults -- namely the "Ldw\{" portion of the meta-command.
        (You may, of course, edit that portion as desired. NOTE: The "\" prevents
        the hack from trying to interpret the "{" itself.) E.g. to boot with the
        defaults, you might be able to simply type (i.e. if "boots" is
        sufficiently unambiguous to the Executive):
            "boots"
        in the Executive window; if you want non-default switches you might type:
            "boots dw{z"

bootTajo -- For booting the User volume using either specified switches or no
        switches. E.g. you might be able to simply type:
            "bootT 7{"

inStar -- For installing a new boot-file in the System volume...then booting it.
        The meta-command takes a mandatory first argument specifying the local or
        remote boot-file and an optional second argument specifying non-default
        boot switches (see the bootStar meta-command).

reStar -- For installing Star without having to re-type the local or remote
        boot-file you specified in the previously executed inStar meta-command.
        An optional argument may be given if you want to override your favorite
        default boot switches. Example:
            "restar dw{"

shipToTajo -- For designating one or more files to be copied over to the User
        volume the next time you execute transferToTajo. E.g. you might be able
        to simply type:
            "ship user.cm *.funnyData"

transferToTajo -- For copying files into the User volume and then booting that
        volume (with no switches). Any files previously recorded by one or more
        shipToTajo commands will be shipped to the User volume, and additional
        files may also be designated by the transferToTajo command. E.g. you
        might be able to simply type:
            "trans MyFunnyHack.bcd"

specialTransfer -- For use solely by the meta-command, transferToTajo.

mCompile -- For compiling one or more modules. The module names may carry
        switch settings which override the defaults. (You will want to edit the
        meta-command's " -b -ej -n -ps -uw -y" portion if you prefer other defaults.)
        The command aborts if you are editing any *.mesa modules. Example:
            "mcompile Foo/p FooPrivate/p FooImpl"

mBind -- For binding one or more configurations. The config names may carry
        switch settings which override the defaults. (You will want to edit the
        meta-command's " -c -e -p -sw" portion if you prefer other defaults.) The
        command aborts if you are editing any *.config modules. Example:
            "mbind TextBackingConfig TextDocConfig TextKernelConfig/p TextConfig"
```

-- MenuTidy.doc

MenuTidy is a program that reorganizes the inactive menu on a Tajo or CoPilot volume, allowing the user to create separate menus for tools whose inactive names match one or more patterns. For instance, all the Adobe tools might be put in a menu labeled "Adobe" whenever they are deactivated.

When MenuTidy is run, it registers the command "MenuTidy.~" with the Executive. Whenever the command is invoked, MenuTidy first looks for a list of menus and patterns in the user.cm (see below for format), then examines all tools (active, tiny, or inactive) looking for name matches. Any tool which matches a pattern will subsequently appear in the corresponding menu whenever it becomes inactive.

The user.cm entry for MenuTidy must be labeled [MenuTidy], and consists of one or more menu descriptions. Each menu description contains a menu name, followed by a colon, followed by one or more patterns separated by space(s), tab(s), and/or comma(s). A pattern may contain alphanumeric and/or special characters (except double quotes). The "wild – card" characters "*" and "#" may be used to match any number of characters or exactly one character, respectively. Also, a pattern may be surrounded by double quotes in order to include spaces or other delimiters.
Here is a sample user.cm entry:

```
[MenuTidy]
Adobe: Adobe*
Tools: *Tool "Command Central"
```

-- MesaUserTIP.doc

<Function>
* When you hit a SPACE bar twice within 200 msec, the character(s) just typed is expanded just like you hit a EXPANSION key. For JStar Level IV keybord users: "HIRAGANA bar twice within 200 msec" or "HIRAGANA bar and SPACE bar simalutanously = within 300msec".

<Possible application>
* Use this together with DictionaryTool; so that you can type MESA reserved words more easily.

<How to install>
* Run MesaUserTIP.bcd. It uses the above default setting if MESA.TIP is not there on <>TIP>. If you want to modify the timing 200msec.

-- MemoryMap.doc

MemoryMap is a tool which can be used to display information about the state of a client's virtual memory. The tool has three commands:

Pages! Enumerates pages in virtual memory, beginning at the number specified in the tool's FirstPage field, and ending at the number specified in the tool's LastPage field. For each of these pages, MemoryMap tells whether the page is in, out, or unmapped, and the module, if any, the page is associated with. If the ShowFlags boolean is turned on, the flags associated with each page will also be displayed under a column marked State. The State codes are:

D	dirty
W	write protected (readonly)
R	referenced

If the Reverse boolean is turned on, the pages will be displayed beginning with LastPage, and counting down to FirstPage.

Totals! Totals all the pages between FirstPage and LastPage and gives the total number of pages swapped in, out, and unmapped. It will also display a summary of the the dirty, write protected, and referenced flags for the pages totalled.

Modules! Displays the state of all the modules, frames, and packs in virtual memory. The modules are grouped by configs and within that config, by segment. Modules which are swapped in have an asterisk placed before their name. The number of pages the modules occupy, and their beginning and ending addresses in VM, are also shown. For each config displayed, totals are given for:

- packaged code pages, in and out
- packaged frames pages, in and out
- unpackaged code pages, in and out
- unpackaged frames pages, in and out

All of these commands can be aborted by pressing the stop key. MemoryMap will work correctly for both local and remote clients. For a remote client, simply start a CoPilot remote debugging session with that client, and then use MemoryMap.

-- MemSize.doc

MemSize registers the Executive command MemSize.~, which reports the real memory size of your machine in K bytes, based on the value of SpecialSpace.realMemorySize.

-- MakeDlionBootFloppyTool.doc

Please refer to the documentation in XDE User's Guide.

-- MazeWar.doc

How To Start

After running MazeWar.bcd, a tool window will appear with a single Form subwindow. To play a game, type is a name into the "Your Name:" field and hit "Start Game!". You will then either join another existing game or if there is no one else playing, you will start a game. The person to first start a game like this is known as the duke.

If more than one games is going on and you know the duke of a particular game, and if you know the duke's network address, type that address into the "Duke Name:" field before starting the game. The Duke Name: field uses AddressTranslation to parse its input. (see below for joining games on other networks)

Game Description

There are three subwindows in the MazeWar game, one with a rats' eye view, one with a birds' eye view, and a scoreboard. You can see your opponents only through the rats' eye view, and you can see yourself with the birds' eye view. The score is automatically updated in the bottom window.

Move the cursor into the MazeWar window to make your moves. The keys are mapped as follows:

Key	Action	
A	AboutFace	turn completely around (180 degrees)
S	LeftTurn	turn left
D	Forward	move forward
F	RightTurn	turn right
Space	BackWard	move backward
X	Shoot	shoot forward
Point	PeekLeft	peek left around a corner
Adjust	PeekRight	peek right around a corner
COMPLETE	ToggleScore	toggle between net addresss and player names

You change this by modifying MazeWar.TIP

Object of the Game

This is a basic seek and destroy game. As you see you opponents, it is best to start shooting at them. After taking a few hits, the opponent will disappear and reappear at some other random spot in the maze.

Joining other games

Up to eight people can play on one game, although many games can be going at the same time on the same network. To join a game on another network, type into the Duke Name: field:

Duke Name: <net number in octal>.*.

ie, to join a game on net 74B

Duke Name: 74.*.

consists of the immediate destination address, the immediate source address, and the packet type (for this level). The next line is the level 1 (**ns**) header information, which includes the internetwork destination and source addresses. The next line is the level 2 header, which the tool recognized as a packet exchange header. The last line is the data. The number in parenthesis is the length of the data.

```
Establishing[1B, delta = 0B]
 Encapsulation[ethernet[#25200016262B# ← #25200002663B#], type: ns]
 NS[checksum: 12706B, control: 0B, length: 52B, type: Spp, 74B#25200016262B#5367B
← 74B#25200002663B#2001B]
 SPP[sys, ackReq, sst[0B], connIDs[4000B ← 776B], seq#s[3B, 3B, 4B]]

Connection[1B, delta = 6B]
 Encapsulation[ethernet[#25200002663B# ← #25200016262B#], type: ns]
 NS[checksum: 64522B, control: 0B, length: 52B, type: Spp, 74B#25200002663B#2001B
← 74B#25200016262B#5367B]
 SPP[sys, sst[0B], connIDs[776B ← 4000B], seq#s[3B, 3B, 4B]]
```

The preceding two packets are from the establishment of an SPP connection (connection number 1). note that both packets are system packets, hence no data displayed.

```
Channel[routing, delta = 0B]
 Encapsulation[ethernet[#7777777777777777B# ← #25200000017B#], type: ns]
      NS[checksum: 62220B, control: 0B, length: 1052B, type: Routing,
0B#7777777777777777B#1B ← 74B#25200000017B#1B]
 Routing[type: {response}]
  [net: 214B, delay: 3B] [net: 131B, delay: 3B] [net: 74B, delay: 1B] [net: 101B, delay: 3B]
[net: 146B, delay: 6B] [net: 302B, delay: 3B] [net: 2026B, delay: 7B] [net: 305B, delay: 2B]
[net: 60B, delay: 2B] [net: 2574B, delay: 15B] ...
```

This packet is a gratuitous routing response being broadcast to all hosts on the net. Since the data in a routing packet has a standard format, the tool displays it in a more readable form than the raw dump format seen for other packet types.

```
Channel[????(173B), delta = 0B]
 Encapsulation[ethernet[#7777777777777777B# ← #25200000763B#], type: ns]
      NS[checksum: 177777B, control: 0B, length: 50B, type: ????(173B),
0B#7777777777777777B#10B ← 0B#25200000763B#111213B]
  (20B)001234B 056710B 000001B 000002B 000001B 000000B 000000B 000000B
```

This packet has an unrecognizable packet type. The tool only formats the known parts, dumping the rest of the packet in data format.

1.5.2 Auxillary options windows

The options windows normally do not exist, but are automatically created via the main tool window filter booleans. Only one options window is can be active at a time, and offers different options depending on the protocol level filter. In general, each window offers packet type filter options and options for displaying the packets. Some or all of the following items are shown in the options windows (See §1.5.1.4.2 for details of which windows these display options are available.)

Apply! **Apply** is a command item that that sets the options the user has just chosen in the options window. The window will not be closed.

Close! **Close** is a command item that first registers the options the user has chosen and then closes the options window. Note that changes to the options will have no effect unless they are set using the **Apply** or **Close** command.

Channel headers This boolean item used to request display of the state information about the packet's channel. This includes the channel name, and a timing number, either **relative** or **absolute**.

Connection headers The **Connection headers** boolean item is used to request display of the state information about the packet's connection. Connections are applicable to spp traffic only. The state information includes the connection number (as created by the tool), and a timing number, either **relative** or **absolute**.

Encapsulation The boolean item used to request display of the level 0 encapsulation portion of the packet is **Encapsulation**. The format will depend on the device type of the net. If the encapsulation is unrecognizable, it will be displayed in raw data format.

Level 1 headers This boolean item is used to request display of the level 1 (IDP) header portion of the packet. If the packet type is **ns** or **pup**, each field of the header will be displayed and labeled. If the packet type is unrecognizable, the header portion will simply be dumped as raw data.

NS headers The boolean item **NS headers** is used to request display of the level 1 **ns** header portion of the packet. It is used only with the protocol level filter **Level 2**.

Level 2 Headers The level 2 header portion of packet may be displayed by selecting this boolean. The packet must be a **ns** packet and the level 2 type must be a well known type such as **sequencedPacket**, **pex**, **routing**, **echo**, **error**, etc. Any packet that is not an **ns** packet or has a unrecognizable type will be displayed as raw data.

Spp system packets The boolean item **Spp system packets** used to request display of spp system packets. This item is only used if the tool has recognized the level 2 packet type as being **sequencedPacket**. If this boolean is not on, only spp packets carrying data will be displayed.

data Display of the packet's data may be specified by selecting this boolean item. The starting point for the data depends on the

level of packet being displayed. Note: Users displaying level 0 packet data from packet which s not ns or pup must be aware that the length of the data is obtained from the length of the raw ethernet packet. This length will always be a at least 46 bytes, the minimum ethernet packet data length.

1.5.2.1 Herald subwindows

The herald subwindow for each auxillary options window contains the window's name, **Broadcast options, Level 0 options, Level 1 options or Level 2 options**.

1.5.2.2 Broadcast options window

The options available to the user for display of broadcast packets are **ConnectionHeaders, Encapsulation** and **Data**. For simplicity's sake, broadcasts are treated as raw level 0 packets. Users who wish to see the level 1 and/or level 2 headers may collect broadcasts in buffered made and then replay them using the level 1 or level 2 filters.

1.5.2.3 Level 0 options window

As with broadcasts, the options available to collectors of level 0 packets are **ConnectionHeaders, Encapsulation** and **Data**.

1.5.2.4 Level 1 options window

When filtering level 1 items, the user may choose to look at **ns** packets, and/or **pup** packets. The display options available at this level are **ConnectionHeaders, Encapsulation, Level 1 Headers** and **Data**.

1.5.2.5 Level 2 options window

Level 2 filtering is applicable to Level 2 **ns** packets only. The options window offers the user a choice of all the well - known level 2 packet types to filter. If the user wishes to collect any level 2 packet, he may choose the special option **any**, and the tool will then collect both well known and unrecognizable level 2 **ns** packets. At this level, the display options are **ConnectionHeaders, Encapsulation, NS Headers, Level 2 Headers, Spp system packets** and **Data**.

1.5.3 Number conversion menu

The tool provides a facility to convert numbers to their corresponding values in various bases. To convert any number to hexidecimal, octal or decimal, the user simply selects the desired nmber to convert and chords the mouse in the main form subwindow to bring up a menu entitled **Convert**. Doing a menu select of teh desired destination base will convert the selected number to that base and dispaly the base and the number in the message subwindow. If the base of the selected number is not indicated by an appended B (octal) or H (hex), the base will be determined by the tool.

1.6 Operating procedures

On an ethernet, the tool is typically used as a third party peek. If spying on a phonenet, X.25 connection or any other type of point-to-point connection, it must reside on one end of that connection. When running on one end, the performance of the connection may be slightly impaired.

XEROX

Communication Services Test Programs

Remote Line Monitor Tool

Version 2.0
September 1984

Xerox Corporation
Office Systems Division
Systems Development Department
3333 Coyote Hill Road
Palo Alto, California 94304

1

Introduction

The Remote Line Monitor (RLM) Tool provides a mechanism for controlling the recording of data as it is transmitted and received by a Communication Server. The data can be written to a file and subsequently replayed in one of a number of display formats such as ASCII, EBCDIC, or HEX.

[In several places references are made to current limitations or to possible future developments and extensions to the features discussed. Such references appear in this font.]

1.1 Operational Considerations

The RLM Tool consists of 2 parts. One part runs on the Communication Server to record the data as it flows through the RS-232-C port. The second part contains the User Interface and display functions and runs in the Mesa Developement Environment on a work station. The tool taps into the data flow through the GateStream Interface on the server. This forces the restriction that only those communication services which use the GateStream interface can be monitored. Currently, this excludes only IRS related services. If the module (RemoteLineMonitorService.bcd) which taps the data flow is not already executing on the server, then it can be loaded using the **Remote Loading Tool** [With Services 8.0, this module is bound with the External Communication Service(ECS)].

The user is given control over the amount of buffer space available for logging data by specifying the number of 512 byte pages which will be allocated for buffering. If the buffer is completely filled during normal operation, the recording of the data may stop or continue by overwriting the oldest data in the buffer, depending on the choice of the user.

1.2 Hardware

No special hardware is required.

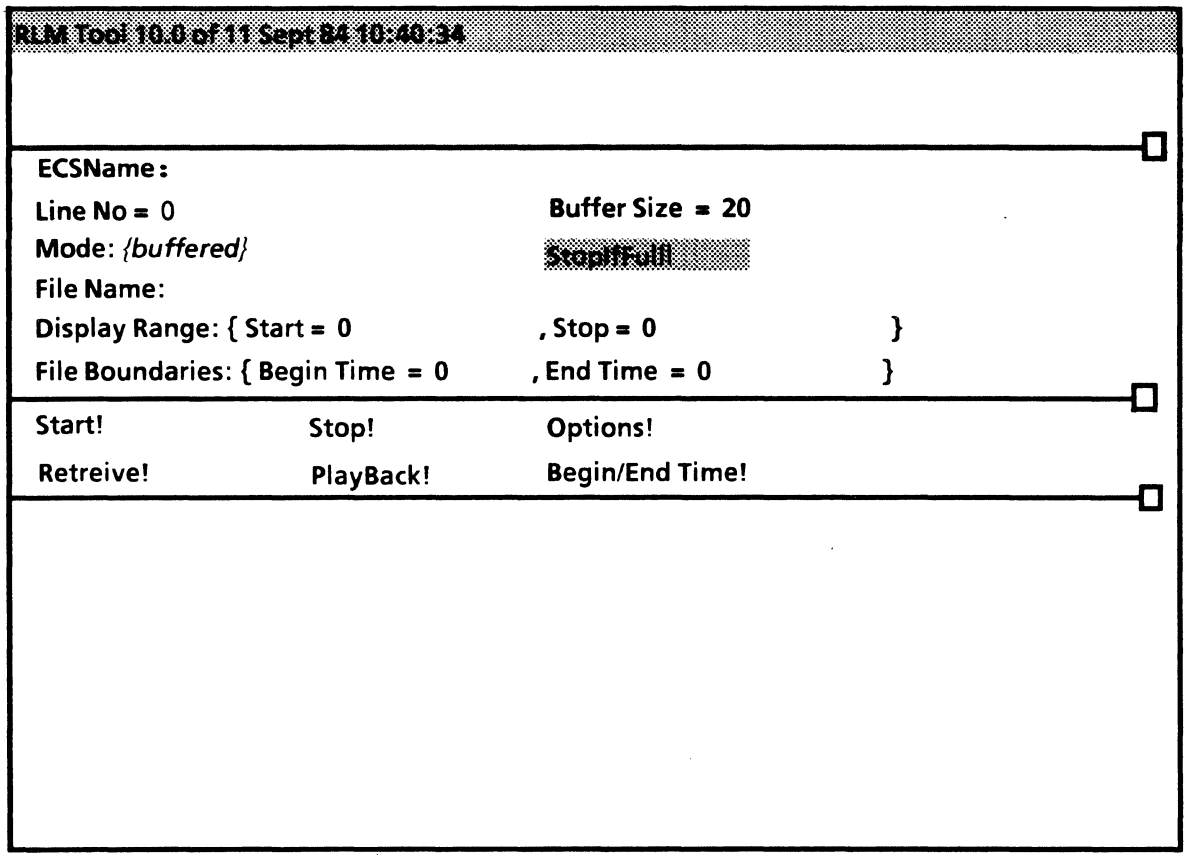
1.3 File Retrieval

RLMTool.bcd is the file required to run the RLMTool. Consult the Services 8.0 Release documentation to find out where Communication Services files are stored.

2

User Interface

The tool is run by typing the command "RLMTool" in the Executive window. After the tool has been loaded and started, the RLM Tool window will appear.



RLM Tool Window

The RLM Tool interacts through a Status Subwindow, a Parameter Subwindow, a Command Subwindow, a File (or TTY) Subwindow, and an Options window, which are described in the following paragraphs.

2.1 Parameter SubWindow

The Parameter Subwindow is used to identify the server and the RS-232-C port. Other pertinent information controls the collection process. The following describes each parameter in more detail with a list of menu items where applicable.

ECSName: The name or network address of the server which controls the RS-232-C port.

Line No = The line number which identifies the particular RS-232-C port. This number must be obtained from the Clearinghouse or ECS RS-232-C Port description. *The port can not be assigned to an IRS.*

Mode Menu Mode defines the collection process. The choices are **{buffered, realtime}**. The default is **buffered**. [realtime currently is not implemented]

Buffer Size = The number of 512 byte pages to allocate for buffering the logged data at the server. The default is 20.

StopIfFull A boolean which when TRUE designates the logging of data to stop when the buffer is full; when FALSE logging occurs using a circular buffer. The default is FALSE.

File Name The file name the data will be stored under when **Retrieved**. The file to be used during **Playback**.

Display Range The time interval chosen by the user to display part of a file. If no interval is chosen, the **Start** and **Stop** values default to the begin and end times of the file.

File Boundaries When **Begin/End** Time is selected , the begin and end times of the file are written to the parameters **Begin Time** and **End Time**, respectively.

2.2 Command SubWindow

Start Monitor! Once the desired parameters have been specified, this command is used to notify the ECS to begin monitoring the specified port.

Stop Monitor! This will stop the remote monitoring of the specified port.

Options! Displays the Options window.

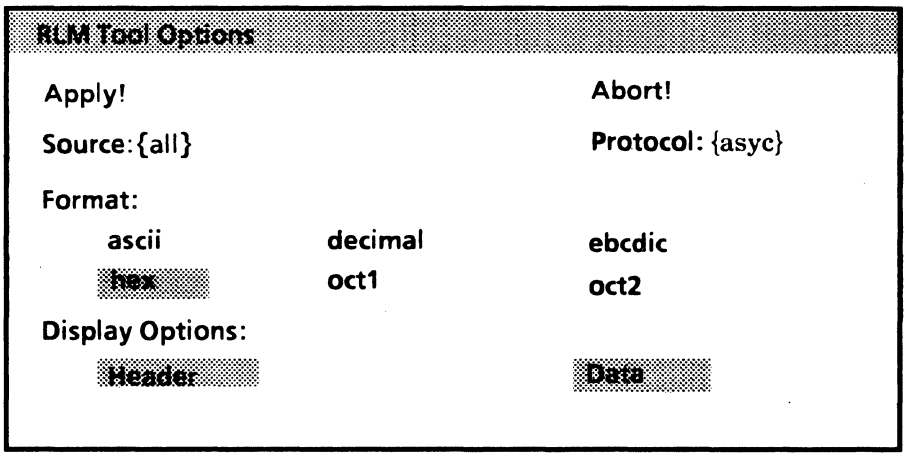
Retreive! The collected data is brought to the workstation and stored in the file specifed in the Parameter subwindow.

PlayBack! Displays the data from the specified file using the format(s) and display options specified in the Options Window. (**PlayBack** can be aborted using the ***Stop*** Key)

Begin/End Time! Displays the begin and end times of the file. This information is useful to display parts of a file without exceeding the boundaries of the file .

2.3 Options Window

The Options window contains the parameters which are used for displaying the collected data. The window appears when the **Options** command is selected.



Apply! Sets the parameters and destroys the window.

Abort! Resets the parameters to the values assigned when the window was opened and then destroys the window.

Source Menu The source of data to be collected or displayed. The menu is: **{none,send, receive, send&receive, status, send&status, receive&status, all}**. The default is **all**.

Protocol Menu The protocol of the data which is to be parsed. The choices are: **{asyc, bsc, pbsc, sna, xns, x25, none}**. The default is **asyc**. [Only **asyc** and **none** are implemented.]

Format Choices The format of the data when displayed. Choices are : **{ascii, decimal, ebcdic, hex, oct1, oct2}**. Up to six of these formats can be displayed during one **PlayBack** session. The default is **hex**.

Display Options When **Header** is FALSE the time offset and tranfer status of the data are not displayed. When **Data** is FALSE no data is displayed. The default is TRUE for both **Header** and **Data**.

2.4 Typescript SubWindow

The steps involved in running the RLM Tool are

1) Fill in the Parameter Subwindow. The parameters **ECSName, Line No, and File Name** must be specified, they cannot be defaulted. The other parameters can be defaulted.

2) Select **Start Monitor** to begin monitoring the specified port.

3) Select **Stop Monitor** to end monitoring the port.

4) Select **Rerieve** to store the data in the specified file.

5) Select **Options** to set the display parameters to be used during **PlayBack**.

6) Select **PlayBack** to display the data from the specified file. To display only part of the data, set the **Start** and **Stop** time parameters in the Parameter Subwindow. Select **Begin/End Time** toget the time boundaries of the file.

The following is output for **acsii** and **hex** formats.

