Software Tools Subsystem
Reference Manual

T. Allen Akin
Terrell L. Countryman
Perry B. Flinn
Daniel H. Forsyth, Jr.
Jefferey S. Lee
Roy J. Mongiovi
Arnold D. Robbins
Peter N. Wan

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

September, 1984

The documentation contained herein pertains to Version 9 of the **Software Tools Subsystem** as implemented on the Prime 400 computer system at the School of Information and Computer Science of the Georgia Institute of Technology. While it is believed that the contents are completely accurate, neither the school, the institute, nor the authors assume any liability resulting from inaccuracies herein or from the use of this documentation or the Subsystem.

The text before you was prepared using the Software Tools text editor, 'se', and text formatter, 'fmt', both with Georgia Tech extensions.

Software Tools Subsystem Reference Manual

This manual is intended to serve as a reference for the manager of the Software Tools Subsystem and for users desiring more detailed knowledge of the workings of the Subsystem. It is divided into six sections:

Section 1: Commands
Descriptions of available commands.

Section 2: Library Subprograms
Descriptions of Subsystem library routines.

Section 3: Locally-Supported Commands
Descriptions of commands in the local command directory.

Section 4: Locally-Supported Library Subprograms
Descriptions of locally-supported subprogram libraries.

Section 5: Low Level Support Commands
Descriptions of low level commands, which are invoked by other commands. These commands should not be directly executed by the user under normal circumstances.

Section 6: Low Level Library Subprograms
Descriptions of low level subprograms, which are invoked by other subprograms. These routines should not be called by the user under normal circumstances.

The reader who wishes further discussion of how the Subsystem may be used effectively is referred to the *Software Tools Subsystem User's Guide*.

Key to Notation

Throughout this manual, (in Sections 1 and 3 in particular,) the syntax of commands is described through the use of various 'meta-symbols'. These symbols comprise a system of notation commonly known as 'Backus-Naur Form', or simply BNF. What follows is a brief description of the BNF that is used in this documentation.

<> A word or phrase enclosed in left and right angle brackets stands for any string of characters whose meaning is either suggested by the word or phrase so enclosed or explicitly defined later in the syntax. For example, "<number>" might stand for "127" or "3" or "98.6". Words or phrases enclosed in these brackets are called 'meta-linguistic variables'.

::= This symbol means "is defined as" and it is used to separate a meta-linguistic variable from its definition. For example,

<number> ::= <integer>

would be read "a number is defined as an integer." Everything to the right of the "::=" is called a 'meta-linguistic formula'.

| The vertical bar means "or" and is used to separate alternatives within a meta-linguistic formula. For example,

<number> ::= <integer> | <real>

would be read "a number is defined as an integer or a real."

() Parentheses are used to enclose a series of alternatives in a formula when the series comprises only one part of the formula. For example,

<signed_number> ::= (+|-)<number>

would be read "a signed number is defined as a plus sign or a minus sign, followed by a number."

[] Formulae (or parts thereof) that are enclosed in square brackets are optional. For example,

<command> ::= <filename> [<parameters>]

would be read "a command is defined as a filename, optionally followed by parameters."

{} Formulae that are enclosed in curly braces may be repeated any number of times, including zero. For example,

<integer> ::= <digit>{<digit>}

would be read "an integer is defined as a digit followed by zero or more digits."

In situations where the syntax requires that one of the above meta-symbols appear literally, the symbol is enclosed in apostrophes. For example, in

<vertical_bar> ::= '|'

the vertical bar on the right hand side is interpreted as a literal character, not as an "or" symbol.

TABLE OF CONTENTS

Section 1 - Commands

Section 2 - Library Subprograms

Section 3 - Locally-Supported Commands

Section 4 - Locally-Supported Library Subprograms

Section 5 - Low Level Support Commands

Section 6 - Low Level Support Library Subprograms

Permuted Index

This section of the Manual contains a key-word-in-context index produced from the headers of sections 1 through 6 with the commands 'kwic', 'sort', and 'unrot'.

Each line begins with the name of a command or subroutine, followed by the (parenthesized) number of the section in which it is documented. The lines are duplicated and rotated so that each keyword in the line appears to the right of the vertical white space that runs down the center of the page.

Section 1 - Commands

By far the most important component of the Software Tools Subsystem is its complement of commands. This section is devoted to the description of the commands currently available.

Documentation for each command is divided into the following sections. Note that sections that would otherwise contain no information will be omitted.

Header Line

The command's name, function, and the date of last modification to the documentation.

Usage

A description of the syntax permitted on the command line. The notation used in this description was discussed above in the section labelled "Key to Notation."

Description

A detailed coverage of the capabilities and operation of the command.

Examples

A few short examples of the command.

Files

A list of the names of special files used by the command.

Messages

A listing of important error messages or diagnostic information issued by the command.

Bugs

Known bugs in the operation of the command.

See Also

References to further information or related commands.

Usage

alarm ([in] <interval> [<units>] | at <time>)

Description

'Alarm' works like an alarm clock, allowing you to set when the alarm goes off. It displays the alarm set time, and then displays the current time in "hh:mm:ss" similar to 'clock'. When the alarm time is reached, 'alarm' sounds the terminal bell every second.

In the first usage format, <interval> is the number of time units before the alarm sounds, expressed as a positive decimal integer. It must be less than 32768. <Units> specifies the time unit. It may be:

"seconds" for seconds,
"minutes" for minutes,
"hours" for hours,

or omitted, in which case "seconds" is assumed. Abbreviations consisting of any initial substring of the above units are allowed. The word "in" may be included to enhance readability; its presence or absence is otherwise insignificant.