```
Start Time: 22-Aug-84 10:29:46 PDT
Stop Time: 22-Aug-84 10:30:39 PDT
Total Sent: 8
Total Received: 6
Total Status: 0
Total: 14
[start + 46499 ms], success, received
  [3]  NUL@1
  [3H]  80C0H EC00H
[start + 48995 ms], success, sent
  [0]
  [0H]
[start + 49000 ms], success, sent
  [1]  r
  [1H]  7200H
[start + 49343 ms], success, sent
  [1]  u
  [1H]  7500H
[start + 49435 ms], success, received
  [1]  .
  [1H]  E000H
[start + 49745 ms], success, received
  [1]  DEL
  [1H]  FF00H
[start + 49873 ms], success, sent
  [1]  s
  [1H]  7300H
[start + 49989 ms], success, sent
  [1]  s
  [1H]  7300H
[start + 50227 ms], success, sent
```

```
  [1]  e
  [1H]  6500H
[start + 50468 ms], success, received
  [2]  O~
  [2H]  CFFEH
[start + 50629 ms], success, sent
  [1]  l
  [1H]  6C00H
[start + 50673 ms], success, received
  [1]  o
  [1H]  EF00H
[start + 50803 ms], success, sent
  [1]  l
  [1H]  6C00H
[start + 51186 ms], success, received
  [2]  DEL.
  [2H]  FF60H
```

- - RemoteBrowser.doc

The Remote Browser displays user - specified files on remote servers in its text subwindow, allowing you to inspect these files without having to retrieve them via the File Tool. The files are in fact retrieved to temporary local files, so the main advantage is that you needn't remember to delete them afterward. Also, you can do wildcard searches and look at the files one at a time, whereas you might not have enough disk space to retrieve them all at once.

Running RemoteBrowser.bcd registers an exec command, RemoteBrowser.~, which creates a new browser window. (If there's an inactive or tiny browser window already available, it is used instead.) The exec command accepts a switch, /t, that causes the new window to come up tinied. You can also choose to give the exec command a file name, in which case the browser window comes up with that file displayed. (If the file name includes *'s, you must precede them with apostrophes to get them past the executive.) You need to include the host and directory, as in

RemoteBrowse.~ [Host]<Directory>Doc>RemoteBrowser.doc

To use the tool, first fill in the Host, Directory, and Source string fields in the form subwindow. The Directory and Source fields may contain wild cards, which will generate a list of files to browse. As in the File Tool, you can also specify the entire qualified name (host, directory, and file name) in the Source field, thereby overriding the Host and Directory items.

Bugging Browse! will display the first file in the list. Bugging Next! will display the next file if there is one. (Bugging Browse! again re - displays the first file in the list.) In addition, you may select to browse All Versions (default FALSE) of a given file. A read - only string in the form subwindow displays the fully - qualified name of the file currently displayed. Finally, if you find a file you like, you may bug Copy!, which will retrieve a copy of the currently - displayed file to the directory currently on top of your search path.

The tool also contains commands Another! and Destroy!, which work as in the Chat Tool to create additional browser windows and to destroy the current one. Other commands let you Clear! the window and delete the temporary file, thereby freeing up the local disk space, and Close! the current connection, the same as in the File Tool (tinying the window also does this).

The tool windows look at the [RemoteBrowser] section of User.cm to override the default window box, tiny place, etc.

There is a Verify! command that works similarly to the Verify! command on the FileTool.

-- RemoteUsageTool.doc

Description:

This tool displays a histogram of the current activity of a remote server in a format similar to that of the Activity hack. Three bars are displayed: % CPU utilization, number of page faults, and number of Courier connections. The graph is updated whenever a utilization sample packet is received, nominally once per second. The scale of the display does not change when the window is adjusted.

Form Subwindow:

<Start!> Starts the RemoteUsageTool and sends a request to the remote server to begin sending utilization sample packets.

<Stop!> Stops the RemoteUsageTool and sends a request to the remote server to stop sending utilization sample packets.

<Host:> The name of the remote server may be in the form of a either a clearinghouse name or the actual network address. Any format acceptable to AddressTranslation will work.

Error information is posted to the Herald window. Extra copies of the tool may be created by running the tool again in the Executive.

-- Reclaim.doc

Reclaim is a hack that recovers "lost" disk space resulting from files whose page count is larger than required by the byte count. Such a file may be on your disk because it was shortened but not closed the last time you crashed or rebooted.

Reclaim registers a command "Reclaim.~" with the SimpleExec. This command can be used in two ways:

Reclaim.~

Enumerates all files on the system volume and shortens each one to the actual number of pages required. (Only permanent files in the Mesa development environment file system are affected.)

Reclaim.~ file1 file2 ... fileN

Reclaims unused pages from the specified files only.

Whenever it shrinks a file, it will display a message such as

<>Mail>Active.mail: shrunk from 85 to 83 pages

Since this operation requires write access to each file, you may also see a message like

<>Debug.log: conflictingAccess

as it encounters a file that is in use. Such a file is not touched.

- - RemoteActivityGraph.doc

Description:

This tool displays a running histogram of the CPU utilization, number of Courier connections, Pilot Operating System (OS) Page Faults, and Disk IO for a remote server as four individual graphs.

A special packet level protocol is used to remotely acquire the information with as little impact on the target server as possible.

The top graph displays CPU usage. The height of each vertical line is proportional to the number of CPU cycles (where 100% is the top of the graph) used during the previous second.

The second graph shows the number of Courier Connections. The maximum for the graph is set by the <Max Courier> item in the FormSW.

The third graph is Pilot Page Faults, and the bottom graph is Disk IO. The <Mag Pg Fault> and <Max Disk IO> items in the FormSW are used.

The entire display holds about 16 minutes of data, a data point is added at one second intervals, and the tick marks indicate 60 seconds of data.

When the display gets to the righthand edge, it wraps around and starts over. On the second pass and succeeding passes, there is a 10 - sample - wide swath of blank space to help you find where the new data is being displayed. The scale of the display does not change when the window is adjusted.

If more than one copy of the tool is desired, the tool should be run again in the Executive to create each new instance. The tool posts error information to the Herald window. The tool will be stopped if made tiny or inactive.

Form Subwindow:

<Start!> Starts the RemoteActivityGraph and sends a request to the remote server to begin sending CPU utilization sample packets.

<Stop!> Stops the RemoteActivityGraph and sends a request to the remote server to stop sending CPU utilization sample packets.

<Reset!> Clears the display.

<Host:> The name of the remote server may be in the form of a either a clearinghouse name or the actual network address. Any format acceptable to AddressTranslation will work.

<Smooth> This switch, when on, causes the average over the last 10 CPU utilization samples to be graphed rather than the value for the previous sample.

<Used:> Displays current activity: % CPU, % CPU smoothed over last 10 samples, % CPU smoothed over last 100 samples, Courier connections, page faults, and disk IO.

<Status:> This item will be in one of four states:

WAITING FOR CALIBRATION indicates that the server side process is self calibrating itself. This condition is transitive and will complete after the number of Courier connections at the server reaches 0. That may be awhile, depending on the server's requests.
INACTIVE indicates that the tool is not watching a server
TIME OUTS indicates that tool did not receive an information packet from the server as expected. This condition may be transitive, caused by internet or server congestion, or may be permanent if the server or a phoneline has failed.
ERROR CONDITION indicates that a protocol error was detected.
NORMAL indicates that the tool is receiving information from the server and is graphing it in the normal fashion.

<Time at left edge> This item shows the time at which the sample on the left edge of the screen was displayed. This time will be updated each time the graph wraps around.

-- Random.doc

Random.mesa is the interface to a uniform random number generator. It is pretty good at being random and it is very fast.

The two main procedures that it exports are

```
Random.Word: PROCEDURE RETURNS [ret: WORD];
```

Word returns a random 16 – bit pattern.

```
Random.InRange: PROCEDURE [low, high: CARDINAL] RETURNS [uniform: CARDINAL];
```

InRange returns a uniform number in [low..high] (Note that closed interval).

To initialize the package, a call can be made on

```
Random.Initialize: PROCEDURE [init: WORD ← 0];
```

This number is added to the seed. If 'init' is defaulted to 0, then the low order word returned by System.GetClockPulses is used to initialize the seed. So, if you want to be able to reproduce your "random" patterns during testing, call Initialize with something other than 0.

If Work or InRange is called before Initialize is called, Initialize is automatically invoked with 'init: 0'.

-- Random2.doc

Random2 is a package of pseudo – random number generators. Although not particularly fast (about 5 msec per), they are VERY random. There is no way to estimate the actual period of the base sequence, but it is at least 2**46, and probably a lot more. That means running flat out, it would repeat after 10000 years. Also, the density is good, since it uses all 32 bits of LONG CARDINALs.

A bunch of different distributions are implemented. As well as the usual uniform distributions (INTEGER, LONG INTEGER, and REAL), there are normal, exponential, chi square, geometric, Poisson, and some others.

– – alterations to ProcList

2) extend the 800 procedure limit.
3) allow the user to specify the number of tracable procs (or number of pages for the backing file) up to some limit (say 4000) on the call to StartTracing[].

- - PSpecAnalyzer.doc

PSpecAnalyzer.~ takes as input, a .map file and any number of .set files generated by ProcList.

Assuming that a .set file represents those routines invoked for some operation, PSpecAnalyzer.~ generates:

1) the total number of swap units needed to execute the operation.
2) the total number of real memory pages that are occupied by those swap units
3) the amount of fragmentation existing within those swap units
4) the total number of pages that the procedures in the .set file would occupy if they were packaged into a single code pack.

If the /v switch is mentioned:

5) a list of the code packs which must be in memory during this operation
6) a list of the frame packs which must be in memory during this operation

The invisioned use for PSpecAnalyzer.~ is:

1) in generating new packaging specs (i.e. this allow one to ascertain whether a particular edit to reduce fragmentation had more benefitrs than costs)
2) validating old ones (i.e. by generating a few choice .set files and determining whether the existing packaging spec is reasonable).

= = = = = = = = = = = = = = = = = = = =

Sample run:

>PSpecAnalyzer.~ TextKernelConfig.map BCBackspace.set

...Reading sets...

- - - - - - - - - -
Set name: [total swap units (code, frame), number of pages (code, frame), fragmentation (code, frame), ideal code size]

Data for...BCBackspace.set: [11 (10, 1), 57 (48, 9), 2 (2, 0), 30]

- - - - - - - - - -

>PSpecAnalyzer/v TextKernelConfig.map BCBackspace.set
...Reading sets...

- - - - - - - - - -
Set name: [total swap units (code, frame), number of pages (code, frame), fragmentation (code, frame), ideal code size, {code packs}, {frame packs}]

Data for...BCBackspace.set: [11 (10, 1), 57 (48, 9), 2 (2, 0), 30,
{BCSysPaintBCSysEditBCSysOpenBCSysCloseCmn, BCSysEditBCSysOpenBCSysCloseCmn, BCSysEditBCSysOpenCmn, BCSysEditBCSysCloseCmn, BCSysInsertCharBCSysDeleteBCSysBackspaceCmn, BCSysInsertChar, BCSysDeleteBCSysBackspaceCmn, BCSysBackspace, BCDocScrollBCDocEditBCDocOpenBCDocCloseCmn, BCSimpleEditBCComplexEditCmn}, {BlockAndChainConfigFrames}]

- - - - - - - - - -

- - Print2.doc

Print2 is just like print, except that it will post status messages in a Status window if one is loaded. See the documentation on status windows.

If you rename Print2.bcd to Print.bcd, NSHardy will use it.

-- ProcList.doc

ProcList is a tool that produces an alphabetically sorted list of procedures executed for specified configurations during a user – specified period. It consists of three files: ProcListNub.bcd, which runs in the client to be traced; ProcList.bcd and ProcListNub.symbols, which run in the debugger. The symbols for the configs you are tracing must also be on the debugger volume.

Tracing can be turned on and off in two ways. It can be controlled manually by interpreting ProcListImpl$StartTracing and ProcListImpl$StopTracing from the debugger, or it can be controlled automatically by using conditional breakpoints as described below.

Typically, you will run ProcListNub in the client you wish to trace, then interrupt to the debugger. Once there, run ProcList. (Have ProcListNub.symbols on the debugger volume). Interpret ProcListImpl$StartTracing, set configurations to be traced by using the Trace! command in the tool, and proceed to your client. (It is recommended that you run this in the order written). Do whatever it is that you want traced, then go back to the debugger. Now you can use the Print Procs! command together with the output filter to produce lists of procedures that were called in that trace. Interpret ProcListImpl$StopTracing when finished. Voila!

When using conditional breakpoints to control the tracing, note that invoking the Trace! command resets the Break Handler to off, so it's a good idea to set the Break Handler last (or keep your eye on it). Otherwise, turning tracing on and off with conditional breaks and the Break Handler works fine.

Form layout:

Break Handler: {off, on} Auto Zero Tables!
Watch: {priority} :
Trace! Configs:
Output Filter:
Nested EV Counts Print Procs! in: {window} : File name...

Break Handler: turns on special breakpoint handler. If this is on, conditional breakpoints are treated specially by ProcListNub. A condition of 0 turns tracing on, a condition of 1 toggles tracing, and a condition of 2 turns tracing off. Any other breakpoints will be passed on to Pilot's normal break handler.

Zero Tables! zeroes out ProcList's data. If the Auto boolean is on, this will be done automatically when you proceed from the debugger.

Watch: controls which processes are traced. The options are all processes, processes of a particular priority, or a specific process. In the last two cases, the priority or process handle is taken from the text field following this item.

Trace! Configs controls which configurations have tracing enabled. The argument to this command is a list of config names separated by white space. This command must be invoked with a non – empty argument for data collection to occur.

Output Filter determines which configurations will be included in the output. If it is left empty, all data collected will be included in the output.

Nested, if on, instructs ProcList to include nested procedures in its output. This parameter is normally off.

EV instructs ProcList to include in the output an item of the form "Module.ENTRY VECTOR" for each module that it encounters.

Counts tells ProcList to include the number of times each procedure was called in its output.

Print Procs! causes ProcList to list the procedures called since the last trace. Output can go to one of two places: ProcList's log window, or a file. The default file is "Trace.set". If no extension is specified in the file name, an extension of ".set" will be provided.

RESTRICTIONS:

Catch code execution is not detected by the tool.

The nub allocates a fixed space of 8 pages for its procedure maps. There are two words per procedure, plus another word of overhead per module. This allows space for approximately 800 procedures. If you try to enable tracing for more procedures than will fit, the tool will complain. An example is NSFilingConfig. Some workarounds suggested by Bruce Lee are:

1) do multiple traces, one for each portion of the config.

- - Play.doc

Input to Play is a string, either given in the tool window or taken from the current selection, interpreted as follows:

A letter from "A" through "G" specifies a note. If the letter is followed by "#" then the corresponding sharp - note is played (meaningful only for C, D, F, G, and A). All notes are eighth - notes, but upper - case letters cause tones that are "held" the full time while lower - case notes last for 3/64 seconds less and followed by a 3/64 rest.

C is the bottom of the octave; B is above C. When ">" is encountered, all subsequent notes are an octave higher; a "<" lowers all subsequent notes by an octave. Going up more than 3 octaves is not permitted (additional ">"s are ignored), and notes near the top of the highest octave may not be struck accurately.

A "/" in the string halves the note durations, down to a minimum of 64th - notes; a "*" doubles the durations up to a maximum of full - notes. A lower - case 1/16th - note would actually be a 64th - note followed by a 3/64 rest, which may or may not be what you want; a lower - case 32nd - note vanishes completely! To halve and double the amount of implicit rest "stolen" from lower - case notes, use "←" and "↑", respectively.

Use "%" to get an explicit rest (as distinct from the implicit ones after lower - case notes). The rest is the same length as a note, i.e., initially an eighth - rest, and changed via "/" and "*".

A "+" causes the preceding note to be held for an extra 50%. Thus a quarter - note followed by a "+" becomes a 3/8 - note, etc. A "-" causes the preceding note's duration to be halved. This is effectively a shorthand for bracketing the note with "/" and "*".

A left parenthesis causes subsequent notes and explicit rests to be at 2/3 normal duration, until a right parenthesis is reached. Three notes enclosed in parentheses yield a "triplet".

If you think you know exactly what tempo you want, use "@<n>" to give the note duration and/or ",<d>" for the lower - case implicit rest, where <n> and <d> are strings of digits representing milliseconds. The values will be constrained to the usual limits (e.g., <n> will be forced between 8 and 1024 ms). Subsequent "*", "←", etc., have their usual effects.

Two or more notes and/or rests enclosed in braces, as "{C%%%G#}", yield a "slide" from the first note to the last. The duration of the slide equals the total duration of the notes and rests; observe that upper - and lower - case notes have the same effect. The slide consists of 64th - notes at equally - spaced (logarithmic) frequencies. Warning: This can eat up a lot of array space!

A period (".") in the string causes the buffer to be shipped out, as for a semi - colon, and resets the octave, note duration, and implicit lower - case rest to their initial values. This is for when you're playing an entire file that contains several separate pieces.

= = = = = = = =

The other tool commands in Play should be fairly straightforward. The format for a music file is one or more pieces, each ending with a period and containing somewhere (usually at the front) a [bracketed] name for the piece at the beginning of a line. Bracketed items elsewhere in the line are ignored (i.e., are treated as comments). Unnamed pieces are assigned the name of the file.

-- PowerMouse.doc

PowerMouse is a tool which modifies the behavior of the mouse icon, or cursor, with respect to the motion of the mouse itself. There are two parameters which describe this modification: 'Threshold' and 'Acceleration'. Threshold is a speed (in pixels per screen refresh), which is the minimum speed of mouse movement for which cursor movement will be modified. Acceleration is a percentage, describing the maximum amount of movement modification. Acceleration = 100 is normal mouse action. Acceleration = 200 is a maximum of doubling cursor response.

The tool communicates to the user through a form window. Amplification and Threshold are readonly number items, which may be modified by using the associated " + " and " - " command items. Below is an approximation of the appearance of the tool (for a fixed point screen font).

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|Power Mouse 10.0 of 16-Jun-83 13:34:07 |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|| Amplification = 200     +!     -! |
|| Threshold = 20          +!     -! |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The tool is activated by loading and running PowerMouse.bcd. When the tool is not inactive, it is modifying the behavior of the mouse. It can be turned off by deactivating the tool (or unloading "PowerMouse").

PowerMouse.bcd registers a single command with the executive, "PowerMouse.~". It is described below, in its help message.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
>help PowerMouse

PowerMouse.~ activates the PowerMouse tool.
When tiny or active, this tool modifies the behavior of the mouse.  There are two parameters:
Threshold and Acceleration
   Threshold is the minimum speed (pixels per screen refresh) for
      modified behavior.  Try 15.
   Acceleration is the amount of behavior modification.  100 = normal.  Try 200.


>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

PowerMouse.bcd reads the user.cm for the initial values of Amplification and Threshold. Below is the [PowerMouse] section of my own user.cm.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[PowerMouse]
WindowBox: [x: 557, y:  0, w: 237, h:  49]
TinyPlace: [x: 964, y:  0]
InitialState: Tiny
Amplification: 200
Threshold: 20
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

p.s. - Those of you that have used previous versions of this program are in for a pleasant surprise: the response is very smooth and more intuitive than before.

I don't implement local file variables (that is, a file variable that is local to a procedure invocation); all files must be global.

8. Subranges.
In most Pascals, it is permissible to send an integer declared to be in the range [2..3] to a VAR parameter of type INTEGER. This is illegal in Mesa for two reasons. The first is that it permits unconscious breaching of the Mesa type system, and bounds checking is nearly impossible. Worse yet, Mesa represents its subranges in biased form, so you would get the wrong answer. This Pascal feature is used fairly often, so I compile a Mesa LOOPHOLE if the biases are compatible. This disables tight type checking, and you may want to fix the Pascal program so that it only sends variables of type X to VAR parameters of type X.

9. Program Headers.
I ignore the files specification part of the program header. In particular, I don't do an automatic RESET or REWRITE on any files mentioned in the header.

10. Character set.
The character set is full Ascii. That means that string constants in your program will appear in the case in which you typed them (except for the CAPITALIZE option; see above), and that when reading an input file you will see every non – control character exactly as it appears in full Ascii. Control characters are turned into blanks except for CR, which turns into a blank coupled with the EOLN condition (I/O is modified by the control field of the record representing a text file; see above).

11. Text I/O.
PasMesa only supports text I/O of integers, reals, characters, strings, and booleans. It doesn't support other enumerated types, or sets, or arrays, or records. There are fairly simple ways of getting around this in either Pascal or Mesa.

12. Depth of Procedure Nesting.
There were three bits free in a word in the symbol table, so Mesa limits this depth to 8, minus a few. I have encountered programs that exceed this limit by two, and I suppose worse is possible. I know this is a silliness in Mesa, but I suggest that you fix the Pascal source code.

13. Global Variables.
Pascal programs frequently include many global variables, and the current Mesa compilers place fairly tight limits on the number of variable interface items that are allowed in a DEFINITIONS – type .bcd. PasMesa's modularization mechanisms discussed above are enough to handle this problem, since you may declare more than one DEFINITIONS module in the .mod file, and parcel out the global variable among them by name. But this is sometimes tedious. If you are willing to be courageous, there is an easier way: importing a PROGRAM module. (This is nefarious process associated with the name POINTER TO FRAME.) It is legal in Mesa, although unusual, to have one PROGRAM module import another PROGRAM module instead of a DEFINITIONS modules. At the price of introducing a compilation dependency between the two PROGRAM modules involved, this allows the latter module to get at all of the variables, types, and procedures of the former one. Furthermore, PROGRAM – type .bcd's are allowed to have more variable items than DEFINITIONS – type .bcd's. To avail yourself of this technique, simply declare a PROGRAM module in the .mod file to which all of the variables are sent (with $other←vars$, presumably), and then list this PROGRAM module as an import when you declare the rest of the PROGRAM modules.

If you decide to go this route, be warned that some Mesa debuggers get confused about this kind of importation; if you ask about an unqualified global variable name, you may get an incorrect answer. In conversations with such debuggers, you will have to qualify all global variable names with the name of the module that defines them.

14. External procedures.
For performance reasons, or simply for convenience, it is sometimes helpful to implement some of the basic operations of a Pascal system directly in Mesa, rather than in Pascal. For example, you might want to invoke graphics primitives from a Pascal program, or you might want to do input and output more efficiently than the standard Pascal runtime support allows. To do so, PasMesa allows you to declare procedures in the outermost block of the Pascal source simply as ``external''. PasMesa will generate a correct Mesa header for each external procedure, and put that header into the DEFINITIONS module that you specify. It will also put a copy of the header without the required body into the PROGRAM module that you specify. You are expected to take this PROGRAM module and edit it as necessary to supply a body for each external procedure.

It would be quite tedious if PasMesa turned around and smashed your carefully hand – written procedure bodies with another copy of the header hints the next time that it was run. To avoid this, I suggest that you specify a funny extension, such as ``.hint$'', for the PROGRAM modules onto which PasMesa is to write these header hints.

-- PhoneBook.doc

The program PhoneBook maintains a personal phone list (or any other use you have for a <name, value> pair list). PhoneBook registers four commands with the executive:

AddPhoneNumber.~ name number
RemovePhoneNumber.~ name
FindPhoneNumber.~ name
ListPhoneBook.~

AddPhoneNumber.~ adds the pair <name, number> to the phone directory, if it already doesn't exist. Name may be any 100 character string,and number any 62 character string. If either name or number includes spaces, quote the string.

RemovePhoneNumber.~ removes the entry name from the phone directory, if there is one.

FindPhoneNumber.~ finds the entries matching "name" in the phone directory and prints their numbers. "name" may contain the wild cards * and #, but they must be quoted to appease the executive. The wildcard \ is equivalent to *, but it need mot be quoted.

ListPhoneBook.~ creates a formatted list of the phone book in the file PhoneBook.txt.

These four commands will also accept as an optional first parameter of the form book/b. This will cause the command to use the phone book with the name "book" or "book.btree" if book contains no periods.

Help procedures are also created by this hack. The phone directory is stored in the file PhoneBook.btree. If AddPhoneNumber.~ can't find PhoneBook.btree, it will create one for you.

compilation, and it goes like this if f is an input text file, and you want it to be called "foo":