In the second format, the alarm will occur when the system clock registers the time of day specified by <time>. <Time> may be expressed in almost any common format. One guideline should be observed, however: a colon must be used to separate hours from minutes and minutes from seconds.

'Alarm' is terminated by typing control-P or by pressing the BREAK key.

Examples

```
alarm
alarm in 5 seconds
alarm at 12:50pm
alarm at 14:55:55
```

Messages

"Usage: alarm ..." for invalid argument syntax.

Bugs

Works only on CRT terminals.

See Also

clock (1), date (1), day (1), pause (1), time (1), date (2)

Usage

```
ar -(a[d] | d | p | t | u[d] | x)[v] <archive> {<file_spec>}
   <file_spec> ::= <pathname> | -n[<pathname>|<stdin_number>]
```

Description

'Ar' is a program designed to manipulate files that have been grouped together into a single "archive" file. Its principal utility is in keeping permanent backup copies of important files. Significant savings in disk space usage may also be realized by maintaining files in archives, resulting from the reduction of internal disk fragmentation.

Arguments to 'ar' consist of one of six directives, discussed below, followed by the name of the archive file, optionally followed by one or more <file_spec>s. The <file_spec>s, if present, designate names of files or archive members and are interpreted according to the specified directive. (For a full discussion of the syntax of <file_spec>, see the entry for 'cat' in section 1.)

The possible directives are:

-a Append. The named files are added to the end of the archive; if the archive did not previously exist, it is created. If any of the files is already contained in the archive, a diagnostic message is printed and the archive is not altered. If the "d" flag is specified and no errors are encountered in appending the new files, the files are deleted from the file system.

-d Delete. The named members are deleted from the archive.

-p Print. The named members are copied to standard output 1, one after another. Files are not necessarily printed in the order specified in the argument list; rather, they are printed in the order in which they appear in the archive. If no names are given, all members of the archive are printed.

-t Table. A table of contents of the archive file is printed on standard output 1. If file names are specified, information for only those members is printed.

-u Update. The named members of the archive are updated from the file system; if no names are specified, all members of the archive are updated. Any files named in the argument list that have no corresponding members in the archive are added to the end of the archive; a new archive may be created in this manner. If the "d" flag is specified and no errors are encountered, all the files named in the argument list are deleted from the

file system.

-x Extract. This directive is similar to the "-p" directive, except that the named members are written to individual files. Again, if no names are specified in the argument list, all archive members are extracted. The archive is not modified.

Any of these directives may be accompanied by the "v" flag, which causes 'ar' to print on standard output 1 the name of each archive member that it operates on. In the case of the "-t" directive, the "v" flag causes more information about each member to be printed. In the case of the "-p" directive, the name of the member is written out immediately before the contents of the member.

Examples

```
ar -tv arch.a
ar -x old_progs.a gamma.r gamma.b
ar -d backup.a rf.r
ar -ud archive.a ar.r ar.d
ar -pv archive.a ar.r >archive.r
lf -fc =src=/std.r | ar -u src.a -n
```

Messages

"Usage: ar ..." for incorrect argument syntax.
"archive not altered" when fatal errors occur and the original archive is left intact.
"<file>: can't add" when a file to be put in the archive can't be opened for reading.
"can't replace archive with <temp>" with the "-u" directive when the old archive can't be replaced with the newly created one. The new archive is left in the file <temp>.
"<member>: already in archive" with the "-a" directive when a named file is already in the archive.
"delete by name only" with the "-d" directive when no member names are specified.
"<file>: can't create" with the "-x" directive when a file can't be opened for writing to receive the extracted archive member, or with the "-a" and "-u" directives when a new archive file can't be created.
"can't handle more than <max> file names" when more than <max> names are specified in the argument list.
"<file>: duplicate file name" when the same name appears more than once in the argument list.
"archive not in proper format" when 'ar' is used on something other than an archive.
"<member>: not in archive" with the "-d", "-p", "-t" and "-x" directives when a member named in the argument list is not in the archive.
"<file>: can't remove" with the "-a" and "-u" directives

when a file can't be deleted from the file system.

"premature EOF" with the "-d", "-p", "-u" and "-x" directives when end of file is encountered on the archive during the copying of a member.

Bugs

There is no way to change the name of an archive member.

It takes a little longer than usual to extract archive members with the "-p" option when standard output is connected directly to the terminal.

See Also

cat (1), *Software Tools*

Usage

arg <argument_number> [<level_offset>]

Description

'Arg' is used from within a shell program to print an argument specified on the command line that invoked that shell program. <Argument_number> is the ordinal position of the argument desired. (A value of zero corresponds to the command name, one corresponds to the first argument, etc.) <Level_offset> is used to specify the number of levels of nested input files and/or function calls that are to be skipped before fetching the specified argument string. A value of zero means fetch the argument from the first higher nesting level; one means skip one level to the second higher level, etc. The string thus obtained is printed on standard output 1, followed by a newline.

Since 'arg' is typically used in a function call within a shell program, the default value of <level offset> is one, so that the level corresponding to the function call is skipped and the shell program arguments are accessed.

If <argument number> is out of range for the specified level, the empty string is returned and only a newline is printed.

Examples

```
print [arg 1]     # These two commands fetch the
arg 1 0           # same argument.

echo [arg 1] [arg 2] [arg 3]
```

See Also

args (1), nargs (1), getarg (2), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

args <first_argument> [<last_argument> [<level_offset>]]

Description

'Args' is similar in function to the 'arg' command, except that multiple arguments are printed. The first argument printed is specified by <first_argument>. If <last_argument> is specified, all succeeding arguments up to and including it are printed, separated from each other by newlines. Otherwise, all remaining arguments are printed, again, separated from each other by newlines.

Unlike 'arg', a newline is printed only if at least one argument was printed.

Examples

```
print [args 1]
pr [args 3 5]
```

Bugs

There is no way to specify a <level_offset> without specifying a <last_argument>.

See Also

arg (1), nargs (1), getarg (2), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

argsto <delim> [<num> [<start> [<level_offset>]]]

Description

'Argsto' is used from within a shell program to print a group of arguments specified on the command line that invoked that shell program. 'Argsto' prints the group of arguments delimited by arguments consisting of the string <delim>. <Num> is an integer that controls which group of arguments is printed. If <number> is 0 or omitted, arguments up to the first occurrence of <delim> are printed; if <number> is 1, arguments between the first occurrence and second occurrence of <delim> are printed, and so on. <Start> is an integer indicating the argument at which the scan is to begin; if <start> is omitted (or is 1), the scan begins at the first argument.

<Level_offset> is used to specify the number of levels of nested input files and/or function calls that are to be skipped before fetching the specified argument string. A value of zero means fetch the argument from the first higher nesting level; one means skip one level to the second higher level, etc. The strings thus obtained are printed on standard output 1, followed by a newlines.

Since 'argsto' is typically used in a function call within a shell program, the default value of <level offset> is one, so that the level corresponding to the function call is skipped and the shell program arguments are accessed.

Examples

```
rp [argsto / 2]
fc [argsto / 1]
```

See Also

args (1), nargs (1), getarg (2), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

banner { - | -c <char> } <string>

Description

'Banner' converts the text in the <string> argument into large characters for printing on a suitable hard copy printer. The printer should be able to handle 132 characters per line.

Output is produced on standard output 1 and may thus be piped to some other program or redirected to a file.

The dash argument, if present, causes the banner to be white-on-black; if absent, the banner is black-on-white.

The character used for printing the banner is the rubout, which appears on the line printer as a small rectangle composed of three vertical lines. This may be changed to any arbitrary ASCII character by using the "-c <char>" argument sequence.

The type font produced is Fortune Light, by the Bauer Type Foundry.

'Banner' is capable of producing all of the printable ASCII characters except for the following:

~ ^ \ ` { } [ ] _

Of these characters, three may be used to specify other special symbols: the caret ("^") is interpreted as the "degrees" symbol (superscript zero), the grave accent ("`") is interpreted as the 'cent' symbol, and the underscore ("_") is interpreted as the superscript 'th' symbol.

Examples

```
banner "Happy Birthday!" >saved_banner
banner - "Hi Mom"
banner "School of I. C. S." >/dev/lps
```

Messages

"Usage: banner ..." for improper arguments.

See Also

block (3)

Usage

basename [-(b | f | s | d | g)] { <pathname> }

Description

'Basename' is the function that knows about the syntax of pathnames and can select various portions of the name based on its arguments. It obtains input pathnames from its argument list, or from standard input if no arguments are specified, and prints the selected components on standard output. Options control the portion of the file name selected as follows:

Option	*Selects*
-b or none	the base file name only
-s	the file suffix only
-d	the directory path only
-f	the directory path and file name
-g	the file name and suffix only

Messages

"Usage: basename ..." for bogus arguments.

Examples

```
basename -s myprog.plg
cd [basename -d [file]]; [basename -g [file]]
ld [basename [file]].b -o [basename [file]]
```

See Also

take (1), drop (1), rot (1)

Usage

```
batch [[-k] <command>] {<options>}
batch -(s | d)[a | t] [<jobname>]
batch -(c | a | r) <jobname>
batch -m <jobname> {<options>}
   <options> ::= -a <acct> | -r | -n
      -h <home dir> | -t <cpu time> | -e <elapsed time> |
      -p <priority> | -q <queue>    | -f <funit>
```

Description

'Batch' provides a Software Tools Subsystem interface to the Primos batch subsystem. It subsumes the functionality of the Primos JOB command, providing in addition a printed listing of the output of the batch job, regardless of its disposition.

Scheduling a Batch Job

The first alternative of 'batch' allows the scheduling of Subsystem commands as batch jobs. <Command> is submitted to the batch queue along with commands to invoke the Subsystem and catch its terminal output. If <command> contains arguments, it should be surrounded by quotes so that it appears as one argument to 'batch'. If <command> begins with a hyphen, it must be preceded by the "-k" flag to identify it as a command. If <command> is omitted, 'batch' reads commands from standard input.

Before mentioning the available <options>, a few words must be said about the Primos batch subsystem. As configured at Georgia Tech, the batch subsystem has three queues, named "fast", "default", and "slow". When a job is scheduled to run in one of these queues, it takes on the default attributes assigned to that queue, unless otherwise specified. The queue attributes are as follows:

	fast	*default*	*slow*
Priority	2	1	0
Timeslice	1.0 sec	2.0 sec	4.0 sec
Cpu Time:			
Default	4 sec	200 sec	60 min
Maximum	8 sec	400 sec	unlimited
Elapsed Time:			
Default	30 sec	50 min	24 hrs
Maximum	60 sec	100 min	unlimited

When scheduling jobs, several options can be specified on the command line to change the default behavior. These options are as follows:

-q <Queue> is the queue name ("fast", "slow", or

"default") in which the batch job is to be scheduled. If this option is omitted, the job will be scheduled in queue "default".

-a <Acct info> is any string of accounting information desired. This information is for documentation only.

-h <Home dir> is the name of the directory in which the batch job is to be started. If this option is omitted, the job will start in the directory from which the 'batch' command is issued.

-t <Cpu time> is the maximum allowable cpu time **in seconds**, after which the job will be terminated. It must not be larger than the maximum allowable elapsed time for the queue. If this option is omitted, the default cpu time for the selected queue will be enforced as the maximum.