```
PascalAttachFile[file: @f.baseFile, name: "foo", dir: input, itemSize: byte];
```

The other value of the dir parameter is output. The itemSize parameter is byte for file of character and word for anything else. The name parameter defaults to NIL, and itemSize defaults to byte. If you omit the name parameter, it will prompt you. There is no built – in editing: if you make a mistake, you start over.

The text file T is a record with a 6 – bit control field, T.control, whose definition can be found in PascalNoviceFiles. Its purpose is to control how T will handle upper/lower case and control characters. The default is that a file will read upper and lower case but will read all control characters except CR as blank, and that the file will write upper and lower case, and all control characters exactly as specified. The default is applied only at module START time. After that you can change these control fields in Mesa to your heart's delight.

Refs versus Long Pointers

Suppose that PasMesa is compiling some Pascal program into Mesa code intended to run within Cedar. At first blush, you might suspect that the right thing to do would be to have PasMesa use REF's and Cedar safe storage as the implementation of Pascal pointer data types. It even seems at first blush that there might be a reasonable chance of making PasMesa's output ``safe'' in the Cedar sense. Unfortunately, these hopes are torpedoed by the presence of VAR parameters in Pascal. After all, if VAR parameters were easy to implement within the current Cedar, they would have been implemented. There is a plan to implement them someday, even though they are hard, and perhaps PasMesa's output can be made SAFE at that time.

Why are VAR parameters hard? First of all, note that one can't implement VAR parameters simply by using REF's, since the actual supplied for a VAR parameter may be a component of a record or array, while REF's can only point at top level objects. This problem could be solved by passing both a REF and an offset as the way of referring to a supplied actual by VAR. But there is a more subtle problem. Under normal conditions, REF's only point at objects in the heap, that is, in counted zones. But the value being passed by VAR might be in a counted zone, or in an uncounted zone, or in a frame. Even if we were willing to stretch the rules somewhat, and use LOOPHOLE's to acquire REF's to uncounted storage, we wouldn't know in general when to increment and decrement reference counts and when to leave them alone.

Hence, even in Cedar, PasMesa uses LONG POINTER's rather than REF's. This means that Pascal programs are unsafe, in the technical Cedar sense.

There is one place, however, where Pascal files are allowed to play with REF's, and that is in the various implementations of Files. The Pascal runtime, given the Pascal record that represents a Pascal file, has to be able to get its hands on an IO.STREAM, the Cedar object that implements the file. And an IO.STREAM is REF – containing! we don't have any choice about that. The safest way to do this association would be to let the Pascal program (which is unsafe code, after all) deal only with integer indices, called JFN's in TENEX. The runtime would have a data structure to map these JFN's quickly into the appropriate IO.STREAM. But I was too lazy to do write the necessary code, so I played a little faster and looser. The Pascal record that represents a file actually contains a LONG POINTER to a record in counted storage, one field of which is a REF to the IO.STREAM. Because this record is referred to by LONG POINTER's, it's reference count is not reliable. The Pascal runtime deals with this by linking all of these records onto a single linked list with REF's, so that they will have reference count 1 in perpetuity; they are never freed. This isn't a big problem, since the records are not large.

Restrictions and Limitations

Mesa has several restrictions and limitations that are not present in most Pascal implementations. For some of these I have provided a detour; for others you will need either to edit the Pascal program or to edit the resulting Mesa.

1. Identifiers.
   For comparison purposes, Pascal identifiers are capitalized and all characters are compared. This can (and has) cause some identifiers that were treated as the same by some other Pascal compilers to be treated as different by this one (e.g., if their first difference is in the 15th character). This happens so often that I put in a special feature that tries to figure out what was probably meant. It looks for any exact match in the context stack, or failing that, the longest match among all identifiers in the context stack with the correct first eight letters. There may still be mismatches in the translated Mesa, but at least PasMesa itself doesn't blow up.

In the Mesa translation, the first letter is capitalized and all others are in lower – case. This avoids stepping on Mesa's reserved words. The character 137B (underscore, or ← in Xerox' version of Ascii) is not translated, but rather causes the next translated letter to be capitalized.

You may find yourself frustrated after a while by the necessity of dealing with two different syntaxes for identifiers. In Pascal, case is not significant and "←" is used to mark word boundaries; while in Mesa, case is significant, and is itself used to mark work boundaries. This can be quite annoying when, for example, you are debugging the running Mesa from the Pascal source, and you want to find the value of some variable. To ameliorate this situation, a special hack program called CapsArrows is available through PasMesa.df. Running CapsArrows adds a button to the system window at the top of the screen. When this button is left – clicked, the currently selected text is translated from caps (Mesa) format to arrows (Pascal) format; right – clicking does the reverse. CapsArrows is a useful hack, offered for whatever it is worth, but don't expect perfection. For example, it will blow up if the selected text spans node boundaries.

2. Record length.
   Mesa puts a limit of 4096 words on the length of a record, and construes local and global frames to be records. Most Pascal systems have no such limitation.

For an array variable declared in the global frame or a local frame, a situation that seems to arise all the time, the solution is to declare that array to be allocated from the heap instead..

A general solution for records would be really complicated, because it involves introducing new pointers within the record, and one is confronted with allocating, freeing, and copying indirectly – referenced sub – records. This situation seems to arise mostly when the programmer wants to read/write a heterogeneous binary file from/to an earlier/later Pascal program execution. He does this by declaring a huge record type, and a file whose records are of that type, and reading/writing one record. At this point your only hope is that he localized this activity in one place, so you can change it easily.

3. Integer size.
   In PasMesa, the type PascalInteger is defined to be INT by the PascalBasic portion of the runtime package, where INT is a 32 bit integer. Be warned that INT's are not yet fully supported by the Mesa compiler. For example, you can't have a subrange type of INT whose representation would exceed 16 bits.

4. Precision of integer arithmetic.
   Be also warned that the current Mesa compiler behaves somewhat unpleasantly as regards arithmetic. If you add two 16 – bit quantities, the Mesa compiler will compile a 16 – bit add, blithely assuming that the sum will fit into 16 – bits as well. Furthermore, there is no ove[illegible] checking at runtime. This caused difficulties during the port of TeX, so I implemented the following dodge. PasMesa now keeps track of the compile – time upper and lower bound of all scalar quantities. When PasMesa is compiling an arithmetic expression, it does interval arithmetic on these bounds. If the result can be guaranteed by this interval arithmetic to fit into 16 bits, PasMesa compiles code that will compile a 16 – bit operation; else, PasMesa coerces one of the arguments to INT to force the arithmetic to be done long.

5. Sets.
   I have implemented Pascal sets as Mesa packed arrays of BOOLEAN. There are three basic sizes: 16 elements, 64 elements, and 256 elements. PasMesa chooses the next larger size. Most of the 16 – bit operations are inline single – word BOOLEAN operations, and should be quite fast.

6. Go To's.
   The GOTO statement in Pascal is of the traditional variety; but the GOTO statement in Mesa is quite restricted, since it always forces an exit from some enclosing block. Let us first consider local GOTO's, that is, GOTO's that do not jump out of a procedure body. PasMesa will insert additional blocks and loops into the translated Mesa program as necessary, in order to handle most combinations of forward and backward local GOTO statements in the Pascal input. There is one thing that PasMesa won't handle, however. Call a pair of GOTO statements head – to – head if one of them jumps backward while the other jumps forward into the loop formed by the backward jump; that is, the two GOTO statements of a head – to – head pair jump into each other's interiors. As long as the Pascal input does not include any head – to – head pairs of GOTO's, PasMesa will output correct and equivalent Mesa code. Any head – to – head pairs that do occur in the Pascal source will engender translation failures in PasMesa that will reveal themselves as error messages from the Mesa compiler.

Some Pascal implementations also allow non – local GOTO's, that is, GOTO statements that jump all of the way out of a procedure body to a label defined in some enclosing block. Consider a non – local GOTO from the point of view of the target label L and the block B in which it is defined. The GOTO statement itself doesn't appear as a statement of the block B (that would be a local GOTO). Instead, some statement in B calls a procedure P, and the GOTO is either a statement in the body of P or in the body of a procedure that P calls. We will define a non – local GOTO to be either forward or backward depending upon whether control goes forward or backward in the block B after it manages to work its way out of the invocation of P. PasMesa handles forward non – local GOTO's by using the Mesa signal/error machinery. But backward non – local GOTO's are a problem. They will be translated into Mesa code that will compile without error, but any attempt to execute the backward non – local goto will result in an uncaught signal. I suspect that handling horrible errors is the only good excuse for non – local GOTO's in any case, and error – handling GOTO's in general jump forward. Thus, I hope that PasMesa's current problems with backward non – local GOTO's will not prove to be a major difficulty.

7. Local Files.

translate a Pascal array is as a Cedar ``computed sequence''. If you want to use this technique, you give the keyword ``ComputedSeqArray'' here, followed by a list of names of the arrays that you want translated this way. The names of the arrays can be fully qualified: ``mem←array'' means the variable ``mem←array'' in the outmost block, while ``output.names'' means the variable ``names'' that is local to the top – level procedure named ``output''. Qualification can go on as deep as necessary: ``a.b.c.name'' will work as well.

The next option has the keyword ``INLINE'', followed by a list of procedure names or function names (or array names, if you have also chosen to implement these arrays with the ProcArray feature discussed below, in which case they are really procedures also). If you specify that a proc should be INLINE, Mesa will translate the body of the proc into the definitions module instead of the implementations module, and will mark it as INLINE. An adroit use of inlines can speed up your program substantially; for example, TeX's main memory array is implemented as an inline ProcArray. But be very cautious about this, since using inlines will make your module size problems worse.

The optional NAMED PARAMETERS clause tells PasMesa that the syntax for parameter procedures is like this:

```
PROCEDURE foo(x: INTEGER; PROCEDURE baz(y,z: INTEGER)).
```

Otherwise it assumes that it is like this:

```
PROCEDURE foo(x: INTEGER; PROCEDURE baz(INTEGER;INTEGER)).
```

The former form is clearly the more rational, and the one that Mesa uses, but the latter has some currency in the Pascal community. The original syntax in the Pascal User Manual and Report is:

```
PROCEDURE foo(x: INTEGER; PROCEDURE baz),
```

which is clearly inferior, since it gives you no clue to baz's parameters. PasMesa will also handle this form, but both it and the Mesa compiler thereby assume that baz takes no parameters. If you call foo giving as a parameter a procedure that takes parameters, the Mesa compiler will complain bitterly (and rightly so). In this paragraph the concept PROCEDURE is intended to include FUNCTIONs. Of course, many Pascal implementations don't allow procedures or functions as parameters at all.

The next option is PROCARRAY, followed by a list of the fully – qualified names of the variables that you would like to implement this way. PasMesa will replace the declaration of the array ``Array[IndexType] of ElementType'' by a declaration of a function that takes IndexType as its argument and returns a LONG POINTER TO ElementType. You get to supply the body of the procedure.

Still one more option about arrays. This switch is called ``SPECIAL ARRAY'', and is followed by a list of fully – qualified names of arrays. The issue here is whether to store the array on the stack or in the heap. If you specify that the target language is Cedar, the default is to allocate arrays in the heap, that is, to replace an array variable by a variable of type LONG POINTER TO an array, with appropriate dereferencing on array accesses. If the target language is Mesa or Long Mesa, then the default is to allocate arrays on the stack. In either case, you can change from the default behavior to the opposite behavior by calling the array SPECIAL. Arrays that are allocated in the heap are allocated from the system UNCOUNTED ZONE. The ones that are local to a procedure are deallocated when that procedure returns, while the global ones are never deallocated.

The Pascal runtime

If you wrote your ``.mod'' file just right, you should be able to shove your program through the Mesa compiler after it has gone through PasMesa by just typing the name that you specified for the MakeFile. This MakeFile will compile the definitions modules first, in the order that you declared them; then, the implementations modules. Finally, it will call the binder on the top – level config.

In order that the resulting bound file should be runnable in Cedar, you will have to arrange that it can get at the services of the Pascal Runtime package.
The runtime services needed by a Pascal program have been divided up into various classes, with different interfaces for each class. Given that there is no way in Mesa to bind up several interfaces into a bigger interface, I couldn't figure out any better way to proceed. The issue is that different Pascal programs want to have different file systems under them, and some Pascal programs don't use Sets at all, while the rest of the runtime stuff is common to all Pascal programs. Hence, there is a PascalBasic with the basic stuff, three different file interfaces, and a Sets interface, along with implementations for each. The PascalNoviceFiles package tries to be really nice to the novice programmer. There is code to make text files avoiding reading one character ahead (as most Pascal files do), so that terminal interaction can work correctly. PascalWizardFiles is a much thinner layer on top of IO.STREAM; this is more to the liking of big applications programs like TeX, which generally open files and the like by calling Cedar procedures that are declared as external to the Pascal program in any case. PascalInlineFiles is an inline version of PascalWizardFiles. Be warned that using the inline version will make the modules into which you have broken your Pascal program somewhat less likely to make it past the size limits of the Mesa compiler.

If you choose to use the PascalNoviceFiles version of the runtime file support, you should set the InventFileNames switch in your ``.mod'' file to TRUE. You will then be working in a world with the following properties : Except for the standard files, all other files by default carry the name of the file variable. To alter the default, you call a procedure PascalAttachFile in PascalNoviceFiles with several arguments including a string containing the name of the file, and after that you RESET or REWRITE the file and everything proceeds normally. I usually insert this call on the Mesa side of the

the ERROR's. Thus, there are six kinds of items all told. In case you haven't guessed already, each statement in the block in the ``.mod'' file tells PasMesa that the definitions code and implementations code generated by the listed top level items is to be placed in the specified definitions module and implementations module.

In addition to giving the names of items explicitly, PasMesa allows you to give defaults in various ways. These defaults look like funny item names that begin and end with a dollar sign, to avoid name conflicts with genuine items. The four names $other←procs$, $other←types$, $other←consts$, and $other←vars$ tell PasMesa where to put those items of each of the four classes that aren't explicitly sent somewhere else. The pseudo – item $main$ means the executable code of the outer – most block, and the pseudo – item $global←labels$ tells PasMesa where to declare necessary ERROR's to implement non – local gotos. In addition, there is a super – default called $rest$, which tells PasMesa where to put everything that isn't classified by the rules above. The Maze program is quite lazy about breaking things up: it puts all defintions into MazePrivate, the external DrawMaze into MazeGraphicsImpl, and everything else into MazeImpl. Only small programs can get away with this easy – go – lucky attitude.

One more comment about the body of the block in the ``.mod'' file, concerned with EXPORTing. PasMesa has to figure out which of the implementations modules in a large system EXPORT which definitions modules. Instead of asking you for this information explicitly, PasMesa figures it out in the following devious manner: if there is a line of the ``.mod'' file of the form