-e <Elapsed time> is the maximum allowable elapsed (wall-clock) time **in seconds**, after which the job will be terminated. It must not be larger than the maximum allowable elapsed time for the queue. If this option is omitted, the default elapsed time for the selected queue will be enforced as the maximum.

-p <Priority> specifies order within the queue as an integer from 0 to 9. It should normally not be used.

-f <Funit> is the Primos file unit from which the batch job is to obtain its input. The default is unit 6.

-r This option specifies that the job should be restarted upon system failure. If it is omitted, the job will not be restarted on system failure.

-n This option specifies that no batch print file be created for the job. If this option is omitted, a batch print file will be created summarizing the job's execution.

Obtaining Batch Job Status

The second alternative of the 'batch' command allows the user to see the status of selected batch jobs. Normally, a user may only request status information about his own jobs. There are two basic status requests: "s" (status), and "d"

(display). "S" produces a one line summary for each job while "d" produces a much more detailed summary.

The additional option "a" with either request returns information about active jobs (rather than completed or aborted jobs), while "t" returns information about jobs scheduled "today". If neither "a" or "t" is specified, all jobs scheduled within the last five days are displayed.

If <jobname> is specified, only information about the named job will be printed, if the job meets the other criterion set by the "a" or "t" option.

Cancelling Existing Jobs

The third alternative to the 'batch' command allows the cancellation of existing jobs. The "-a" option will cause the named job to be immediately aborted if it is executing, or cause it to be removed from the queue if it is not executing. The "-c" option causes the named job to be removed from the queue if it is not executing, but be allowed to continue if it is executing. The "-r" option forces immediate termination of an executing job, but returns it to the queue for re-execution.

Modifying Existing Jobs

Job attributes may be modified after a job is scheduled with the fourth alternative of the 'batch' command. Any options specified on the command will cause corresponding changes to the named job. The "-p <priority>" and "-q <queue>" options may not be specified.

Files

Numerous files in "//batchq".
=varsdir=/=user=.<line>.<seq>
=varsdir=/.batch_seq
=temp=/tm?*

Messages

"Usage: batch ..." for erroneous syntax.
"=varsdir=/.batch_seq: can't open"

Bugs

Job modification is not implemented.

See Also

ph (1), Primos phant$, Primos batch$

Usage

bye

Description

'Bye' is a shell program which may be used to log out from the Subsystem. It contains

stop -

For details on the action of 'stop', see its documentation.

Examples

bye

See Also

stop (1), Primos logo$$

Usage

```
case <value>
   when <alternative1>
      { <command> }
   when <alternative2>
      { <command> }
   ...
   out
      { <command> }
esac
```

Description

'Case' provides capabilities for conditional execution of commands in a manner similar to the case statements of the Algol 68 and Pascal programming languages. It allows a group of commands to be selected for execution based upon the value of some expression.

'Case' is always associated with a corresponding 'esac' command which marks the end of the scope of the 'case'.

'Case' accepts one argument to determine which of the subsequent groups of commands is to be executed. Any construct that yields a valid argument may be used. Each group of commands following 'case' is introduced by either a 'when' command or an 'out' command. The 'when' command takes a string argument which is compared with <value>. If the two match, the associated group of commands is executed and the remaining alternatives are skipped; otherwise, the associated commands are skipped. This proceeds until either an 'out' command or an 'esac' command is seen. If 'out' is seen, the associated command group is unconditionally executed; otherwise, the whole 'case' command results in no action. Thus, the commands associated with an 'out' command are executed by default, if no other alternative is selected.

Examples

```
case [term_type]
   when "b200"
      se -t b200 [args]
   when "diablo"
      ed [args]
   out
      echo "Unknown terminal type."
esac
```

```
case [login_name]
   when ICS002
      set name = Allen
      set class = 4A
   when ICS005
      set name = Dan
      set class = 6D
   when ISLAB
      set name = Perry
      set class = Staff
   out
      echo "I'm sorry, but I don't recognize you."
      set (name class) = ([login_name] UNKNOWN)
esac
```

Messages

"missing esac" when end of file is encountered before matching 'esac' command is seen

Bugs

The string on the 'when' command is not evaluated, so function calls, iteration, etc. are not allowed there.

See Also

if (1), esac (1), out (1), when (1), goto (1)

Usage

```
cat { <file_spec> | -h | -s }

<file_spec> ::= <filename> | -[<stdin_number>] |
                -n(<stdin_number>|<filename>)
```

Description

'Cat' concatenates the contents of the files specified in its argument list and writes the result on its first standard output. Files to be concatenated may be specified in any of several ways:

<filename>	an ordinary Subsystem pathname.
-<stdin_number>	a dash followed by a decimal number, 'n', designates the 'n'th standard input. 'n' must be a legal standard input number.
-	this is the same as specifying "-1" (i.e. standard input 1).
-n<stdin_number>	"-n" followed by a decimal number 'n' indicates that the names of the files to be concatenated are to be read from the 'n'th standard input.
-n	this is the same as "-n1".
-n<filename>	the names of the files to be concatenated are to be read from the named file.

If no arguments appear, the first standard input file is copied to standard output until end-of-file.

If the "-h" argument is given, 'cat' precedes the contents of each file copied with a header line consisting of twenty equals-signs ("=") followed by a blank, the name of the file, another blank, and twenty more equals signs.

If the "-s" argument is given, 'cat' will be "silent". In other words, if it cannot open a file, it will not complain. This is mainly for the benefit of shell scripts like 'sep', and the Subsystem 'build' procedures.