```
ADefs, BImpl : = stuff
```

in which some stuff is stuck in ADefs and BImpl, then BImpl is assumed to export ADefs. You can prevent PasMesa from drawing this conclusion, if necessary, by putting an asterisk after ADefs, as in the line

```
ADefs*, BImpl : = stuff
```

Also, as mentioned above, ADefs won't be exported by anybody if it was declared to be trash.

After the ``end'' in the mod file comes a right paren and an assignment operator followed by the name of the source file where the Pascal program can be found.

The mod file finishes up with list of switch names and switch settings by which the behavior of PasMesa can be appropriately modified for different situations. The switches must come in the correct order, and that order is alphabetical according to the name of the switch. The Maze example has only a few switches. The InventFileNames switch (remember, case is not significant in Pascal text) tells PasMesa to assume that the PascalNoviceFiles version of the file IO portion of the Pascal runtime package will be used. This version of file support tries (or tried, back when Ed wrote it) to be very nice to you. In particular, it tries to allow you to think of files in Pascal's unusual manner, as a sort of extensible record. The files are actually stored on disk, and the name of the file on the disk is derived from the name of the file variable in your Pascal program. To make this work, PasMesa has to generate some calls to a routine from the PascalNoviceFiles runtime package; the InventFileNames switch requests this service.

The MakeFile clause gives the name of the file on which you would like the compile – and – bind command file to be written; the default extension is ``.cm''. In the example, the command file will be called CompileMaze.cm.

The PREDEFINE clause is a place where you can put the declarations of procedures that will be available to the Pascal program even though they are not defined there. The block that you give here is treated as if it enclosed the outer block of your program. If editing your Pascal source program isn't a big problem, another alternative is to include such external function definitions in the outermost block there. In either case, the implementations of the external functions are done directly in Cedar.

Maze is a simple program, and doesn't stretch PasMesa to it's limits; in fact, there are several switches that Maze doesn't need. Our next subject is a discussion of the rest of the possible switches and what they do.

There is an optional CAPITALIZE clause because of the unfortunate history of Pascal: in the original implementation, the character set was a 64 – character set not even vaguely related to Ascii. PasMesa implements CHAR as full Ascii, including upper – and lower – case characters, but you may come across a program that is not prepared to deal with lower – case characters, even though they appear in the Pascal source file, because the programmer expects the Pascal compiler to capitalize everything. If you say CAPITALIZE CHARS, string constants that PasMesa represents as single characters or as packed arrays of characters be capitalized. If you say CAPITALIZE EVERYTHING, PasMesa will even capitalize things that it will represent as Mesa strings. Mostly what this means is that the string constants that you write with WRITE or WRITELN will be capitalized. If you omit this clause, PasMesa will compile exactly the case you use in string or character constants.

The optional CompilerSwitches clause is followed by a Pascal string (written with single quotes; remember, you are talking to a Pascal compiler!); this string will be put into the compile – and – bind command file at the place where Mesa compiler switches belong. For example, we chose to compile the Mesa TeX with bounds checking and nil checking turned off, since this helped us out a little with our storage overflows in the Mesa compiler. Thus, the file TeX.mod includes the clause ``COMPILERSWITCHES '/ – b – n';''.

Large arrays in Pascal programs cause PasMesa no end of hassle. One of the ways that you might want to

module. Any module that is imported by the config counts as declared, of course. Usually, you will want to declare all of the definitions modules first and then the implementations modules, so that any of the latter can take any of the former as arguments. The order in which you declare modules is also the order in which they will be compiled, by the way.

What does it mean for one module to take another as an argument? It usually means to DIRECTORY, IMPORT, and OPEN it. Pascal has no sense of name scopes local to a module, and PasMesa only outputs unqualified Mesa names. Thus, a module has to OPEN anything that it DIRECTORY's in order to get any good out of it. Note that there can't be any name conflicts in the Mesa version of the program because there weren't any name conflicts in the Pascal version. IMPORT'ing is a bit of an issue, though. The compiler will issue a warning message if a module IMPORT's something that it doesn't really need. Furthermore, you can't turn off this warning message without turning off all warning messages, which would be rather dangerous. Thus, PasMesa offers the following feature: if you put an asterisk after the name of an argument in the declaration of a module, the declared module will DIRECTORY and OPEN but not IMPORT that argument. I recommend that you leave the asterisks off until you get warnings from the compiler, and then add them as needed.

A word for wizards: When one module takes another as an argument, the argument module is almost always a definitions module rather than a program module. But not always. It is possible for a module to DIRECTORY and IMPORT a program module; the former is said to be taking the latter as a POINTER TO FRAME. Doing so has definite disadvantages: it introduces a compile – order dependency between implementations modules and it confuses the debugger in many cases. But it also has an advantage that is relevant to PasMesa users: the importer gets access to all of the global variables of the importee, and the importee can have lots and lots of variables. You see, implementations modules have more slots in their global frames for variables than do definitions modules. To make a long story short, PasMesa does allow you to declare a program module that takes another program module as an argument; do so only at your peril.

Each declaration of a module ends with one of three words: either ``forward", ``external", or ``trash". The keyword ``forward" corresponds to the normal case; PasMesa will produce source text for a forward module, will request that it be compiled at the appropriate time, and will request that it be bound into the resulting configuration. The keyword ``external" is used for modules that are to be bound into the configuration, but should neither be written by PasMesa nor compiled anew. The prime examples of ``external" modules are the modules that implement the Pascal runtime environment. The program Maze imports the pascal runtime from the outside world, but many Pascal programs choose to bind the pascal runtime routines into their configs. Such programs declare the various modules of the runtime and specify them to be ``external". The third possibility is ``trash": PasMesa won't write, compile, or bind a module that is declared to be ``trash"; this gives a way of naming a place to put things that you don't want. In summary, the three options can be thought of as follows:

forward: write this module, compile it, and bind it
external: don't write or compile this module, but do bind it
trash: don't write, compile, or bind this module.

The list above tells you what the various options mean for implementations modules, but the situation is a little different for definitions modules. In the definitions case, the notion of binding doesn't make sense. Instead, the important question is whether or not various implementations modules should be marked as exporting this interface or not. The basic rules by which PasMesa figures out who exports what are explained below. But they are modified for the case of a definitions module that is declared to be trash: PasMesa will guarantee that no implementations module exports a trash interface.

Everything in Maze.mod should make sense now up through the keyword ``begin" except for the funny extension specified for MazeGraphicsImpl. What is going on here is the following. The procedure that draws the maze is easier to write in Cedar directly, since it wants to call procedures from Graphics, GraphicsToPress, and the like. If you give an explicit extension on any file name, PasMesa will use the filename that you have specified; if not, it will use the appropriate default extension for the file: either ``.cm" or ``.mesa" or ``.config". In this case, PasMesa will write a header for the DrawMaze procedure into MazeGraphicImpl.hint$ with a body of ``??", pointing out that you must write this procedure yourself. When you replace ``??" by the correct body, you should store the result out on the file MazeGraphicImpl.mesa, where the compiler will be able to find it. Future runs through PasMesa will now avoid smashing your hand – written file because the PasMesa output will once again be written on the file MazeGraphicImpl.hint$.

Between the ``begin" and the ``end" are a sequence of lines that look sort of like funny assignment statements. Each left hand side must have precisely two predeclared module names; the first must be a definitions module and the second an implementations module. On the right – hand side of the statement is a sequence of names of top – level Pascal items. PasMesa is prepared to help you split up the outermost block of your Pascal program. Splitting up inner blocks is much harder, and PasMesa offers you no help in that regard; so, if your Pascal program has a single procedure that is too long to fit, even all by itself, into a Mesa module, then you are in trouble. But PasMesa does allow you to split up the outermost block however you see fit. Call each thing in the outermost block that could generate corresponding Mesa code an item. There are four major types of items: types, constants, variables, and procs (procedures and functions). The executable code on the outermost block is also an item by itself. And it turns out that labels in that executable portion can also generate Mesa code, since non – local gotos to those labels are implemented using ERROR's, and someone has to declare

and short Ropes, and hence tends to pollute your virtual memory quite severely. I recommend that you Rollback after running PasMesa extensively, and perhaps before as well.

Running the Compiler

The compiler is called PasMesa. You can invoke PasMesa itself on a particular program by typing

```
PasMesa maze
```

where ``maze.mod'' is the name of a text file that you must have already made up that gives PasMesa its instructions. I shall use the ``maze.pas'' program as an example.

The ``.mod'' file gives PasMesa basically two kinds of information. First, most significant Pascal programs are too large to fit into one Mesa module that the current Mesa compiler can digest. To address this problem, PasMesa has facilities for taking the various components of the outermost block of the Pascal input program and parceling them out to a collection of Mesa modules. The ``.mod'' file declares these Mesa modules and tells PasMesa what outer block items to put where. After this modularization information, the ``.mod'' file also allows the user to specify the settings of various switches and parameters.

The format and features of ``.mod'' files are rather complex, rather like those of DF files. Let's look at ``maze.mod'':

```
(configuration maze(graphics, graphics←to←press, random, pascal←basic,
                                                                    pascal←novice←files);

definitions maze←private(pascal←basic*); forward;
program maze←impl(pascal←basic, pascal←novice←files, maze←private, random);
        forward;
program maze←graphics←impl.hint$(graphics, graphics←to←press, maze←private,
        pascal←basic); forward;
begin
   maze←private, maze←graphics←impl : = draw←maze;
   maze←private, maze←impl : = $rest$;
end) : = maze.pas;
INVENTFILENAMES TRUE;
MAKEFILE compile←maze;
PREDEFINE
function choose(i,j: integer): integer; external;
function init(i,j: integer): integer; external;
begin
end
TARGET cedar;
```

You're probably wondering right off why there are back – arrows in the middle of the identifier names. Those characters have code '\137. In the Xerox character set, that is a back – arrow all right. But in ASCII, it is an underline; as such, it is the character generally used when writing Pascal code to separate the words in multi – word identifiers. PasMesa, like all Pascals, ignores the case of characters in its input. When translating an identifier to Mesa, it capitalizes the first letter and makes the rest lower case. If the identifier contains any underscores, they are removed, and the letter immediately following the underscore is made upper case as well. Thus, the Pascal identifier ``graphics←to←press'' will be translated by PasMesa into the Mesa identifier ``GraphicsToPress''. It's a little funny at first, but you'll get used to it. Remember that assignment in a Pascal program uses the operator ``: = '' rather than a left – arrow. For more details and a pointer to a nifty hack program named CapsArrows that will help you translate back and forth between these two identifier syntaxes, see Restriction 1 below.

The first half of the mod file looks like a bastard form of Pascal block. You should think about it as a declaration of the configuration that the Pascal program is going to be translated into. PasMesa assumes that every Pascal program will be broken up into a number of modules that will be bound up in one binding step into the final configuration. The identifier after the word ``configuration'' is the name of that config; in our example, that name is ``maze'' in Pascal form, which translates into ``Maze'' in Mesa form. PasMesa will write the config itself onto the file named ``maze.config''.

The identifiers in the parentheses after the word ``maze'' are the arguments to the config, that is, they are the definitions modules that this config will import. Therefore, the config that PasMesa writes will have a header of the form:

```
– – file: Maze.config
– – Pascal – to – Mesa Configuration

– – Pascal – to – Mesa translator output, translated at January 22, 1984 4:22 pm

Maze: CONFIGURATION IMPORTS Graphics, GraphicsToPress, Random, PascalBasic, PascalNoviceFiles
```

The next three lines declare the modules that are involved. Definitions modules are declared with the keyword ``definitions'' and implementations modules are declared with the keyword ``program''. Each module that you declare takes, as arguments, the names of the modules that it depends upon. PasMesa is a one – pass operation: you must declare a module before you use it as an argument to another

– – PasMesa.doc

The PasMesa translator has been converted to Mesa to run in XDE. Following are some insights and restrictions for the current state of PasMesa.

* PasMesa is a memory hog. It cannot in good conscience be called a Mesa program since it has no concept of storage deallocation. In particular, the global frames for all the modules require at least 4K, most of which is string literals. Also, PasMesa will trash up your system heap with a lot of garbage it doesn't collect. The moral is, if you run PasMesa, do it in your Tajo volume where it can't do much harm. Either that or reboot soon after using PasMesa.

* Only the PascalNoviceFiles version of th runtime i/o support has been implemented. The other versions (PascalWizardFiles and PascalInlineFiles) haven't been touched.

* The standard files Input and Output initially are set to the executive.

* The TARGET language must be LONG MESA.

* The resulting Mesa code will be FULL of warnings, mostly about long integers being truncated, and usually around for loops.

You should retrieve PascalBasic.bcd, PascalBasicImpl.bcd, PascalNoviceFiles.bcd, PascalNoviceFilesImpl.bcd, all from the subdirectory of the directory.

Included in the source directory are the pascal source, .mod file, and resulting mesa files for a benchmark program, Bench2*. It will give you a start. Following is the original documentation from the people over at PARC.

Abstract

PasMesa is a Pascal – to – Mesa source translation tool. It can help import Pascal programs into the Mesa environment.

An Overview of PasMesa

The most straightforward way of importing a relatively small amount of Pascal software into a relatively large Mesa environment is to translate the Pascal as directly as possible into Mesa source code. The translation is straightforward precisely because of the simplicity of Pascal and the similarity of the two languages. I have implemented in Mesa a Pascal – to – Mesa translator called PasMesa. A fair thing to say about this translator is that it has compiled some Pascal programs to Mesa, and, usually after some modifications, the programs have worked. It is a bit early to claim that PasMesa will translate any Pascal program to runable Mesa. In any case, PasMesa's translations are not perfect. I assume that the person doing the translation is both a Pascal and a Mesa programmer, and that he is prepared to do an average of 15 to 30 minutes' work per (paper) page of source code to repair the translation where it failed. All known translation failures result in errors that are caught by the Mesa compiler (except for backward non – local GOTO statements, which result instead in uncaught signals; see Restriction 6 below).

Error recovery in PasMesa is non – existent, and the error reporting is only rudimentary. PasMesa is not at all intended as a means by which one could reasonably develop a Pascal program from scratch. I make no guarantees about what will happen if the Pascal program will not pass through somebody's standard production Pascal compiler without complaint.

The Pascal program is assumed all to fit on one (perhaps gigantic) Pascal source file. To coalesce a multi – module program into a single module looks like a straightforward editing task; however, I might be receptive to a chorus of complaints in this area.

One final generality: PasMesa runs rather slowly. Just be patient. Remember, it's faster than doing it with a text editor.

Further generalities

Most of these new features are connected with making it possible to do the translation of a large system completely automatically. That is, I considered it cheating to have to modify the resulting Mesa code by hand; instead, I would add some other funny feature to PasMesa itself.

The code that PasMesa outputs is intended to run in XDE. No effort has been made to produce a version of the Pascal runtime routines in Alto Mesa 6 that is compatible with the new PasMesa.

Not only does the PasMesa run slowly, it also runs impolitely! It does many, many allocates of long

- - ResetMapLog.doc

FUNCTION

ResetMapLog runs in Copilot ONLY and allows you to have multiple remote debugging sessions of the same debugee.

BACKGROUND

When you open a remote debugging session, CoPilot processes map logging information about the debuggee. This allows CoPilot to meaningfully interpret the state of the debugee. After CoPilot has processed the map log, it resets the debugee map log pointer, thereby allowing the debugee to reuse entries in the map log. If you try debugging from a CoPilot that had not originally processed all of the debugee maplog, CoPilot will detect an ***Invalid Load State*** and you will not be able to debug. ResetMapLog allows you to side step this problem by resetting the map log pointer to the value it had when you first started debugging. Therefore each debugger will always see the the entire map log.

UTILIZATION

To use ResetMapLog you must first open a remote debugging session to a debugee, via the CoPilot Remote Debugee command. Next, run ResetMapLog in the Executive and that's it. You may then examine the state of the debuggee. If you don't find anything of interest to you, you can pass the buck and allow anyone else to continue teledebugging by starting a remote debugging session from their CoPilot. ResetMapLog was specifically designed to allow you to discover why a machine had crashed without being stuck with having to completely debug it. You can start a teledebugging session, poke around and then let someone else debug it, if necessary.

PRECAUTIONS AND DISCLAIMERS

The behaviour of CoPilot is unknown if ResetMapLog is run and then the debugee is debugged from the same CoPilot which processed all of the original map log info. This means that CoPilot is processing the same map log info twice, and that may lead to unknown and unsavory side effects.

ResetMapLog must be run after the teledebugging session is completely opened and before you want to disassocate yourself from the debuggee, either by Proceeding, or by just allowing another person to debug from their own CoPilot. It will not work if you open a session, set a breakpoint, proceed the debugee, return when the breakpoint is taken, and THEN run ResetMapLog. After that, any attempts to debug the debugee from a different CoPilot to result in an ***Invalid Load State***.

Appendix A

Status and Error Messages

Courier Error = > caller aborted

Courier Error = > duplicate program export

Courier Error = > invalid arguments

Courier Error = > invalid handle

Courier Error = > invalid message

Courier Error = > no answer or busy

Courier Error = > no courier at remote site

Courier Error = > no such procedure number

Courier Error = > no such program export

Courier Error = > no such program number

Courier Error = > no route to system element

Courier Error = > parameter inconsistency

Courier Error = > protocol mismatch

Courier Error = > remote system element not responding

Courier Error = > return timed out

Courier Error = > stream not yours

Courier Error = > too many connections

Courier Error = > Transmission medium hardware problem

Courier Error = > transmission medium not ready

Courier Error = > transmission medium unavailable

Courier Error = > transport timeout	
Courier Error = > unknown error in remote procedure	
Courier Error = > ****unknown error reason	
Remote error: insufficientSpace	**Start**
Remote error: invalidaddress	**Start**
Remote error: invalidLineNumber	**Start**
Remote error: not Implemented	**Start**
Remote error: notStarted	**Start**
Remote error: procedureNotFound	**Start**
Remote error: transmissionMediumDown	**Start**
Remote error: invalidFileName	**Retreive/Playback**
Remote error: No Data	**Retreive**
Remote error: notStopped	**Retreive**
Remote error: ???	unknown Error
Bad remote address specification	**Start**
Clearinghouse lookup problem	**Start**
Circuit has terminated	**Start**
Buffer Full at Server	**Start**
Specified Port is Active	**Start**
Waiting...	**Start**
GateSpy.Reason = other	
...ABORTED	**Stop (or Stop Key)**
Data collection stopped	**Stop**
Data retreived and stored	**Retreive**
End of File reached	**PlayBack**
Interval out of Bounds	**PlayBack**
File is of incorrect type	**PlayBack**
Parser not bound/implemented	**PlayBack**

-- TIPTest.doc

TIP Test provides a way of testing TIP tables before you commit them to a system that you're going to try to run on. It lets you parse tables to check for syntax errors, then lets you try out the productions to see if the correct results are being generated.

The tool consists of the usual message subwindow and form subwindow, with an additional large window at the bottom for testing the new tables.

The commands available are:

> **Create** – Calls TIP.CreateTable with the filename in the Name: field of the subwindow. If the parse is successful, the new TIP.Table is associated with the bottom subwindow of the tool. At this point, only real – estate events are sent to the window. This command calls the Destroy command before attempting any of this.
>
> **Destroy** – removes the TIP.Table, if any, from the bottom window and releases the storage associated with it. Note that since there is no direct way to destroy a TIP table, this is done by creating a private heap for the table and deleting it for this command.
>
> Take Input – if the tool has a TIP.Table in its hands from a previous Create, this command causes the bottom subwindow of the tool to become the current input focus. This allow all events, not just real estate events to be passed to the tool.

The bottom subwindow of the tool has TIP.NotifyProc which simply prints the TIP.Results passed to it, one line per call. Results elements are separated by commas. Individual results are printed in the following formats:

char = >	quoted character
coords = >	[x: 0, y: 0]
keys = >	<KEYS>
atom = >	pname of the atom
int = >	a long integer
string = >	quoted string
time = >	[14514352345B]

(e.g., NSMailChecker).

To generate picture bitmaps use TinyPictureTool.bcd. For the less artistic there is a collection of "standard" icons, TinyWindow.icons. As noted above, this file contains several versions of some icons; initialization will go more quickly if you delete the ones you don't want. Likewise, removing leading zeroes from the bitmap numbers will reduce the file size and thereby dramatically improve startup time, at the cost of readability.

A sample TinyWindow.icons file would look as follows:

[TinyWindow.Icons]

- - Copyright (C) 1984 by Xerox Corporation. All rights reserved.

[Icons] - - you might also have separate [CoPilot:Icons] and [CoCoPilot:Icons]
Tool: Chat 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 17777B 0 1777B 100000B 10001B 0 7777B 160000B 13775B 0
17777B 170000B 12005B 0 17777B 170000B 12005B 0 16000B 70000B 12005B 177777B 176306B 74000B 12005B 0
16306B 72000B 13775B 0 777B 4000B 10001B 0 777B 2000B 17777B 0 1777B 104000B 3774B 0 1727B 102000B
37777B 100000B 3653B 144000B 20000B 100000B 3727B 150000B 25252B 100000B 7653B 164000B 22524B 100000B
7727B 170000B 25252B 100000B 7777B 160000B 20000B 100000B 7777B 160000B 37777B 100000B 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Tool: CommandCentral 0 0 0 0 30710B 136020B 362B 24200B 45055B 121060B 212B 26200B 41052B 121177B
176212B 25200B 41050B 136377B 176362B 25200B 41050B 120177B 176222B 25200B 45050B 120060B 212B 24600B
30710B 120020B 211B 144200B 0 0 0 0 0 0 0 0 16000B 0 0 40000B 16000B 0 0 160000B 16000B 0 1 170000B
16000B 0 3 174000B 16000B 0 0 160000B 16000B 0 0 160000B 16000B 0 0 160000B 16000B 0 0 160000B 16000B
0 0 160000B 16000B 0 0 160000B 16004B 17121B 74000B 160000B 17006B 10531B 42001B 160000B 7777B 10525B
42377B 140000B 3777B 117125B 42377B 100000B 1777B 10525B 42377B 0 6 10523B 42000B 0 4 17121B 74000B 0
0 0 0 0
Empty: 0 0 0 0 0 17777B 0 0 0 17777B 100000B 0 0 14007B 140000B 0 0 14006B 160000B 0 0 14006B 70000B 0
0 14006B 34000B 0 0 14006B 16000B 0 0 14007B 176000B 0 0 14007B 176000B 0 0 14000B 6000B 0 0 14000B
6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B
6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B 6000B 0 0 14000B
6000B 0 0 14000B 6000B 0 0 17777B 176000B 0 0 17777B 176000B 0 0 0 0 0
Loaded: Calendar 177777B 177777B 177777B 177700B 100100B 40100B 40100B 40100B 106121B 57521B 57536B
46100B 111133B 42121B 42120B 51100B 104125B 42121B 42120B 44100B 102121B 42125B 42134B 42100B 111121B
42133B 42120B 51100B 106121B 42121B 42120B 46100B 100100B 40100B 40100B 40100B 177777B 177777B 177777B
177700B 100100B 40100B 40100B 40100B 100100B 40100B 40104B 46100B 100100B 40100B 40114B 51100B 100100B
40100B 40104B 41100B 100100B 40100B 40104B 46100B 100100B 40100B 40104B 50100B 100100B 40100B 40116B
57100B 100100B 40100B 40100B 40100B 177777B 177777B 177777B 177700B 100100B 40100B 40100B 40100B
106122B 57114B 47514B 46100B 111122B 50120B 40522B 51100B 102122B 56134B 41114B 51100B 101136B 41122B
42122B 47100B 111102B 51122B 42122B 41100B 106102B 46114B 42114B 46100B 100100B 40100B 40100B 40100B
177777B 177777B 177777B 177700B
Gray: TRUE
Verbose: FALSE

- - TinyWindowPictures.doc

TinyWindowPictures allows Tajo users to customize their environments by allowing the user to specify icons for tiny windows. TinyWindowPictures will read bitmaps from the TinyWindow.icons file and place the pictures in a tool's tiny window. Hence users can create icons for individual tools to represent their functions.

To run this program:

1. Retrieve TinyWindowPictures.bcd
2. Create a TinyWindow.icons as specifed below.
   (Or Retrieve TinyWindow.icons from the diskette - note: this file has several versions of some icons. For better quicker initialization, it is best to decide which version you like best and delete the others from the file, rather than simply commenting them out.)
3. Type into the Executive:
   TinyWindowPictures

The program reads the icon file only once, when started, and converts the bitmaps to a binary form, ready to spray onto the screen. It then enumerates existing windows to see if any have icons, replacing the DisplayProcs of those that do. Subsequently, whenever a tool or file window is created, it searches for the name in its collection of bitmaps, and replaces the normal tiny name display with the stored bitmap. If a tool is inactive, it will be converted when it is activated.

In addition to the TinyWindow.icons file, TinyWindowPictures puts a menu on the root window, with two items, SetIcon and Verbose/Quiet. Selecting SetIcon changes the cursor to a bullseye, then clicking Point over a tool window takes the current selection as a bitmap and makes that the bitmap for the selected window. This is useful for "debugging" icons, and for setting icons on tools that have already been started without icons.

Verbose/Quiet changes the value of the "verbose" flag to the value shown in the menu. If Verbose is true, TinyWindowPictures will put a message in the herald window every time it fails to find an icon for a window, noting the name that it used for the search. This is useful for tools for which the exact name is not known, or which have no User.cm section and also have an awkward name - - the StarFileTool, for instance, used to need an entry named "StarFileToolof". (It now has a User.cm section.) Verbose mode defaults to false, and can be set either by TinyWindow.icons or by the menu.
Note: the program waits two seconds after posting, to avoid having messages scroll off the screen before you can read them, so loading goes much more slowly in this mode if there are several tools without icons. Recommended usage is to turn on Verbose before starting a new tool with an unknown name, then turn it off after noting the name in the herald window.

There are 7 kinds of bitmap entries in TinyWindow.icons, and two booleans. The type and format of the bitmap entries is:

NSHardy: <bitmap> - - bitmap for NSHardy when there is no new mail, or no new mail icon.

NewNSMail: <bitmap> - - bitmap for NSHardy, when there is new mail.

Tool: <toolName> <bitmap> - - bitmap for tool toolName, where toolName is either the tool's User.cm section title, or its tiny name.

Empty: <bitmap> - - bitmap for empty file windows.

Loaded: <fileName> <bitmap> - - bitmap to use when a file window is loaded with file fileName. fileName is NOT qualified with a directory.

The two boolean options in TinyWindow.icons are:

Verbose: <boolean value> - - sets the initial value of the "verbose" flag, as described above. Default is FALSE.

Gray: <boolean value> - - If TRUE, file windows that are being edited will have a "dirty" gray background on their tiny windows. Default is TRUE.

To see the name of a tiny window that has an icon, place the mouse cursor over the tiny window and hit HELP. If the HELP key is depressed when the cursor is removed from the tiny window, the original text will remain displayed. To get the icon back, move the cursor over the tiny window and hit HELP and let go of the key. Also, any of the seven forms of <bitmap> entries in TinyWindow.icons can have a boolean tacked on the end. If this boolean is TRUE (default is false), then the standard tiny text will be displayed whenever the cursor is moved into the tiny window. This is useful for the CoPilotHerald, and also for other tools that present information by changing their tiny names

- - TinyTidy.doc

Environments: CoPilot|Tajo
Documentation: TinyTidy.doc.
Description: TinyTidy is a program that reorganizes tiny windows. It reclaims tinywindow screen space lost by destroyed tools. TinyTidy reassigns tiny window slots so that each tool window has a distinct tiny window slot from other tool windows. The order of the tiny window slots is preserved from left to right and bottom to top for all but inactive windows. Overlapping tiny windows will be assigned sequential slots in an arbitrary order. Inactive windows are not affected. Also, a User.cm entry can specify a list of tools to ignore. TinyTidy registers TinyTidy.~ with the Exec.
The User.cm entry is as follows:
[TinyTidy]
Ignore: CoPIlotHerald Activity

It is intended for tools that are rarely made tiny, and thus would otherwise take up slots that would never be used.

– – TinyPictureTool.doc

This is an outgrowth of the old CursorTool that can be used to produce the 58 x 28 bitmaps used for pictures in tiny windows. (See TinyWindowPictures.doc.)

Mouse actions

Point:	Turn the square black.
Adjust:	Turn the square white.
Menu:	Pick up the picture (left corner) in the cursor. If Menu comes up below or to the right of the grid, deposit the picture on the screen.

Commands

clear!	Turn all squares off.
read!	Take the current selection as an array of numbers, OR this picture into the grid. Non – numeric characters ahead of a number are ignored; thus it usually works if you line – select in TinyWindow.icons.
comp!	Complement the value of each square.
write!	Write the current picture in the log as 112 numbers (4*28). The low order 6 bits of every fourth word are zero.
left!	Shift entire picture one column left.

right!, up!, down! Likewise.

The write! command writes to the log subwindow and into the file PictureTool.log. The tool uses Tool.UnusedLogName so more than one can be run at once.

The text written by write! has any of four formats, selected by an enumerated item in the form subwindow.

verbose:	six – digit octal numbers separated by commas, with a pair of brackets surrounding the whole set; this is suitable for initialising an array in a Mesa source
compact:	octal with no leading zeroes, no commas, and no brackets; one – digit numbers have no "B" appended; suitable for TinyWindow.icons
decimal:	same as compact but in decimal; less readable (perhaps) than compact, but takes fewer characters in an icons file
signed:	decimal, but uses negative numbers for the range – 999 to – 1, thereby using still fewer characters; using signed format in place of compact typically saves about 20% of a TinyWindow.icons file and hence can save several seconds when starting TinyWindowPictures.bcd

Any of the four formats can be read by the read! command. The default format is compact, but this can be set by including a Format item in the [TinyPictureTool] section of User.cm, e.g.:

[TinyPictureTool]
Format: signed

Time.Append.

Pack[T1]!
Semantics: treats T1 as a date, packs it with Time.Pack and displays that value.

- - TimeWarp.doc

TimeWarp is a tool for manipulating dates and quantities of time. It may be used to determine the difference between two dates or times or the sum of two quantities of time, or to convert seconds to hh:mm:ss format, or to pack and unpack time values. The tool has a form and a file subwindow. The form subwindow contains the string fields T1 and T2, (arguments for functions), and the Functions T1 + T2, T1 - T2, Now, Epoch, Pack[T1] and Unpack[T1].

Arguments:

T1 and T2 are arguments for functions, and may be in Date, Time, or packed time format. The type of each argument is indicated by the format of the contents of that field.

A value of date may have the formats exemplified here:

27 - May - 1982 16:41:22 EST	
27 - May - 82 16:41:22 EST	(27 - May - 1982 16:41:00 EST)
27 - May - 82 16:41:22	(27 - May - 1982 16:41:00 in the local time zone)
27 - May - 82 16:41	(27 - May - 1982 16:41:00 in the local time zone)

A quantity of time is some number of hours, minutes and seconds, and is not relative to a particular date. Time may be entered in these formats:

16:41:22	(hh:mm:ss)
16:41	(hh:mm)
60060	(ss)

Packed time is a value of the type Time.Packed

2568843682	(same as 27 - May - 1982 16:41:22 PDT)

Results of operations are written in the file subwindow. Results will have the type Date or Time, depending on the types of the arguments and the operation invoked. Results of the type Date are given in unpacked date format and packed time format. For example,

27 - May - 82 16:41:22 PDT = 2568843682

Results of the type Time are given in number of seconds, "days hh:mm:ss", and "hh:mm:ss". For example,

12345678 = 142 21:21:18 = 3429:21:18

Operations:

If one of the arguments of an operation has a disallowed type or was not entered in an exceptable format, the display will blink and the operation will not be performed.

T1 + T2!
Add the quantity T1 to T2. The type of the result depends on the types of the arguments.

(Date + Time) yields a Date
(Time + Time) yields a Time
other combinations are not supported

T1 - T2!
Subtract the quantity T2 from the quantity T1. The type of the result depends on the types of the arguments.

(Date - Date) yields a Time
(Date - Time) yields a Date
(Time - Time) yields a Time
other combinations are not supported

Now!
Display the current date and time.

Epoch!
Display the value of System.gmtEpoch. Note that Time.Unpack interprets the argument of System.gmtEpoch as a request for the current date and time, and so the current date and time is displayed along with the digital representation of the epoch.

Unpack[T1]!
Treats T1 as having the type Time.Packed, and converts it to a formatted date and time string with

-- TimeTracker.doc

TimeTracker is a program which keeps your local clock synchronized with the time service. Every two hours (roughly) it wakes up and polls the time service in order to calculate an accurate time (this operation is hereby called a resync). TimeTracker also does a resync when it is running in CoPilot and CoPilot is reentered.

In doing the resync, TimeTracker assumes that a certain percentage of the time servers might have the incorrect time. The larger the percentage assumed incorrect, the less accurate your clock might be. If TimeTracker can determine that the percentage has been set too small, it will log this fact and temporarily assume a higher percentage.

TimeTracker registers the command TimeTracker.~ with the executive. This command can be one of the following forms:

TimeTracker.~ start	Start the tracker
TimeTracker.~ stop	Stop the tracker
TimeTracker.~ resync	If tracker started, causes a resync
TimeTracker.~ information	Display the current state of the tracker
TimeTracker.~ log	Display the last thirty events
TimeTracker.~ params pct/P reset/R	Sets the max percent of faulty servers to pct (default 5%, must be in range [0.. 100]) annd sets the period between resets to reset minutes (default approximately 120 minutes).
TimeTracker.~	Display the date and time

The tracker initializes to started with pct = 5 and reset = approximately 120.

In addition to the tracker, this program creates a background process which checks for an invalid clock (currently, one which appears to be fast by more than four hours of the real time). If it finds an invalid clock, it will blink the display and post a message in the Herald window every fifteen seconds. If this happens, check your clock. If it looks ok, the time service is probably messed up. If your clock looks incorrect, you should reboot.

You can stop the blinking mentioned above by unloading TimeTracker.~.

- - TicTacToe.doc

TicTacToe.bcd is a 3 - dimensional game of TicTacToe consisting of a 4 planes of 4 by 4 grids. This game is played with two people both running TicTacToe.bcd. To play, you and a friend must retrieve TicTacToe.bcd. Running TicTacToe will cause a window to be displayed.

To start a game between two people:

0. Both players must run TicTacToe.bcd
1. Enter opponent's machine's name in the Opponent: field. The opponent's machine's name is the name of the opponent machine found in a clearinghouse. T One could also use a fully qualified name, ie UserName:Domain Name:Organization.(Actually, one could use any input that AddressTranslation would understand.)
2. At this point, there must be a minimal amount of coordination between the two players. That is, the players must decide on who is going to hit the StartGame! command in the games formSW. After one player hits StartGame!, the playing board will be cleared, a new formSW will appear, and the game is started. Whoever hits StartGame! has the option of being first, by selecting the ImFirst boolean on the formSW.
3. It is your turn to move when YourTurn(a read only boolean item) is highlighted on the new formSW. To move, move the cursor over the desired cell and select it (using the left mouse button). This will cause an icon to be placed in your cell. The YourTurn will be de - highlighted and you will notice that you cant select another cell. This is because it is now you opponent's turn. After you opponent moves, it will be your turn again. If you are running Play.bcd , some a little tune will be played when it is you turn. One can specify this tune in the User.cm (explained below).
4. In the new formSW, there is a Message: field, where a player can type in a message and hit SendMessage! and that message will appear in the opponent's message subwindow.
5. If you want to quit a game, it is best to wait until it is your turn, then hit QuitGames!, and then make a final move. This should cause the game to terminate with both players. The old board will remain displayed until a new game is started. Also the original formSW will appear. Note: the game will not allow a user to deactivate it until a game is terminated.
6. When one player eventually wins, the winner will hear a winner's song and the loser will hear a loser's song and both will be notified of the result(provided the players are both running Play). The win and lose tunes can be specified in the User.cm (explained below).

Customizing you TicTacToe game:

The user can specify his game piece, the opponent's game piece, a move tune, a win tune, and a lose tune, as well as the standard WindowBox, TinyPlace, and InitialState in the User.cm. In my User.cm I have:

[TicTacToe]
WinTune:g - % - %%g - g - g - % - %%g - g - gecegeg>c< gecegeg>c<g
LoseTune:@360,135>CccFFFFFF%%%CcfAAAAAA%%%
MoveTune:*>ccdeced
MyPiece:1700B, 7760B, 7760B, 17170B, 17170B, 17770B, 147763B, 147763B, 140603B, 141703B, 157773B, 175737B, 141703B, 141703B, 140603B, 140603B
YourPiece:017770B, 010010B, 013750B, 012050B, 012050B, 032054B, 063742B, 112525B, 105251B, 177777B, 000000B, 077776B, 100001B, 100001B, 100001B, 077776B

I would recommend having a very short MoveTune since it is played often. Use Play to generate these tunes. In Play.music are a bunch of tunes you can borrow.

Known Bugs:

1. If both players hit StartGame!, the one player will find that he will not be able to do any more StartGame!'s. This is a synchronization problem that will be fixed in the next release of TicTacToe.
2. If two people are playing TicTacToe, and a third person tries to play with one of the two currently playing people, funny things will happen.

Helpful Hints:

1. If you cant get a game started with another player, any of following may be true:
   a. The other player is not running TicTacToe.
   b. The other player's machine is not registered with you default clearinghouse.
   c. The other player login name is not the same as the machines name registered with the clearinghouse.

-- TextSWLineHack.doc

TextSWLineHack adds a new command, "Line", to the Text Ops menu in any text subwindow (including variants such as file and exec sws). This command is similar to the "Position" command, except that it repositions by line number rather than by character position. For example, if you select "10" and invoke "Line" in a file subwindow, the tenth line of the file will become the top line in the subwindow. If the selection is not a number, is 0, or is greater than the number of lines in the text, the screen will flash and the text subwindow will remain unchanged.

= = = = = =

EXECUTE

- - - - - - -

Mq Execute string or buffer contents of Q – reg q as a TECO command.

ITERATION

- - - - - - - - -

n< cmd > Command is executed n times, or indefinitely if n is null.

n; Does nothing if n<0, otherwise passes control to char after next ">", i.e., terminates iteration. Null n = > use value of last search.

TEST AND BRANCH

- - - - - - - - - - - - - - -

arg"x then – cmd '"# else – cmd '
Conditional which checks arg according to condition x; discards arg; executes then – cmd if condition was true, else – cmd if false.

arg"x then – cmd '
Conditional without else – cmd.

argF"x Same as arg"x, but passes first arg to then – cmd or else – cmd.

Note: The conditions x are:

A	Arg is IN ['a..'z] or ['A..'Z].
C	Arg is printable ASCII code.
D	Arg is IN ['0..'9].
E	Arg = 0.
G	Arg > 0.
L	Arg < 0.
N	Arg # 0.
V	Arg is IN ['a..'z].
W	Arg is IN ['A..'Z].

TAG

- - -

!label! Defines label, or brackets comments.

	ABSOLUTE POSITION
mJ	Position ptr to after m – th char in buffer.
	RELATIVE CHARACTER POSITION
mR	Move ptr left m char; R = > 1R, – mR = > mC.
mC	Move ptr right m char; C = > 1C.
	LINE POSITION
mL	Move ptr to beginning of m – th line after current position; 0L = > beginning of current line.

DELETING

	ABSOLUTE POSITION
m,nK	Kill chars in the range m,n. Move ptr there.
	CHARACTER
nD	Delete n chars to right of ptr.
– nD	Delete n chars to left of ptr.
	LINE
nK	Kill chars from . to position nL would have moved to. K = > 1K, kill to beginning of next line.

TYPE – OUT

kT	Type out text in range k (n lines or m,n chars).
n =	Types out n.

SEARCH

nSstring$	Find n – th occurrence of string searching forward and position ptr after it.
– nSstring$	Same as nSstring$, but search backward.

SEARCH AND REPLACE

nFstring$string2$	Replace n occurrences of string and replace with string1.
– nSstring$	Same as nSstring$, but search backward.
m,nFstring$string2$	Do Search and Replace over range m,n in buffer

INPUT

Yfile$	Reads contexts of file into buffer at right of ptr.

OUTPUT

P$	Output file to input file.
m,nPfile$	Output specified range of buffer to file.
Pfile$	Output to file.

CLOSE

E	Return to Executive

Q – REGISTERS

nUq	Inserts number n in Q – reg q; returns no value.
kXq	Inserts text range k into Q – reg q, replacing prior contents.
Gq	Inserts text (or decimal representation of number) from Q – reg q into buffer.
%q	Increments numeric contents of Q – reg q and returns result.
Qq	Value = numeric value in Q – reg.
Vfile$	Saves contents of all Q – regs in file. To load this file from the Exec: Teco.~ inputfile qRegFile/initialCommand

MACROS

- - Teco.doc

AN ANNOTATED OUTLINE OF TECO COMMANDS

****

From the Executive:

- - - - - - - - - - - - - - - - - -

Teco.~ [inputfile] [macroFile]/[cmd]

where
- inputfile is the name of the file to be edited,
- macrofile is the file which contains the contents of the Q - Regs (using the V command)
- cmd is a initial command string that will be executed

Examples:
- Teco.~ User.cm Teco.macros/mj Load User.cm and execute macro in Q - Reg J
- Teco.~ /c macroFile/[cmd] No inputfile, Q Regs ← macroFile
- Teco.~ No inputfile, no macrofile.

META - NOTATION

= = = = = = = = = = = =

|x	Denotes the single character, control - x.
|	Used alone to denote alternation.
m|n|arg	Integer arguments.
string	String argument.
cmd	A command string.
$	Denotes the character ESCAPE unless mentioned otherwise.
k	Denotes either "m,n", or "n"; a text range of characters m through n, or n lines.
file	Denotes a file name .

SPECIAL CHARACTERS

= = = = = = = = = = = = = = = = = =

ESCAPE	Terminates text argument; two successive altmodes terminate command string.
BACKSPACE	Deletes last character typed in.

ARITHMETIC

- - - - - - - - - - -

0 - 9	Digits: xxxx is interpreted in base (10)
+	Addition, an arithmetic operator.
-	Subtraction, an arithmetic operator.
*	Multiplication, an arithmetic operator(with no operator precedence).
/	Division, an arithmetic operator(with no operator precedence).
!	MOD operator.
(|)	Parentheses, grouping in arithmetic expressions.

ARGUMENT SEPARATOR

- - - - - - - - - - - - - - - - - -

,	Separates numerical arguments.

THE BUFFER

= = = = = = = = = =

.	Value is number of char to left of pointer.
Z	Value is number of char in buffer*.
H	Equivalent to "B,Z", i.e., specifies whole buffer*.
B	Value is 0, i.e., beginning of buffer*.
	*Meaning is modified by use of virtual boundaries.

INSERTING

- - - - - - - - - -

Istring$	Insert string in buffer to left of pointer.
nI	Insert the char with ASCII code n.
m,nI	Insert m copies of the char with code n.

UPPER - CASE/LOWER - MOVING AROUND

- - - - - - - - - - - - -

ABSOLUTE POSITION

- - SwapReasonTune.doc

SwapReasonTune is similar to NewMailTune, except that instead of playing a tune when you get new mail, it plays a tune when you world - swap. Not only that, it plays a DIFFERENT tune depending on the reason for the world - swap. To use SwapReasonTune you must also load Play. You specify the tunes in the "SwapReasonTune" section of your User.cm.

Sample User.cm entries:

```
[System]
InitialCommand: Run.~ Play SwapReasonTune

[SwapReasonTune]
breakpoint: >>>>bbb%%%bbb%%%bbb%%%bbb
interrupt: @50{A>>>A}<<<{A>>>A}<<<{A>>>A}
storageFault: @360,135>CccFFFFFF%%%CcfAAAAAA
uncaughtSignal: **ABG<G>DD
callDebugger: *CCCcfffefa AAAa
```

Full list of possible swap reasons (taken from CPSwapDefs):

- - Generated by debuggee, handled by debugger:

interrupt	- - Runtime.Interrupt called.
breakpoint	
storageFault	- - Better known as address fault.
uncaughtSignal	- - Good ol' Uncle Sig.
callDebugger	- - Runtime.CallDebugger or WorryCallDebugger called.
bug	- - RuntimeInternal.Bug called.
cleanMapLog	
noOp	- - Do nothing; just return to debuggee.
return	- - Return from SwapReason[callDebuggee, resizeBreakBlockTable, resizePatchTable, showScreen, or start].
returnAborted	- - Return from SwapReason[callDebuggee, etc] due to ABORTED signal.
debuggeeSpareA	
debuggeeSpareB	

- - Generated by debugger, handled by debuggee:

callDebuggee	- - Call a procedure.
kill	- - Terminate debuggee boot session.
proceed	- - Continue execution. Resume a SIGNAL.
quit	- - Raise ABORTED.
resizeBreakBlockTable	- - Grow or shrink break block table.
resizePatchTable	- - Grow or shrink patch table.
showScreen	- - Show display screen for a while and return.
start	- - Start or restart a module.
resume	- - Resume a signal (with return values).
debuggerSpareA	
debuggerSpareB	

The two cards slithered towards him across the green sea. Like an octopus under a rock, Le Chiffre watched from the other side of the table.Bond reached out a steady right hand and drew the cards towards him.Would it be the lift of the heart which a nine brings, or an eight brings? He fanned the two cards under the curtain of his hand. The muscles of his jaw rippled as he clenched his teeth.His whole body stiffened in a reflex of self – defense. He had two queens, two red queens. They looked rougishly back at him from the shadows.They were the worst.They were nothing.Zero.Baccarat.

The remote directory:

```
<remoteDirectoryName>
   dotedit!1          202 20 – Aug – 82 10:53:06 PDT
   dotfile.txt!1       586 20 – Aug – 82 10:13:20 PDT
   foo.bar!1          532 29 – Jul – 82 15:31:11 PDT
   fooedit!1           76  9 – Aug – 82 15:25:45 PDT
<remoteDirectoryName>subdir
   foo.mesa!1         532 29 – Jul – 82 15:31:11 PDT
  Total of 5 files
```

The command line:

substitute2 /b open/c remoteServerName dir/c remoteDirectoryName fooedit/t foo.bar/i foo.bar/ – y bogus.file foo.bar/ – b dir/c remoteDirectoryName>subdir foo dir/c remoteDirectoryName dotedit/t dotfile.txt/d close/c junkedit/t junk.file junk.bogus/ – b

The result:

```
Loading table fooedit ... done
 – – unknown switch: i
foo.bar skipped
 – – unknown switch:  – y
foo.bar skipped
[rain]thorup>bogus.file was not found
Editing foo.bar ... done
Editing foo.mesa ... done
Loading table dotedit ... done
Editing dotfile.txt ... done
Loading table junkedit ... done
Editing junk.file ... done
junk.bogus was not found
```

After execution:

File: dot.file

The two cards SLITHERED towards him across the green sea. Like an octopus under a rock, Le Chiffre watched from the other side of the TABLE. BOND reached out a steady right hand and drew the cards towards HIM. WOULD it be the lift of the heart which a nine brings, or an eight brings? He fanned the two cards under the curtain of his hand. The muscles of his jaw rippled as he clenched his TEETH. HIS whole body stiffened in a reflex of self – defense. He had two queens, two red queens. They looked ROUGISHLY back at him from the SHADOWS. THEY were the WORST. THEY were NOTHING ZERO BACCARAT!

The remote directory:

```
<remoteDirectoryName>
   dotedit!1          202 20 – Aug – 82 10:53:06 PDT
   dotfile.txt!1       591 25 – Aug – 82 10:50:17 PDT
   dotfile.txt$!1      586 20 – Aug – 82 10:13:20 PDT
   foo.bar!1          532 29 – Jul – 82 15:31:11 PDT
   foo.bar!2          552 25 – Aug – 82 10:50:00 PDT
   fooedit!1           76  9 – Aug – 82 15:25:45 PDT
<remoteDirectoryName>subdir
   foo.mesa!1         552 25 – Aug – 82 10:50:07 PDT
   foo.mesa$!1        532 29 – Jul – 82 15:31:11 PDT
  Total of 8 files
```

Switches with Substitute2 may either be global or local, depending on the switch and the desired result. Global switches appear alone in the command line and are preceded with a "/". Local switches are also preceded by a "/", but this "/" immediately follows either a Substitute2 command or a filename. The Substitute2 switches are:

c — Causes the token preceding it to be interpreted as a command. Only legal as a local switch.

t — Causes the token preceding it to be interpreted as the name of a table file to be loaded into the substitution hash table. Only legal as a local switch.

b — Changes the backup file option to TRUE. When this option is on, the applicable file(s) will be copied to filename$, and the file containing the substitutions will be written to filename. When given as a global switch, filenames following it will have backup files created before substitution. When used as a local switch, only that file will have a backup created. This switch may be used in conjunction with the d or -d switch. This option's default value is FALSE.

-b — Changes the backup file option to FALSE. The substitutions will be written out to the specified file, and no backup will be created.

d — Changes the dot - in - token option to TRUE. When this option is on, a dot (.), may be part of a the oldstring for which a substitution is being made. For example, when this option is on, Append.String can be a valid oldstring in the substitution table file. This switch may be either local or global, and maybe used in conjunction with the b or -b switch. This option's default value is FALSE.

-d — Changes the dot - in - token option to default value of FALSE. With this option off, a dot (.) is not considered a legal part of the old string for which a substitution is being made, and any oldstring table file entry containing a dot, will be ignored.

EXAMPLE

The following is an example of how to use Substitute2, showing its effects on one of the files being edited and on the remote directory where some of the substitution files are located.

Before execution:

File: dotedit

```
table.Bond              TABLE. BOND
slithered               SLITHERED
rougishly               ROUGISHLY
him.Would               HIM. WOULD
teeth.His               TEETH. HIS
shadows.They            SHADOWS. THEY
worst.They              WORST. THEY
nothing.Zero.Baccarat.          NOTHING ZERO BACCARAT!
```

File: dotfile.txt

- - Substitute2.doc

OVERVIEW

Substitute2 is an updated version of the previously released Substitute hack. This hack registers the command, Substitute2.~ with the Executive and is used to replace strings within a file. Commands are obtained from the command line, and have the following basic syntax, with optional commands appearing in <>:

Substitute2 <open/c hostname directory/c dirname> table1/t File1 File2 File3 <close/c> table2/t File4 File5.

Result:
Substitutions are made in remote File1, 2, and 3 according to the directions in remote table file, table1. Local File4 & 5 are altered according to local table file, table2. If a filename does not contain an extension, and it is not specifying a table file, the extension "mesa" will be added to the filename.

FUNCTIONAL DESCRIPTION

To use Substitute2, you must first make a file which contains a table of identifiers and their desired replacements. The lines in this table file must be in the following format:

oldstring<TAB>newstring<CR>

Due to the parsing of the input, oldstring must be a single identifier token, i.e. it cannot contain any white space. newstring, however, may be any text which does not contain a CR. Please note that the last line in your table file must end in a CR, or that line will be left out of the replacement table, and none of the substitutions pertaining to that line, will be made.

Substitute2 works by reading in the file containing the identifiers and their replacements, and placing them into a hash table. It then takes each file specified in the command line and looks up each of its tokens(chars separated by white space), in the hash table. If the token is found there, then its replacement is written out to the output file. All white space, and tokens which are not found, are written directly to the output file. File lines which are preceded by " - - "(comment lines), or text strings which are enclosed by double quotes ("), are not checked for substitutions, but are simply copied to the output file. Substitute2 will find any files which are on the current searchpath, but when it has finished with the substitutions, it will leave the output file at the top level of the search path. Similarly, when you are working with files across volumes, Substitute2 will find files on the specified volumes(volumes must be open for read/write), but will write the output file to the volume on which Substitute2 is running.

COMMANDS AND SWITCHES

Substitute2 has three commands which are used for specifying remote files for substitution. For proper interpretation, all of these commands must be followed by a "/c".

open — Interprets the next token in the command line as the name of a remote host. Filenames following an open command will be looked for on the host specified. If a connection to another host is open when the open command is given, that connection is closed, and another is opened to the new host.

close — Closes a connection, which was opened to a host via the open command. Filenames which follow a close command, will be looked for on the local file system.

directory — Causes the next token on the command line to be interpreted as the name of a directory. Filenames following a directory command will be looked for in the specified directory. The directory command may be abbreviated to "dir".

-- Strings.doc

This command searches binary files for any printable strings. The default threshold for distinguishing a bona – fide string from accidentally – printable junk is 5 – i.e. if there are 5 printable characters in a row, it's a string. You can specify a different threshold with a switch. Examples:

Strings Xyz.bcd
Strings Xyz.bcd/10

– – SpellChecker.doc

To use, retrieve SpellChecker.bcd and SpellChecker.BitTable. When you run SpellChecker, a toolwindow will appear. In the tools form you have:

File Name: file to be spell checked. If this field is empty, the current selection will be spell checked.

Private Dictionary: is the name of your current private dictionary. The private dictionary is used to hold words that are not in the SpellChecker standard dictionary. You can have more than one private dictionary, to create a new one or to load an old one, put the name of the dictionary file (default is Active.BitTable) in this field and hit ChangeDictionary!

Spell! will start the spell checker.

Add! allow you to add words in the current selection to the dictionary. The added words will be appended to a .dictTxt file. For example, if the current dictionary is Active.BitTable, the text form of the added words will be in Active.dictTxt.

Delete! will delete the words in the current selection from the dictionary. It is advised to use this command with discretion since in may delete more entries than you anticipate.

ChangeDictionary! will cause the file in the Private Dictionary field to be loaded.

Sort will cause the output be be sorted and remove any redundant words.

Strip Plurals is a heuristic to reduce the number of erroneous matches by the spell checker. This will try a word that ends in "s" without the "s". It also tells the spell checker to ignore words with less than two characters.

-- SourceTime.doc

SourceTime is a hack which adds a menu to all loaded FileWindows. Once SourceTime has been loaded, Whenever the "TSave" menu item on the "Source Time" menu is invoked, SourceTime makes sure there is a "standard header" at the top of the file (If there is none, it puts it there. If there is one, it edits it to update it), causes the file to be "Save"d, and then forces the create date of the file to match the date put in the header.

I have found it useful to put the string "TSave" in my sticky menu symbiote, instead of "Save"

The standard header is of the following form: (not including the line of dashes before and after)

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-- File: Blither.mesa -- last edit:
-- userA      20-Aug-81 16:48:57
-- userB      9-Jul-81 18:52:41
-- userC      28-May-81 19:51:28

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The dates appear in chronological order, most recent first. The username is the user's Profile.GetUser, with qualification = registry. Note that any one username never appears on more than one line of this header; SourceTime deletes old lines.

-- SmoothScroll.doc

SmoothScroll adds continuous scrolling to Tajo windows. Simply hold down point or adjust in a scrollbar and the window will begin scrolling continuously, forward or backward. (Actually you have to hold it down for more than 0.4 seconds before it starts.)

Normal scrolling still works as it always has. You have to button – up within 0.4 seconds in order to get normal scrolling, but most people button – up that quick anyway. Thumbing still works too (without the 0.4 second requirement).

If you press down the other mouse button while you are scrolling, you get double speed scrolling. As long as you hold down the other button, you get double speed scrolling. When you let up on the other button, it goes back to single speed. You may speed up and slow down as often as you want.

Installation:

A NEW TIP>Scrollbar.TIP IS REQUIRED FOR THIS HACK.

1. Retrieve Scrollbar.TIP from the diskette and replace the one that's currently on your TIP> subdirectory. (i.e. Delete TIP>Scrollbar.TIP, then retrieve the new one with a Dest:TIP>Scrollbar.TIP).

1a. If you were using a previous version of SmoothScroll, you should also delete TIP>SmoothScroll.TIP. If you don't, you will find yourself unable to stop a scroll once it has started!

2. Retrieve SmoothScroll.bcd.

3. Re – Boot! You must do this before the new Scrollbar.TIP table will take effect.

4. Run SmoothScroll.

I recommend running SmoothScroll from your User.cm InitialCommand line.

-- SModel.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- SmashExportedTypeBit.doc

If you can't load a BCD because it has an exported type clash with something in the boot file or something that was previously loaded, SmashExportedTypeBit can fix the problem.

SmashExportedTypeBit rewrites the header of a BCD so that its "typeExported" flag is FALSE. This allows you to load a BCD for which the loader would otherwise complain of an "exported type clash". The file's create date is NOT changed by SmashExportedTypeBit.

To run it, just type

SmashExportedTypeBit.~ This.bcd That.bcd TheOther.bcd...

to the Executive. You can stop it with the STOP key. You can omit the ".bcd" extension if you want. If it has trouble acquiring any of the files, it will complain "problem acquiring file" and return an error to the Executive without completing the remainder of its command line.

-- SimpleCalc.doc

SimpleCalc is a calculator tool derived from the Pitts Jarvis's calculator.

The calculator provides facilities for performing arithmetic operations and radix conversion on long (32 bit) cardinals and integers. The calculator uses reverse Polish notation, RPN, with a four element stack. The names of the four stack registers are X, Y, Z, and T with X the top of the stack.

The calculator window consists of three subwindows. The top window displays the contents of X. The middle window provides some calculator functions. The bottom window is a scratch pad where text can be stored and edited. The scratch pad is helpful for retaining and labeling intermediate results.

Calculator Type In

When the cursor lies in the top two subwindows all type in goes to the top of the stack. As the user types digits, the calculator accumulates the result in X. Depending upon the radix a digit can be any of the decimal digits or letters a through f (case is not important). If the radix is decimal, numbers can be entered in scientific notation using E ahead of the exponent, e.g., 4e4 yields 40000 and 4.5e-3 yields 0.0045.

Functions that can be type from the key board are as follows:

BS	back space	rubs out the last character
CR, SP	enter	T ← Z; Z ← Y; Y ← X;
DEL	delete	X ← 0;
+, =	addition	X + Y
-	subtraction	X - Y
*	multiplication	X*Y
/	division	X/Y
\	remainder	X mod Y
↑	exponentation	Y**X
↑A	and	X and Y
↑C	cardinal	display X as unsigned number
↑D	decimal	set radix to decimal
↑E	exchange	exhange X and Y
↑I	integer	display X as two's complement number
↑L	list	display stack registers
↑N	negate	X ← two's complement of X
↑O	octal	set radix to octal
↑R	or	X or Y
↑S	hex	set radix to hex
↑T	not	not X
↑X	xor	X xor Y

Any illegal character typed echoes as a flash on the display.

Functions

The middle subwindow provides an alternate method to invoke some of the functions above. These are:

Radix	Octal, decimal, or hex
Type	integer or cardinal
Exhange	exchange X and Y
Negate	change sign of X
NOT	bit wise complement of X
XOR	bit wise exclusive or of X and Y
AND	bit wise and of X and Y
OR	bit wise or inclusive or X and Y
List	displays all four stack registers in the scratch pad

Various scientific functions are also available via commands in this subwindow.

-- ShowSearchPath.doc

ShowSearchPath.bcd creates a small window that contains the current search path. Unlike previous versions of this hack, it no longer shows the free page count; the herald provides this function. The default window box is [x: 0, y: 0, w: 512, h: 30], but this can be overridden by the [ShowSearchPath] section in User.cm.

The hack predates the built in command ShowSearchPath.~, but the presence of this command makes it a little harder to start up this program (you must explicitly say Run).

- - ShowDirectories.doc

ShowDirectories makes MFile.EnumerateDirectory[...directoriesOnly] available at the User level. It registers an Exec command which takes an optional root directory and an optional "T" switch (for "Top – level only" – actually any switch will work in the current implementation). The current search path is ignored. No argument implies the entire current volume. Other volumes may be specified, but will produce the message "** MFile.Error **" if they do not exist or are not open. The directories listed are not sorted. Examples:

```
>ShowDirectories

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<>4.2 <>5.0 <>star <>tools <>star>temp <>star>temp>inner
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

>ShowDirectories/t

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<>directoryName1 <>directoryName2 <>directoryName3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

>ShowDirectories directoryName

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<>directoryName>temp <>directoryName>temp>other
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

>ShowDirectories directoryName/t

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<>directoryName>temp
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

-- SetTool.doc.

Environments: CoPilot|Tajo.
Documentation: SetTool.doc.
Description: A set manipulation Tool. Takes as input sorted files of elements that are separated by carriage returns. Operates (Union, Intersection, Copy, Difference and XOR) on Operand1 and Operand2 (string parameters that usually contain file names) and places the output in Result (a string parameter that usually contains a file name). If one of the input parameters is the same as the result parameter then the result is buffered and then copied to the result. If any of the strings "1", "2" or "R" are used then the corresponding string parameters in the bottom subwindow are operated upon. The Tool may be driven from the Tool Driver (window = "SetTool", subwindows are from top to bottom: "MsgSW", "CmdSW" and "DataSW") and is not run in the background.

- - SetsToPackSpec.doc.

Environments: CoPilot|Tajo SimpleExec.
Documentation: SetsToPackSpec.doc.
Description: SetsToPackSpec takes an ordered list of input files, each of which is a set of procedures created by tools like TopoGiggio, SetTool, or ReductionTool. It creates a packaging description for the modules described by the input files. It formats each set into one CODE PACK, listing all of the procedures in that set except for any that were named in previous sets. It also generates merged or separate "Orphan" CODE PACKs (see the "m" switch, below). This program produces one SEGMENT and one FRAME PACK. The program has an entry vector option (see the "e" switch) and an option to create output sets (see the "s" switch).