Examples

```
cat time_sheet
cat >junk
print_file> cat
prog | cat -2 - >two_and_one
```

```
files .r$ | cat -n
cat -h -nnamelist >/dev/lps
```

Messages

"<file>: can't open" if it can't open the named file, and the "-s" option was not specified.

See Also

copy (1), cp (1), print (1), pr (1), tee (1), gfnarg (2)

Usage

```
cc [<pathname>] [-afy]
                {-D<name>[=<val>] | -I<dir>}
                [-b[<b_pathname>]]
                [-s[<s_pathname>]]
                [-u<u_number>]
```

Description

'Cc' compiles the C program in <pathname>. It is an error to invoke 'cc' without a path name. The ".c" suffix on the source file name is optional, although 'cc' requires that the source code reside in a file named with a ".c" suffix. If the source file name specified in <pathname> does not have a ".c" suffix, 'cc' will append a ".c" and attempt to process a file with that name. The object code is stored in "<pathname>.b".

A full description of the C language is beyond the scope of this document. For complete information, refer to *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie (Prentice-Hall, 1978).

The following options are available:

-a Abort all active shell programs if any errors were encountered during processing. This option is useful in shell programs like 'ccl' that wish to inhibit compilation and loading if processing failed. By default, this option is not selected; that is, errors in processing do not terminate active shell programs.

-b Compile the source code into the object file named "<b_pathname>". 'Cc' effectively ignores this option if <b_pathname> is unspecified.

-f Suppress automatic inclusion of standard definitions file. Macro and common block definitions for the C Standard I/O Library and interfacing with the Subsystem reside in the file "=cdefs=". 'Cc' will process these definitions automatically, unless the "-f" option is specified.

-s Compile the source code into a PMA file named "<s_pathname>". The object code will be left in a file named "<s_pathname>.b" (".s" suffix replaced by ".b"). If <s_pathname> is not specified, 'cc' places the compiler output in "<pathname>.s" and the object module in "<pathname>.b". The "<s_pathname>" will over-ride any path name given to the "-b" option. In addition, 'cc' will *always* use 'vcg' to generate binary.

-u Reserved.

-y Check for potential problems, e.g. type mismatches. (This is similar in purpose to the Unix 'lint' program.)

-D Defines the identifier <name> with optional <value> for program internal use (maximum of 10).

-I Specifies a directory where include files reside (maximum of 10). All "-I" directories are searched after the current directory and before "=incl=".

Examples

```
cc file.c
cc prog.c -af
```

Messages

Numerous and self-explanatory.

Bugs

The "-a" flag doesn't always work yet.

The "-y" option complains about many things that are not problems. For instance, it does not know about the run-time library.

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

compile (1), ccl (1), ld (1), ucc (1), vcg (1), bind (3), c1 (5), *User's Guide for the Georgia Tech C Compiler*

Usage

ccl [<pathname>] [<'ld' args>] [/ <'cc' args>]

Description

'Ccl' is a shell program that compiles and loads the C program in <pathname>. If 'ccl' is invoked with no <pathname> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. If the source file name specified in <pathname> does not have a ".c" suffix, 'ccl' will append a ".c" and attempt to process a file with that name. The ".c" suffix on the source file name is not required, although 'ccl' requires that the source code reside in a file named with a ".c" suffix. The executable code is stored in <pathname>, or a file named appropriately from <'ld' args> (e.g., "-o gorf") or from <'cc' args> (e.g., "-b bonzo").

Options for 'ld' (names of libraries, for example) may follow the name of the source file, e.g. "ccl prog -l mylib". Special options for 'cc' may be placed after the 'ld' options, as long as they are separated by an argument consisting only of a slash; for example, "ccl prog -l mylib / -f". Aberrent command syntax may produce bizarre results.

Examples

```
ccl      # cc and ld the last file edited with 'e'

ccl profile
ccl profile.c

ccl change -l mylib
```

Messages

"<source_file>: can't open" for missing ".c" file

Bugs

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

compile (1), cc (1), vcg (1), ld (1), ucc (1), c1 (5), bind (3), *User's Guide for the Georgia Tech C Compiler*

Usage

cd [-p] [<pathname>]

Description

'Cd' changes the current working directory. <Pathname> gives the pathname of the target directory; if no arguments are present, the user's login directory is assumed.

If "-p" is given as the first argument, 'cd' prints on standard output the full pathname of the directory specified as the second argument. If there is no second argument, the full pathname of the current directory is printed. In neither case is the current working directory changed.

Examples

```
cd
cd =extra=/fmacro
cd subdir
cd -p
cd -p =src=
```

Messages

"bad pathname" when an invalid pathname is specified.

See Also

mkdir (1), Primos atch$$, Primos gpath$, gcdir$ (6)

Usage

```
cdmlc <input file>
         [-b [<output file>]]
         [-l [<listing file>]]
         [-z <CDML option>]
```

Description

'Cdmlc' serves as the Subsystem interface to the Prime DBMS Cobol DML preprocessor (CDML). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and output files as needed, and then produces a Primos CDML command and causes it to be executed.

The "-b" option is used to select the name of the file to receive the output Cobol code from the preprocessor. If a file name follows the option, then that file receives the output. If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".dcob". For example, if the input filename is "prog.cob", the output file will be "prog.dcob"; if the input filename is "foo", the output file will be "foo.dcob".

The "-l" option is used to select the name of the file to receive the listing generated by the preprocessor. If a file name follows the option, then that file receives the listing. If the "-l" option is specified without a file name following it or is not specified, a default filename is constructed from the input filename by changing its suffix to ".dl". For example, if the input filename is "gonzo.cob", the listing file will be "gonzo.dl"; if the input filename is "bar", the listing file will be "bar.dl".

The input filename must be a disk file name (conventionally ending in ".cob" or ".cobol").

In summary, then, the default command line for compiling a file named "file.cob" is

```
cdmlc   file.cob  -b file.dcob  -l file.dl
```

which corresponds to the CDML command

```
cdml -i *>file.f -b *>file.dcob -l *>file.dl
```

Examples

```
cdmlc file.cob
cdmlc payroll.cob -b b_payroll -l l_payroll
cdmlc funnyprog.cob -z"-newopt"
```

Messages

"Usage: cdmlc ..." for invalid option syntax.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable output file name" if an attempt was made to output on the terminal or line printer spooler.
"Sorry, the listing file must be a disk file" if the listing file was directed to a device file.
"Sorry, the output file must be a disk file" if the output file was directed to a device file.

Bugs

'Cdmlc' pays no attention to standard ports.

There is no way to avoid getting both a listing and output file.

See Also

cobc (1), csubc (1), ddlc (1), cdmlcl (1), ld (1), bind (3)

Usage

cdmlcl <program name> [<'ld' options>] [/ <'cobc' options>]

Description

'Cdmlcl' is a shell file that invokes the Prime DBMS Cobol DML preprocessor, the Primos Cobol compiler and the Primos segmented loader. If 'cdmlcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".cob", although in <program name> it may be specified with or without the ending ".cob". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'cobc' will be called with the <'cobc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
cdmlcl myprog.cob
cdmlcl myprog subs.b subs2.b -l mylib
cdmlcl myprog / -ok -l mylist
```

Messages

"<program name>.cob: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

cdmlc (1), cobc (1), ld (1), bind (3)

Usage

change <pattern> [<substitution> { <string> }]

Description

'Change' searches text strings for a pattern, changes each occurrence of that pattern to the specified substitution string, and writes the result on the standard output. The first argument specifies the pattern to be matched; the second (optional) argument specifies the substitution string to replace the matched string. If the substitution string is missing, it is assumed to be null (i.e., the matched string is deleted). Any additional arguments are taken as strings to be changed. Each is interpreted as a newline-terminated string; thus, lacking specific instances of the newline character in the <pattern> or <substitution> strings, each additional argument will cause one line of output to be produced. If no <string> arguments are supplied, lines of text to be changed are read from the standard input.

Patterns and substitution strings recognized by 'change' may take any form allowed in the text editor's substitute command. For a discussion of this syntax, refer to the documentation for the Subsystem text editor, 'ed', found in the *Introduction to the Software Tools Subsystem Text Editor*.

Examples

```
lf -c | sort | change "?*" "/mfd/&" >files
file.f> change "%C" "#" >file.r
change  ".pl1$"  ".l"  [source_file]
```

Messages

"Usage: change ..." if no arguments are supplied.
"illegal pattern string" for bad pattern.
"illegal substitution string" for bad substitution string.

See Also

ed (1), find (1), tlit (1), makpat (2), maksub (2), match (2), catsub (2)

Usage

```
chat { { <option> } { <pathname> } }
   <option>  ::=  -k <lock> | -m <date> <time> |
                  -p <protect> | -s [<depth>] | -u
   <lock>    ::=  sys | n-1 | n+1 | n+n
   <date>    ::=  mm/dd/yy
   <time>    ::=  hh:mm:ss
   <protect> ::=  {t | w | r | a}[/{t | w | r | a}]
   <depth>   ::=  <positive integer>
```

Description

'Chat' allows a user to change the attributes associated with a file or a group of files. Arguments to 'chat' are generally of the form:

```
<attributes> <files> { <attributes> <files> }
```

where <attributes> consists of a series of one or more options, and <files> is a list of files to which the specified attributes should be applied. Options consist of an option flag ("-" followed by a single character) usually followed by a string specifying the value of the attribute to be set. In most cases, this value string may either be a separate argument from the option flag, or appended to the option flag itself. The exceptions to this rule are the "-u" option which takes no value string, and the "-m" option which requires two value strings, each of which must be a separate argument.

The following options are available:

-k the lock that governs concurrent access to a given file by multiple users is set for each named file according to the value string which may be any of the following:

  sys the default system value is used; at most installations this is equivalent to "n-1".

  n-1 multiple readers or one writer.

  n+1 multiple readers and one writer.

  n+n multiple readers and writers.

-m the date and time of last modification is set for each of the named files according to the two value strings that follow. The first string specifies a date, and the second a time in military (24 hour) format.

-p the protection mode associated with the named files

is set according to the value string which is composed of two fields separated by a "/". The characters to the left of the "/" specify the types of access to be allowed to users with "owner" status, and those to the right specify the types of access to be given to users with "non-owner" status. The possible types of access are "truncate" (or "delete"), "write" and "read", represented by the characters "t", "w" and "r" respectively. If all three types of access are to be allowed, the character "a" may be used instead of "twr". If nonowners are to receive no access whatsoever, the slash may be omitted.

-s the named files are assumed to be directories and the specified attributes are applied to all files in the subtree rooted in those directories. If a value string is specified, it must be a positive integer that indicates the maximum number of levels below the named directories to which 'chat' is to descend. For example, if "-s1" is specified, only the files immediately contained in the named directory will have their attributes set.

-u turn on the "dumped" flag associated with each named file. Primos turns off the "dumped" flag associated with a given file each time the file is modified. When the file system is periodically dumped to tape, only those files whose "dumped" flags are off (i.e. that have been changed since the last backup) are actually copied to the tape. The "dumped" flags of those files are then turned on to indicate that the files have been backed up.

If no options (other than "-s") are specified, the attributes "-p a/r" are assumed. If the "-s" option is used and no <pathname> arguments are given, a pathname equivalent to that of the current directory is assumed.

To find the attribute values currently held by a file, use the 'lf' command.

Examples