SetsToPackSpec runs in the simple exec. The command line has the format
SetsToPackSpec Output.pack/switches [prefix/p] Set1.raw Set2.raw Set3.raw ...

If there is no input on the command line, the tool will prompt the user.

The segment name in the packaging description will be the output filename sans extension. The code pack names will be the input filenames sans extension, with an optional prefix (see the "p" switch, below).

Options are controlled by the following switches:

/e = > Global switch: Add "ENTRY VECTOR" automatically to the first code pack that each module appears in. Defaults to TRUE.
This switch can only be specified on the output filename or the first input filename.
NOTE: "Automatic" ENTRY VECTORs will interfere with any that are specified in the input files. I recommend that you turn this feature off and put the entry vectors in the input sets with TopoGiggio's "EV" option.
"Automatic" ENTRY VECTORs are never written to output set files (see "/s", below).
Example:
SetsToPackSpec Output.pack/ - e Set1.raw Set2.raw Set3.raw ...

/m = > Global switch: Merge the orphan code packs into one named "Orphan". This switch defaults to FALSE, meaning to create a separate "*Orphan" code pack for each module.
The last specification of /m or / - m is what counts.
BEWARE of merging ALL the orphan code into one pack. According to the semiautomatic packaging philosophy, orphan packs are supposed to contain infrequently referenced stuff as well as never referenced stuff. If you make these packs too big, when an infrequently referenced procedure is touched, you pay a (potentially) large swapping penalty.

prefix/p = > Prepend "prefix" to the input file names to create the code pack names.
The prefix specification will hold until a new prefix is specified.
" /p" will turn off any prefix that has been set.
NOTE: A "prefix/p" specification must come AFTER the output filename.
Example:
SetsToPackSpec Output.pack/ - e XConfig/p Set1.raw Set2.raw Set3.raw ...

/s = > Generate "finalized" output sets corresponding to the "raw" input sets.
For each input file (e.g. "Repaint.raw"), this switch will create a finalized output file set with the extension ".set" (in this case "Repaint.set").
For any input file with the extension ".set", this won't create an output set.
Example:
SetsToPackSpec Output.pack/s Set1.set Set2.raw
will create only one output set "Set2.set".
The "/s" option can be turned on and off.
Example:
SetsToPackSpec Output.pack Set1.raw Set2.raw/s Set3.raw Set4.raw/ - s Set5.raw
will create two output sets "Set2.set" and "Set3.set".
"Automatic" ENTRY VECTORs are never written to output set files (see "/e", above).

- - SendInterpress.doc

SendInterpress will send interpress masters to a product print service. The files may either be remote or local. Remote files are transferred to a temporary local file before they are transferred to the printer.

Command format:

SendInterpress.~ ... [host/h][dir/d][printer/p][/c<number>] filenames

The following switches apply (case of switches is ignored):

host/h	host is the default host for fetching files
dir/d	dir is the default directory for fetching files
printer/p	printer is where the interpress masters are sent
/c<number>	print <number> copies of the file. file

Examples:

SendInterpress Foo.ip/c2

SendInterpress ""ServerName:Domain""/h PrinterName/p ""DirectoryName""/d Foo.ip

-- SearchPath.doc

SearchPath.bcd creates a menu that has two lists of directories, separated by a pair of commands:

<first search path directory>
...
<last search path directory>
↑ (rotate)
←(directory)
<root directory>
...
<last subdirectory>

Pointing at any name in the search path list deletes that directory from the search path.

Pointing at any name in the directory list adds that directory to the search path (at the top).

Pointing at " ↑ " causes the search path to rotate once (the top becomes the bottom, all others shift up one).

Pointing at "←" causes a small window to appear that allows you to create and delete directories.

-- Scroller.doc

Scroller.bcd makes use of the Dandelion smooth scroll microcode to smooth scroll portions of the screen. After running Scroller, a tool window will appear. If you move the cursor over the Scroller's window and hit Point, this cause the Scroller window to disappear. If you hit Point again, the bits that the Scroller window used to cover will smooth scroll. Initially the bits scroll a rate of 2 scan lines per screen refresh. To increase the rate, hit Point and that will cause the rate to increase by 2 scan lines per refresh. You can decrease the scroll rate by 2 scan lines per refresh by hitting Adjust. To stop scrolling, hit [STOP] once, and the get the screen back to its original state, hit [STOP] again.

-- ScavengeVolume.doc

ScavengeVolume registers the command ScavengeVolume..~ with the executive. With this command, you can invoke the Pilot and XDE scavengers.

The command is interactive, and should be reasonably self – explanatory. Type "y" ora carriage return to answer "yes" to a question, and type "n" to answer "no". Pressing DEL (the delete key) in response to a question aborts the command.

-- ScavengerTool.doc

The ScavengerTool allows a user to scavenge a volume that has a Mesa file system on it but does not have a Tajo boot file. Type ScavengerTool into the Executive to invoke a window interface.

In the ScavengerTool's formSW is Volume: , the volume to be scavenged, Repair and Verbose, booleans that cause ScavengerTool to repair volumes and give a verbose output. Scavenge! starts the scavenger, and FreePages! tell the number of pages free on the scavenged volume.

Do not try to scavenge a volume of higher order than the current system volume.(ie dont try to scavenge CoCoPilot from Tajo or CoPilot) It is also probably wise to run ScavengerTool after Pilot has had a chance to scavenge the volume using Othello.

The scavenger log is placed on the scavenged volume in MScavenger.log.

- and , and . are - ,. with COMMAND held down

Programable Keys:
PF1 - PF4 are MENU, SCROLLBAR, JFIRST, and JSELECT

If you are a TIP wizard, you can place all these functions wherever you please by modifying <>TIP>Emulator.TIP.

- - RS232XChat.doc

RS232XChat allows you to chat through the Dandelions RS232C port. Just set the parameters on the tool to match the configuration of the thing that is connected to the port, and hit Connect!. To quit, hit Disconnect!
The following is standard fare for the Emulator subwindow:

1) The enumerated item Terminal: in the form subwindow has a pop up menu with the various terminals that can be emulated. The enumerated items represent the following terminals:

addrinfo	General Terminal
adm3	Lear Siegler Adm3
adm3a	Lear Siegler Adm3A
cdc456	Control Data 456
dm1520	Data Median 1520, 1521
gt100	General Terminal 100A
h1000	Hazeltine 1000
h1420	Hazeltine 1420
h1500	Hazeltine 1500
h1510	Hazeltine 1510
h1520	Hazeltine 1520
h2000	Hazeltine 2000
isc8001	Interactive Systems
soroc	Soroc 120
teletec	Teletec Datascreen
trs80	Radio Shack
vc303	Volker - Craig 303
vt100	DEC VT100
vt50	DEC VT50
vt50h	DEC VT50H
vt52	DEC VT52
x820	Xerox 820

2) The enumerated item Refresh: in the form subwindow has a pop up menu with the following items:

always	update screen on every character
never	update only if nothing else is happening
half	force an update when the screen is half invalid
full	force an update when the screen is all invalid

Using the never option, you can get tranfer rates of over 9600 baud, but the display will only be updated when data is not coming over the line. This is useful for transferring files from your host computer to your dandelion since the entire transcript is stored in Chat2.log.

3) The command TerminalOptions! will bring up a property sheet where you can set different terminal options. Using the mouse, you can set up the tabs on the bottom subwindow of the options sheet. There are various enumerated which you may or may not be able to set.

4) At the bottom of the emulator subwindow are some bells and whistles. The DATA one is a set of flippers that are inverted every time some data is sent to the emulator subwindow. The ONLINE and LOCAL buttons tell you if you have a connections. The L1, L2, L3, and L4 buttons are settable by the host in the VT100 mode.

5) The emulator subwindow is not a standard Tajo TextSW or TTYSW. Selections can be made using Point and Select to define the boundaries of the selection. There is no selection tracking as in regular text subwindows, and the selection disappears once new text is written to the screen. Selection can be stuffed into other windows using the Stuff button, and text from other windows can be stuffed into the emulator subwindow. There are no scrollbars on the emulator subwindow, to see the full context of the window one must grow the window to be large enough. Hitting Adjust in the emulator subwindow will cause the window to become the input focus if it does not already contain a selection. A log is kept in the file Chat2.log.

6) Special Keys of note (refer to the XDE User's Guide for a description of the key names):
The CNTL key is CONTROL
The ESC key is COMPLETE
The DEL key is DELETE
Cursor Keys:
Up, Down, Left, and Right are HELP, DOIT, NEXT, and UNDO

If you are in the vt100 mode, there are several KeyPad and Programmable Functions Keys available to you. With the built in Emulator.TIP file, you have the following:
KeyPad Char:
0 - 9 are 0 - 9 WITH COMMAND held down
Enter is COMMAND Return

- - Rotate.doc

This program registers an exec command "Rotate.~" that rotates the letters in its argument by the number specified by the switch (mod 26). The default rotation is 13. It ignores non letters, and it preserves case. If the argument is "$$$" it uses the current selection as the text to rotate. It is useful for un - rotating usenet "non - spoilers".

The text of this file rotated 13 is

" - - Ebgngr.qbp

Guvf cebtenz ertvfgref na rkrp pbzznaq "Ebgngr.~" gung ebgngrf gur yrggref va vgf nethzrag ol gur ah zore fcrpvsvrq ol gur fjvgpu (zbq 26). Gur qrsnhyg ebgngvba vf 13. Vg vtaberf aba yrggref, naq vg cerfreirf pnfr. Vs gur nethzrag vf "$$$" vg hfrf gur pheerag fryrpgvba nf gur grkg gb ebgngr. Vg v f hfrshy sbe ha - ebgngvat hfrarg "aba - fcbvyref"."

-- RootPicture.doc

This hack runs in Tajo and lets you replace the "officetalk gray" background of the root window with a background of your choosing. The background is a "vanilla" bitmap picture generated by Markup which begins with eight words of data – values and is then followed by the bitmap. RootPicture uses this kind of press file.

Invoke the hack by getting the bcd and typing "RootPicture filename/switches" to the Executive or via user.cm/InitialCommand: "RootPicture filename/switches" startup. If no filename is mentioned, it is defaulted to "RootPicture". If the filename contains no extension, ".press" is appended.

Command line switches let you control where and how the picture is displayed.

Switch /d [for 'duplex']: RootPicture attempts to place two press files on the screen, one on the left half and one on the right. The first file is "filenameL" and the second is "filenameR"; they are flushed bottom and toward the center. This is a useful switch to use to get a full – screen picture. e.g., You will have to rename "TiffanyLeft.press" = > "Tiffany.pressL".

Switch /c [for 'center']: RootPicture attempts to place one press file on the screen. The name is "filename", and it is centered and flush bottom.

Neither /d nor /c: Like /c but the picture is flush left and bottom.

Switch /i [for 'invert']: If your terminal background is inverted (black), the press files' bits will be automatically complemented so that you do not get a negative image. If your background is white, then this switch doesn't do anything.

Switch /p [for 'paint']: the picture is painted (ORed) into the display after painting in an officetalk gray. This has the effect of making white bits in the bitmap transparent and letting the normal gray background shine through. If this switch is not specified, the default mode, replace, will be used.

Switch /g [for 'gray'] followed by 0 to 16 octal numbers separated by whitespace: the background gray will be that specified by the octal numbers instead of officetalk. The octal numbers specify a Display.Brick that is to be painted before the picture. If no numbers are specified, the gray is reset to officetalk.

If /d and one of the "filenameL" and "filenameR" files is absent, then only half the screen gets a picture.

If /d and both "filenameL" and "filenameR" files are absent, then the /d is ignored. [and a single "filename" file is centered or flushed left]

If the file "filename" is absent (or if all of "filenameL", "filenameR" and "filename" are absent if /d), then RootPicture puts the officetalk gray back.

If RootPicture is run from the "InitialCommand" line in your user.cm, it sets the switches to /dc.

Examples:
"RootPicture Tiffany/c" centers Tiffany.press at the bottom.
"RootPicture /d" builds a wide picture from RootPicture.pressL and RootPicture.pressR.

-- RootDirectoryTool.doc

The Root Directory Tool provides user – level access to the Volume.*RootFile procedures. It is a full – fledged tool with one window consisting of three subwindows: a message subwindow for posting error messages, a form subwindow, and a log subwindow.

The form subwindow logically consists of two sections: a volume section and a root directory section. It looks something like this:

```
Volume: {Tajo}
Type: {normal}      Status: {openReadWrite}  readOnly
Open! Close!

FileID: [0B, 125000B]
Type = 9344
LookUp! Insert! Remove! GetNext!
```

"Volume" is an enumerated item that consists of all currently on – line logical volumes. Changing it will also change the "Type" and "Status" items to the appropriate values. The last two items are read – only.

"Open!" and "Close!" will invoke the appropriate operations on the current volume. "readOnly" is a boolean that affects how the Open is done. Note that when a volume is opened with ~readOnly, all temporary files on that volume will be deleted.

WARNING: Due to a bug in Pilot, DO NOT open a volume for readWrite if it has a higher type than the one you are running on (normal < debugger < debuggerDebugger.) This is a bad idea in general, but in this particular case, a monitor inside of Pilot will be locked.

In the root directory section there are 4 commands: LookUp, Insert, Remove, and GetNext, that correspond to the appropriate procedures in the Volume interface. They take their arguments from the Volume, FileID, and Type fields, and update them with the results of the operation. I.e.,

Command	Arguments	Sets
LookUp!	Type, Volume	FileID
Insert!	Type, FileID, Volume	- - - -
Remove!	Type, Volume	- - - -
GetNext!	Type, Volume	Type, FileID

Warnings will be given if the file does not exist, or the actual file type differs from the value in the "Type" field.

-- RLMTool.doc

Please refer to the document, "Remote Line Monitor Tool", in XDE Unsupported Software Description.

– – Wall.doc

The Wall Game is a game for 1 to 4 players (the program can provide up to 3 of the players), written for the Tajo environment. It's related to the lightcycle race in TRON – – they're both ripoffs of a 1975 – vintage arcade game called Barricade.

In The Wall Game, each player controls a linear wall that grows by constantly adding bricks at an accelerating pace. If the growing end of a wall crashes into anything, the whole wall vanishes. The object of the game is to steer your wall so as to out – survive the other players, usually by walling them off into small areas of the screen.

The game has a variety of parameters and options, which can vary the relative importance of intellectual strategy, physical coordination, and luck. The program contains on – line help & training, which will be sufficient documentation. The text layout will look best with Gacha12.strike as system font.

– – WindowBoxTool.doc

WindowBoxTool displays the "bitmap box" (position & dimensions) of any or all not – inactive tools windows in your development environment. You can use it after you've positioned your windows the way you like them, in order to copy the data into your user.cm for those tools that check user.cm for initial window size and position.

SelectedWindow! looks for the window containing the selection, and prints something like:
Selected window's bitmap box:
WindowBox: [x: 500, y: 640, w: 480, h: 115]

AllWindows! shows every window's box; e.g.:
Bitmap Boxes for all windows:
    WindowBox: [x: 500, y: 640, w: 480, h: 115]
    WindowBox: [x: 40, y: 40, w: 400, h: 250]
    WindowBox: [x: 40, y: 40, w: 400, h: 250]
    WindowBox: [x: 0, y: 32, w: 500, h: 70]
    WindowBox: [x: 25, y: 336, w: 500, h: 300]
    WindowBox: [x: 484, y: 61, w: 480, h: 350]
    WindowBox: [x: 0, y: 0, w: 1024, h: 30]
7 tool windows found.

AllWindows! inverts each window for about 1 second as it is printing its log.

-- VolumeVersion.doc

If you are responsible for any program that does a Volume.Erase, you may be interested in VolumeVersion. If you are not, read no further.

VolumeVersion is useful in implementing a user interface which will prevent the user from accidentally erasing a volume when that erasing would change the format of the volume. VolumeVersion is typically used in volume initializer programs such as Othello.

VolumeVersion consists of an interface VolumeVersion.bcd and its implementation VolumeVersionImpl.bcd. The interface contains one operation:

```
Examine: PROCEDURE [volume: Volume.ID] RETURNS [result: VolumeVersion.Result];

VolumeVersion.Result: TYPE = {
  currentVersion, badRootPageLabel, ioError, trashedRootPage,
  otherVersion, volumeUnknown};
```

which can be used to tell if a logical volume is compatible with the version of Pilot that is currently running.

- - Wait.doc.

The ExecProc Wait.~ returns to the Executive after desired time. Will abort within one second if the executive is aborted. Time is either 1) number of seconds, 2) hour:min:sec, or 3) dd - mmm - yy hh:mm:ss zzz. If time is in format 1, the switch 'm will treat number as minutes, and the switch 'h will treat number as hours.

Examples:

Wait.~ 5; - - waits five seconds
Wait.~ 4/m - - waits 4 minutes
Wait.~ 2/h - - waits 2 hours.

Wait.~ 8:20:15 - - waits until 8:20:15 in this or next morning
Wait.~ 13:30 - - waits until 1:30:00 in this or next afternoon

Wait.~ "15 - Apr - 83 23:59:59" - - waits until too late to file next year's taxes

Note that if a date is specified, it must be in quotes.

-- UserTip.doc

UserTip is a package that allows you do two thigs: specify keys to be window or input focus events in your User.cm, and have private TIP tables created and inserted into the system chains at initialization time.

All of this is done through the [TIP] section of your User.cm. The following entries are recognised:

ActionToWindow: <Key> -- forces <Key> to be a window event.
ActionToFocus: <Key> -- forces <Key> to be an input focus event.

<globalTable>: <FileName> -- tries to create the TIP table called <FileName>. If successful, it then does a TIP.PushGlobal of the resulting table onto <globalTable>.

<Key> must be a Keys.KeyName (case is significant!)
Currently recognized golbal tables are {root, formSW, textSW, fileWindow, ttySW, executive, spare1, spare2}

Example:

[TIP]
ActionToWindow: SCROLLBAR
textSW: ExtendEdit.TIP

This entry causes the SCROLLBAR key to be a window event, and adds ExtendEdit.TIP to the user actions that ALL text subwindows understand.

-- [illegible]DF.doc

Please refer to the document, "DF Software Reference Manual", in XDE Unsupported Software Description.

-- Type.doc

Type.bcd replaces the Type.~ command that comes built into the Exec. The new Type.~ command is just like the Exec's, except (1) it allows remote file names, and (2) it lets you specify a range of characters within the file(s) to be typed. This is particularly useful for looking at files via a RemoteExec, especially in conjunction with the Find.~ command for determining starting and ending character positions.

Syntax: "Type <list of files to type>". Follow a file name with /FxxxTyyy to type only character positions from xxx through yyy.

-- Unique.doc

Unique

Environments: CoPilot | Tajo

Description: Unique is a program for parsing tokens from files, the current selection, or the Executive's command line. It sorts it's output alphabetically with duplicates removed and outputs it to the Executive. It parses filenames, words, numbers, and lines. Unique registers itself with the Exec. Command line syntax is:
(filename)/{AppendToWords ""suffix "", Extensions, Filenames, Lines, Numbers, Words} (token list).

filename is the optional name of the input file.

A switch to signify the function to perform follows the optional file name and can be one of the following:

a -- Append. /a must be followed by a token which is to be appended to each word in the input source. Useful for alphabetizing lists of words with a comma or some other extension appended to each one.

e -- Extentions. Sorts filenames by extension. Useful for cleaning up directories and such.
Example: Type TAB to the Executive. Select the result and invoke "Unique.~ /e".

f -- Filenames. Extracts tokens of the form name.prefix.
Example: Select some newly written code and use the /f switch to extract all the qualified names. This can be very useful for preparing USING clauses, especially in combination with the "/a " switch.

l -- Lines. Sorts lines uniquely. Can be used to help combine CODE PACKs in Packaging descriptions, or to sort directory statements.

n -- Numbers. Strips out anything but sequences of digits. Unfortunately, it sorts alphabetically, but it can be useful. Can be used to extract interesting groups of AR numbers from sorted AR reports.

w -- Words. Uniquely sorts all tokens encountered.

The optional token list is present if the user wishes input to come from the Executive's command line. In this case, all tokens following the switch will be used as input to Unique. If a filename is present however, any tokens following the switch will be ignored.

– – TurboFile.doc

TurboFile is an extra – cost performance option for the Pilot file system. It speeds up Pilot's file allocation operations (File.Create, File.Delete, and File.SetSize) at the cost of making it more likely that you will have to scavenge when your system crashes. Your files are not jeopardized by TurboFile.

Background and Motivation

Pilot maintains performance accelerator data about the contents of logical volumes, and this data is stored. If the system crashes and this accelerator data on the volume is not correct, the volume will be automatically scavenged the next time is opened in order to reconstruct the accelerator data from the files themselves.