```
chat -s
chat -u junkfile
chat -p wr/r f1 f2  -p a/wr f3
chat -s1 //src
```

Messages

"Usage: chat ..." for unrecognizable options.
"<pathname>: bad pathname" if a specified pathname refers to a non-existent file.
"<pathname>: not a directory" if the "-s" option is used on

a non-directory file.

The following messages occur when a specified attribute cannot be set:

"<pathname>: can't set protection"
"<pathname>: can't set modification date/time"
"<pathname>: can't set dumped bit"
"<pathname>: can't set read/write lock"

The following messages occur when specific argument syntax errors are detected:

"<arg>: bad date"
"<arg>: bad time"
"<arg>: bad protection mode"
"<arg>: bad lock specification"

See Also

lf (1), sacl (1), tscan$ (6), sprot$ (6)

Usage

clear

Description

'Clear' outputs the correct characters to clear a terminal screen. It calls 'vtinit' to get the user's terminal characteristics. If the terminal type is found, the screen is updated with the blank screen to clear it, otherwise 25 blank lines are output to clear the screen.

Examples

clear

Files

=vth=/<terminal_type>

See Also

vtinit (2), vtupd (2), and other VTH routines (vt?*) (2)

Usage

clock

Description

'Clock' generates the display for a digital clock, in the form "hh:mm:ss". It can be used on any CRT terminal that supports the "backspace" function. Time-of-day is guaranteed to be as accurate as the wristwatch of whoever last set the system time.

'Clock' is terminated by typing control-P or by pressing the BREAK key.

Examples

clock

Bugs

Works only on CRT terminals.

See Also

date (1), day (1), time (1), date (2)

Usage

cmp <string1> <relation> <string2>

Description

'Cmp' is a string comparison utility that is designed for use in function calls within arithmetic expressions. It compares the two strings given as arguments, and returns 1 if the specified relation holds, 0 otherwise. The following relations are supported (operators are the same as those in Ratfor, with some synonyms):

```
==   equal to
=    equal to
<    less than
>    greater than
<=   less than or equal to
=<   less than or equal to
>=   greater than or equal to
=>   greater than or equal to
~=   not equal to
<>   not equal to
><   not equal to
```

Notice that if the "greater than" symbol (">") is used in the <relation> argument, the argument must be quoted to prevent the shell from interpreting it as an I/O redirector.

Examples

```
if [cmp [day] = friday]; echo T.G.I.F.; fi
cmp [response] ~= "yes"
cmp [term] ">=" [term_list[i]]
```

Messages

"Usage: cmp ..." for invalid arguments.

Bugs

Redirection problem mentioned above.

See Also

case (1), eval (1), if (1), equal (2), strcmp (2)

Usage

cn <pathname> <new name> { <pathname> <new name> }

Description

'Cn' changes the names of the files named as arguments. Arguments must be paired; the first argument in a pair is the pathname of the file whose name is to be changed, the second argument in the pair is the new name to be given to the file. The new name must be a simple file name, not a pathname. Thus, 'cn' may not be used to move files from one directory to another. Use 'cp' for this purpose.

Examples

```
cn //cmdnc0/new_go go
cn old new   first last   always never
```

Messages

"<pathname>: missing name" if <new name> is missing
"<pathname>: bad pathname" if <pathname> could not be followed
"Usage: cn old new {old new}" for no arguments
"<new name>: already exists" for duplicate file name
"<pathname>: not found" for non-existent file name.
"<new name>: cannot move file to new directory" for an unescaped slash in the new name.

See Also

cp (1), Primos cname$

Usage

```
cobc {-<option>[<level>]} <input file>
         [-b [<binary file>]]
         [-l [<listing file>]]
         [-z <COBOL option>]
   <option> ::= d | m | v | x
```

Description

'Cobc' serves as the Subsystem interface to the Primos Cobol compiler (COBOL). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos COBOL command and causes it to be executed.

Options

The general structure of an 'cobc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-d(1) [0..2] - Debugging.

Level 0 causes debugging statements in the source program to be ignored.

Levels 1 and 2 cause debugging statements in the source program to be compiled.

-m(1) [1..2] - Addressing.

Level 1 causes the compiler to generate code in 64R mode.

Level 2 causes the compiler to generate code in 64V mode.

-v(1) [1..2] - Listing verbosity.

Level 1 generates a full source code listing containing the machine code representation of each instruction.

Level 2 generates a full source code listing that includes the code generated by all macro calls.

-x(1) [0..2] - Cross-reference listing control.

Level 0 causes the compiler to generate no cross reference listing at the end of the source program listing.

Levels 1 and 2 cause the compiler to generate a full cross-reference of all variables at the end of the source listing.

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.cob", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.cob", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".cob" or ".cobol") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.cob" is

```
cobc -d1m1v1x1 file.cob -b file.b -l /dev/null
```

which corresponds to the COBOL command

```
cobol -i *>file.cob -b *>file.b -l no
```

Examples

```
cobc file.cob
cobc -xm2 payroll.cob -b b_payroll -l l_payroll
cobc -v2 funnyprog.cob -z"-newopt"
```

Messages

"Usage: cobc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'cobc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Cobc' pays no attention to standard ports.

See Also

cobcl (1), ld (1), bind (3)

Usage

cobcl <program name> [<'ld' options>] [/ <'cobc' options>]

Description

'Cobcl' is a shell file that invokes the Primos Cobol compiler and the Primos segmented loader. The program is compiled and linked in 64V mode. If 'cobcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".cob", although in <program name> it may be specified with or without the ending ".cob". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'Cobc' will be called with the <'cobc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
cobcl myprog.cob
cobcl myprog subs.b subs2.b -l mylib
cobcl myprog / -dx -l mylist
```

Messages

"<program name>.cob: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

cobc (1), ld (1), bind (3)

Usage

```
col { -c <columns> | -g <gutter width> | -i <indent> |
      -l <page length> | -w <column width> | -t }
```

Description

'Col' is a filter that reads lines from standard input and writes multi-column pages on standard output. The arguments control what assumptions are made about such things as the size of the input lines, the length of the output page, the number of columns per page, and so on; any combination of the following may be used:

-c may be used to control the number of columns per page; it must be followed by a positive integer. The current implementation of 'col' restricts the maximum number of columns per page to 8. If "-c" is omitted, two columns per page is assumed.