In normal operation, Pilot makes its accelerator data correct and consistent at the end of every file allocation operation. This reduces to a minimum the period during which a system crash will require the volume to be scavenged. This is appropriate behavior if the probability of crashing is high AND the time to scavenge a volume is high. If that is not the case (that is, if the probability of crashing is low or the time to scavenge a volume is low) it is INappropriate, and TurboFile may profitably be used.

To aid you in deciding whether or not to use TurboFile, here are some examples of current scavenging performance from a workstation:

volume	vol size	pages used	Pilot Scavenge	MScavenge
Tajo	5,000	4,500	30 sec	20 sec
CoPilot	40,000(!)	30,000	6 min	5 min

In a simple test of deleting thirty small files, TurboFile reduced the normal elapsed time from 40 seconds to 30 seconds.

Functionality

TurboFile speeds up file system operations by telling Pilot to delay its expensive consistency actions. At user – settable intervals, TurboFile tells Pilot to make its accelerators consistent. This is done so that many, perhaps most, crashes will not require the volume to be scavenged. The default interval is five minutes. At present, TurboFile only acts on the system volume – – other volumes are managed in their normal fashion.

Operation

To get TurboFile running, just type its name to the Executive; it then runs in its own tool window. When TurboFile is started, it registers a command of the same name with the Exec. Executing the command when the tool is inactive will put it in the active state; otherwise the command has no effect.

TurboFile's form subwindow contains three items:

{running, stopped} Force Consistency! Interval (sec) = 300

When "running" is chosen, TurboFile performs its speedup functions, and updates Pilot's accelerators at the specified interval. When TurboFile is "stopped", TurboFile does nothing and Pilot acts in its standard fashion.

"Force Consistency!" causes Pilot's accelerators to be written to the disk immediately.

"Interval" is the number of seconds TurboFile waits between accelerator data updates. Normal usage is to start the tool and then just leave it alone.

TurboFile performs its functions in either the normal or tiny window state. TurboFile updates Pilot's accelerator data (A) at the requested intervals, (B) when it is stopped, deactivated, or unloaded, and (C) before world swaps.

Miscellaneous

If the system crashes while the tool is running, one MAY be able to avoid doing a scavenge by interpret calling the procedure TurnOffWatchDog in TurboFile:

```
>Set Module Context: TurboFile
> TurnOffWatchDog[]
```

TurboFile honors the standard User.cm entries. For example:

```
[TurboFile]
InitialState: tiny
WindowBox: [...]
TinyPlace: [...]
IntervalSeconds: 300
```

- - Triton.doc

Triton provides a simple facility for enumerating files according to various filters, displaying them sorted according to various keys, then deleting a specified subset of them. It provides some of the same features as Neptune or Posiedon.

User Interface:

Triton consists of three windows: The Message SW, used to report progress and problems. The Form SW, used to specify filters, sort keys, commands, and provide credentials. The Scratch SW used to display the file list, and mark files for deletion.

The filters are: Name, Type, and Size. Name is a fully qualified path name, and can specify local, remote NS, remote IFS, or remote Tenex (Maxc) files. Type can be any type known to FileTransfer, including text, binary, directory, or null. Size can be in bytes or pages and can be a single relation or a range.

Files can be sorted on name, extension, size, create date, write date, read date, or type, either ascending or descending.

After all filters and the sort key are specified, bugging Show! will cause Triton to enumerate, filter, sort, and display the files in the Scratch SW. After the files have been displayed, moving the cursor over the line containing a file you want deleted, and bugging Point will mark the file for deletion. Dragging the mouse with Point down will mark a series of files for deletion. Bugging Adjust over a file un – marks it, and similarly dragging with Adjust down un – marks a series of files. Chording, hitting Menu (the middle mouse button), or MENU (the key), while over a file name will cause info about that file to be displayed in the Message window.

After you are satisfied with the files you have marked, bugging Do!t! will delete the marked files and then redisplay the remaining files.

Changing the sort key or direction and bugging Show! will resort the files according to the new specification, without refiltering, or losing marks.

Changing any of the filters and bugging Show! will cause Triton to re – enumerate, filter, and sort the files, discarding old marks.

Triton requires approximately 40 free disk pages to start up (for heaps), and requires about 1 disk page per 5 files enumerated (so you start with enough room for about 200 files). Enumerating more than a few hundred files is PAINFULLY SLOW.

```
-- TraceImpl.doc

-- Copyright (C) 1984 by Xerox Corporation. All rights reserved.
```

TraceImpl is a module which impelements data tracing. If you want to take a trip to the debugger when all or some of the bits in a specific word are either different from, or equal to a specified value, read on.

To use TraceImpl, load it into your program, go to the debugger and call StartTrace[loc, val, mask, equal]. TraceImpl will examine the location on each control transfer. If val = Inline.BITAND[loc↑, mask], then if equal is TRUE, the debugger will be called. TraceImpl will not pinpoint the statement that altered the memory location. It will merely interrupt shortly after some process has altered the location. To stop tracing, call StopTrace. If you simply want to watch a location and get called when the current value changes, you may call Watch[loc]. This is equivalent to StartTrace[loc, loc↑, 177777B, FALSE].

```
StartTrace: PUBLIC PROC [
    loc: LONG POINTER, val: UNSPECIFIED, mask: WORD, equal: BOOLEAN];
-- Does: IF equal = (val = Inline.BITAND[loc↑, mask]) THEN call the debugger;

StopTrace: PUBLIC PROC = {continueTracing ← FALSE};
-- Does: Stop tracing.

Watch: PUBLIC PROC [loc: LONG POINTER] = {StartTrace[loc, loc↑, 177777B, FALSE]};
```

– – Topper.doc

"Topper" eliminates the hassle of fishing around for the WindowMgr menu in order to bring windows to the top or bottom of your Tajo desktop. It adds three functions to key combinations specified in the TIP table "Topper.TIP":

Top: Bring the window containing the cursor to the top. (This is equivalent to the Window Mgr menu's "Top" command.)

Bottom: Put the window containing the cursor on the bottom. (This is equivalent to the Window Mgr menu's "Bottom" command.)

Switch: If the window containing the cursor is obscured by another, bring it to the top, else put it on the bottom. (This is equivalent to clicking RED in the left or right section of the name frame.)

Repeatedly invoking "Switch" will quickly cycle through all the windows which overlap a particular point on the screen.

If <>TIP>Topper.TIP does not exist when you run Topper, the following key assignments will be made:

Top: RESERVED (the key marked "SUPER – SCRIPT" on the Star keyboard)
Switch: Shift – RESERVED
Bottom: Control – RESERVED

If you want to change Topper's key assignments, edit <>TIP>Topper.TIP and reboot. This is not especially recommended for frequent use, since the old TIP table remains on the global TIP chain, and any key transitions not overridden by tne new TIP table will continue to trigger the old table.

Further notes on the use of Tom

One of the major features of Tom is its flexibility. The variables that it displays need not exist as such in any "...Perf" interface. The Timer interface is one such: the updateVariablesProc simply updates a variable to be the current time. A different use of the updateVariablesProc would be to apply arbitrary transformations to variables from ...Perf interfaces. For example, one might choose to take the logarithm of some variable or to combine some variables from different interfaces. These calculations may well be undesirable at the low level that the raw data is generated; putting them in the updateVariablesProc means that they are calculated as infrequently as possible, thus minimising the overhead.
Variables could even be provided to provide synchronising pulses for displays. If one counter were incremented at the beginning of an operation and another at the end then the endpoints of the operation will be immediately obvious on the histogram display. An excellent example would be the number of the current pass of the compiler, were one to monitor that.

To register a new display facility, clients should use routines and definitions available in the TomD interface.

```
Register:
  PROCEDURE
    [name:LONG STRING,              - - the name of the display routine
     setDisplayProc:SetDisplayProcType,    - - a routine to start and stop displays
     updateDisplayProc:UpdateDisplayProcType - - a routine to update the display
    ];
```

Clients should call Register to make a new display facility available to Tom. They must supply the name of the interface, a routine to start and stop displays and a routine to update displays. The name is not copied and should remain in existence for as long as the display is registered.

```
SetDisplayProcType:
  TYPE = PROCEDURE
   [clientHandle:LONG POINTER,          - - handle to display to be stopped
    msgSW: Window.Handle,               - - the tool's msgSW
    parent,                             - - the main Tool window
    insertBeforeThis: Window.Handle,    - - the last subwindow of the Tool
    items:Items]                        - - an array of info for new display
    RETURNS [newClientHandle:LONG POINTER];   - - client representation of new display
```

setDisplayProc's are responsible for both starting and stopping displays. This is so that if only the list of items to be displayed is changing (and not the display facility itself) then a clever setDisplayProc need not completely dismantle its display apparatus having stopped one display, prior to starting the next.
The first time a setDisplayProc is called, clientHandle will be NIL and items will be an array of Items (which contain the variables' names, addresses etc.). The client should set up the display and return a handle to any internal data structures that may have been set up. The client should not use global data to hold the data structures because the display may be multiply active from different instances of Tom. The handle will be passed to the client each time the display is to be updated and when the display is to be stopped. Various window handles are passed to the setDisplayProc. The msgSW allows messages to be posted in the standard Tom message subwindow; other subwindows required by the client are best made by ToolWindow.CreateSubwindow and Tool.AddThisSW using the parent and insertBeforeThis parameters in the obvious way.
(Note: The latter is necessary because of internal nasties in the space allocation between subwindows of a window. insertBeforeThis is a handle to the bottom subwindow of the tool, which, without user intervention will extend from Tom's text subwindow to the bottom of Tom's window ensuring that any subwindows added without quoting insertBeforeThis to Tool.AddThisSW will be out of sight, below the windows bottom edge!)
Subsequent calls of setDisplayProc will be passed the clientHandle result from the previous call so that the client can stop that display and recover the resources used. When the display facility is no longer required, the setDisplayProc will be called one last time with the latest clientHandle and with items = NIL. This is to give the routine a chance to tidy up and return all resources used.
Note also that the array of items passed to the setDisplayProc is defined to remain in existence until after the next call of setDisplayProc, so there is no need for the client to take a copy of it. In fact, there are even client data words inside the elements of the array so that further client specific information may be stored there.

```
UpdateDisplayProcType: TYPE = PROCEDURE [clientHandle:LONG POINTER];
```

Clients must supply an updateDisplayProc to Register. This routine will be called at the priority and frequency specified in the main Tom form subwindow to allow the client to update the display. The handle passed to it is the handle returned from the setDisplayProc.

```
Items: TYPE = LONG DESCRIPTOR FOR ARRAY OF ItemP;
ItemP: TYPE = LONG POINTER TO Item;
Item: TYPE = RECORD
  [interfaceName,                       - - the interface to which this variable belongs
   varName: LONG STRING,                - - the name of the variable!
   varAddr:LONG POINTER TO LONG CARDINAL,          - - the address of the variable
   counter:WordBoolean,     - - the hint from the interface
   client:LONG POINTER ← NIL];

WordBoolean: TYPE = MACHINE DEPENDENT RECORD[zeroes:[0..77777B], b:BOOLEAN];
```

Items are passed to the setDisplayProc to specify the variables to be displayed. As far as possible, the information comes directly from the registered interface. The counter is a WordBoolean for sordid implementation reasons but is otherwise the value returned from a call of the getVariablesProc passed to TomP.Register; i.e. it is an indication that the variable is unbounded monotonic and so it may be better to display first differences rather than the absolute value.

Loading interface and display stubs

Tom.bcd is bound with the Timer interface routine and the Graph and Text display routines. In addition, the user can specify in User.cm that other (user – supplied) interfaces or displays should be loaded when Tom is started. Furthermore, if a name is typed into the Interface: or Display: fields of the form subwindow and the appropriate routine is not found, then Tom will try to dynamically load the routine. It does this by appending either "TomP.bcd" to an interface name or "TomD.bcd" to a display name to get a filename to be loaded. An error message will be given if the file is not found or cannot be loaded.

Registering new interfaces

To register a new interface, clients should use routines and definitions available in the TomP interface.

```
Register:
  PROCEDURE
    [name:LONG STRING,                  -- the name of the interface
     nVariables:CARDINAL,               -- the number of variables
     getVariableProc:GetVariableProcType,    -- a routine to access the variables
     updateVariablesProc:UpdateVariablesProcType] -- a routine to update the variables
    RETURNS [ok:BOOLEAN];
```

Clients should call Register to make a new interface available to Tom (or TomStub). They must supply the name of the interface, the number of variables being made available and two routines. The first routine supplies information about each variable, and the other (possibly NULL) ensures that all the variable are up to date. The name will be not be copied by Register and must remain extant for as long as the interface is registered. Ownership of the string remains with the client. The routine will return FALSE if an interface of the the same name is already registered.

```
Deregister:
  PROCEDURE [name:LONG STRING] RETURNS [ok:BOOLEAN];
```

Clients should call Deregister if they wish to make an interface unavailable for use. This may be because the code is being unloaded or because the client has finished generating the values for the interface for some reason. The routine will return FALSE if no interface of the given name can be found.

```
GetVariableProcType: TYPE = PROCEDURE [index:CARDINAL] RETURNS [variable:Variable];
```

The client must supply a routine of this type in the call to Register. It will be called by Tom to find information about the individual variables; this information is in "variable".

```
UpdateVariablesProcType: TYPE = PROCEDURE[];
```

Clients must either supply NIL or a routine of this type to Register. It will always be called just prior to accessing any address made available via the getVariableProc. The client may use this routine to ensure that all the variables are up – to – date – this is particularly useful if these. values are derived from others.

```
Variable: TYPE =
  RECORD [name:LONG STRING,                  -- the name of the variable
          addr:LONG POINTER TO LONG CARDINAL,  -- the address of the variable
          counter:BOOLEAN ← TRUE             -- a hint to display routines
         ];
```

RECORDs of type Variable are used to return information about individual variables to Tom. Tom only understands LONG CARDINALs; if the client wishes to display other types, they must coerced by the client to a LONG CARDINAL, probably with the help of the updateVariablesProc. 'counter' is an indication (for the benefit of display routines) that the variable is unbounded monotonic and so it is probably better to display first differences rather than the absolute value of the variable. All LONG STRINGs returned to Tom in Variable records remain the property of the client and must remain in existence for as long as the interface is registered.

Registering new displays

Display routines

There are currently two system provided display routines: Text and Graph.

Text is very simple, displaying numerically the selected variables. For those variables that have changed since the previous update, it also gives the first difference as well. Hold is a boolean to allow the user to stop the display temporarily and Accumulate is a boolean to allow the first differences to be accumulated rather than reset each update. This is to simplify taking cumulative counts over a short interval of time.

Graph is more sophisticated and displays the recent history of each variable in histogram form. It automatically displays the absolute value or first difference value as appropriate to each variable. It has a form subwindow to control various parameters:

Hold
This boolean allows the user to temporarily inhibit updating the display. For those variables being displayed in first difference mode, this will result in an accumulated value on the next reading.

Normalize
This boolean allows the first difference variables to be normalized to a per – second value. This is to smooth out inaccuracies in the scheduling of the update process but users should be careful of how they interpret the normalized results.

Rescale!
The scale for each graph is initially set to an arbitrary value for each variable. For those graphs which are part of the current selection (see below), Rescale! will change them so that the maximum value within the selected region of a particular graph becomes the full scale reading for that graph.

Reset Origin!
The graphs normally "wrap around" once they are full. Reset Origin rotates the graph so the the current update point is just to the right of the graph. This command is particularly useful in conjunction with Hold! and either Print or the TajoInterpressCamera tool.

Clear!
The graphs are cleared (but the display process is not stopped).

Print! Filename:
Print the values of the selected range of the graphs in the current selection. This feature interacts nicely with filewindows: if the specified file is on display in a filewindow, it will be released by the filewindow during the Print! and will display the selected values as soon as Print! completes.

HistorySize: set! reset!
This allows the user to alter the number of values for each variable which will be shown in the histogram. Currently, no attempt is made to save any existing values in the graphs when a new historySize is set.

The current selection:
Some commands, notably Print and Rescale, use the current selection as an operand. Vertically, any subset of graphs can be selected and those selected need not necessarily be contiguous. Horizontally, any subrange of the graphs can be selected, and is the same for all the graphs selected. Selecting the subrange is very analogous to selecting text – Point and Adjust are used in the normal way. Multi – clicking Point changes the scale from bar to graph to everything (and then back to bar). Selecting a particular graph can be done with Point. Because non – contiguous subsets of graphs can be selected, selecting a set of graphs is slightly unconventional: either add individual graphs to the selection with Adjust or, with Adjust held down, drag the cursor over a set of graphs which are to be added to the selection. To remove a graph from the selection, double – click Adjust (and possibly drag it over others to be removed).

User.cm

Tom will pick up the standard entries of WindowBox, InitialState and TinyPlace from the [Tom] section of User.cm, if it exists. In addition, it recognises two further entries: Interfaces and Displays (with synonyms Interface and Display). These entries provide automatic preloading of user stubs of code. Each keyword can be followed by a list of names. For each name after Interface, Tom will generate a filename by appending "TomP.bcd" and then will try to load that file. For names after Display, the action is similar, except that "TomD.bcd" is appended. The names have no significance other than as part of a filename and need not necessarily be the same as the interface being registered. This allows one file to register many interfaces.

- - Tom.doc

Tom provides facilities for monitoring variables exported from "perf" interfaces - client interfaces containing performance and statistics information. Arbitrary interfaces can be monitored: clients merely have to provide a simple stub of code to make their private interface available to Tom. In addition, clients may supply different display routines if those provided are insufficient. Tom also includes facilities for accessing the "perf" - interfaces on a remote machine. This is done by loading a small part of Tom into that machine which is accessed from the user machine by Courier. The overhead of the monitoring is substantially less when monitoring remotely than when monitoring the local machine.

Using Tom

After starting, Tom has four sub - windows. The top is a message subwindow for displaying error messages. The second is a form subwindow containing fields and commands for selecting a set of variables to be monitored. The third is a text subwindow used to build up the list of interfaces and variables to be monitored. Text is normally entered via an accelerator in the form subwindow, but it can be typed in, moved, copied, edited or deleted in the normal way. The final subwindow is made available to the display routine.

The form subwindow:

Another!
Create another instance of the tool. The new instance will share all registered interfaces and display routines but is otherwise independent. This facility will typically be used to monitor different variables with different display routines.

Destroy!
Delete this instance of the tool. Tom is registered with the executive so if all instances are deleted, another can be created by typing "Tom" in an executive window.

Reset!
Stop the display (if any), clear the list of variables in the text subwindow and reset all values in the form subwindow.

Load! Save! Parameter File:
The current display parameters in the form subwindow (priority, interval, routine) and the text in the text subwindow can be saved in a file for later use by entering a file name and selecting Save!. The values can be reloaded with Load!. If no extension is given for the filename, ".tom" is used. All files ending in ".tom" will be given as a menu for Parameter File.

Remote Machine:
If this field is blank, Tom will directly access the registered interfaces on the local machine. Otherwise, this field should be the address or name of a machine known in the ClearingHouse and Tom will attempt to access the registered interfaces on that machine via Courier, even if the machine is the local one.

Add! Interface: Variable:
Interface: has a menu of the currently registered interfaces, and if one is selected then Variable: will have a menu of the variables available in that interface. Add! will cause interface.variable to be appended to the list in the list subwindow. There is a special 'variable' <ALL> which is interpreted by Add! with the obvious meaning.
The Timer interface is somewhat special in that Timer.milliseconds will always be internally prepended to the list when Apply! is selected (see below). There is therefore no need for the user to select this variable.

Apply! Display: Priority: Interval:
The desired display routine can be selected from the menu for Display. The priority (background, normal, foreground) and frequency of display update (default 1000mS) can also be set in the obvious fields. Apply! starts or changes a display routine. It parses the list of the selected variables and interprets them in the context of the currently selected Remote Machine (if any). The display will then start to monitor the specified variables. To stop a display, either use Reset! or select a new display. Similarly, to change the priority or frequency of update, simply change the appropriate parameters and select Apply!. If the list of variables has not changed the display routine will not be reinitialized (although the parameters of the update routine may have changed, of course).
The Interval: parameter is limited by the accuracy of Pilot and its process scheduling. Furthermore, it does not allow for time spent in the display routine. For this reason, there is a system provided variable Timer.milliseconds which is always included in the list of variables to be monitored. Timer.milliseconds always provides an accurate indication of the time since the previous update of the variables.

-- ToggleDisplay.doc

ToggleDisplay, when run will push a Tip Table/Interpreter onto the root that provides the following functions:

1) Toggle the state of the screen background (invoked via COM - I)
2) Toggle the state of the display (on or off) (invoked via COM - P)

It also provides two atoms On, and Off, as well as Toggle, that can be used as prefix arguments to the two command atoms Screen and Display.