-g may be used to set the width of the "gutters" that separate the columns from each other; it must be followed by a non-negative integer. If "-g" is omitted, five blanks are placed between columns.

-i may be used to set a running indentation of the left margin and must also be followed by a non-negative integer. If no "-i" is given an indentation value of zero is assumed.

-l may be used to specify the number of lines on each page of output and must be followed by a positive integer. If it is omitted, 'col' assumes a page length of 54 lines, which incidentally is the number lines placed on each page by the 'print' command.

-w may be used to set the width of each column and should also be followed by a positive integer. To allow lines containing backspaces and overstruck characters whose length exceed their printed width, 'col' never truncates input lines; consequently, best results occur when all the input lines have a printed width no greater than the specified value. If "-w" is omitted, three inch wide columns are produced (i.e., 30 characters per column, printed at 10 characters per inch).

-t may be used to select parameter values suitable for generating output on a CRT screen. Specifically, this option selects five columns of 14 characters each per 22 line page with two character gutters and no indentation. The output generated under these parameters is suitable to be piped into the 'pg' command. If additional options are used, the parameter values so specified override those selected by "-t".

Examples

```
file> col | print
files .r$ | col -t | pg
paper> col -c 2 -w 60 -l 66 >/dev/lps
```

Messages

"Usage: col ..." for improper arguments.
"too many columns" if more that 8 columns are requested.
"too many lines" if there is inadequate buffer space to hold an entire page.

Bugs

The default parameter values are probably wrong. Misbehaves when input lines contain more backspaces than printable characters.

See Also

pg (1), print (1)

Usage

common [-{1 | 2 | 3}] [<file1> [<file2>]]

Description

'Common' prints the lines common to two sorted files. It normally produces three columns of output: Column one contains lines present in <file1> but not present in <file2>; column two contains lines present in <file2> but not in <file1>; and column three contains lines common to both files.

The first argument may be used to select the columns to be printed. A dash followed by a "1" selects the first column, a dash followed by "12" selects columns one and two, etc. For example, to print lines in the second file or in both files (i.e. columns two and three), the argument should be "-23".

If the second file name argument is omitted, the first standard input is used for <file2>; if no <file >name arguments appear, the first and second standard inputs will be used for <file1> and <file2> respectively.

Examples

```
lf -c =bin= | sort >file1;
   lf -c =doc=/fman/s1 | sort >file2;
      common -1 file1 file2

common -1 wordlist =dictionary=
```

Messages

"Usage: common ..." for illegal arguments.

See Also

diff (1), sort (1), lf (1)

Usage

```
como { -{c | n | p | t} } [ <pathname> ]
```

Description

The 'como' command is used to control the destination of command output; that is, output from a program that would otherwise appear on the terminal. (This is in no way related to the redirection of standard inputs and outputs provided by the Subsystem.) It is useful in conjunction with phantoms or long command files that are usually run without human supervision.

Command output may be routed to the terminal (the normal case), a file, both the terminal and a file, or to neither destination (in which case the output is lost). The options are as follows:

-c (Continue.) If a <pathname> argument is specified, subsequent command output is appended to the named file; otherwise, output to a previously opened file is continued (see the "-p" option). Terminal output is not affected.

-n (No output to terminal.) Terminal output is turned off. File output is not affected.

-p (Pause.) File output is turned off. The file is not closed, so that file output may be subsequently resumed with a "como -c" command. Terminal output is not affected.

-t (Output to terminal.) Terminal output is turned on. The use of this option in no way affects the status of file output.

In all cases, the specification of a <pathname> results in the opening of the named file and the turning on of file output, even when the "-p" option is specified. When used without any arguments, 'como' closes any file that may have been receiving command output, turns off file output, and turns on terminal output.

Examples

```
como listing
como
como -cn save
```

Messages

"Usage: como ..." for invalid argument syntax.
"bad pathname" the <pathname> could not be found.

Bugs

If a <pathname> is specified and the file did not previously exist, a direct access file is created, rather than a sequential file.

See Also

Primos como$$, Primos COMO command

Usage

```
compile {<input_files>} [-c] [-m <language>] [-C<'cc' options>]
      [-R<'rp' options>] [-F<'fc' options>] [-S<'pmac' options>]
      [-P<'pc' options>] [{-l <library>}] [-o <output_file>]
```

Description

'Compile' is a general purpose interlude for calling the various compilers available. The choice of compiler is determined by the suffix of the file name.

'Compile' compiles and loads the pathnames specified. The following options are available:

-c Compile only. The various source files will be compiled, but the loader will not be called.

-m <language>
Specify a "main" language. If the "main" language requires a special library and/or start-off routine, then 'compile' will arrange to load it. The <language> should be one of the suffix letters listed below. By default, no special libraries (besides the regular "vswtlb") will be loaded.

-l <library>
Load <library>.

-o <output file>
Place executable file in <output file>.

'Compile' recognizes the following file naming conventions and will utilize the appropriate preprocessor and/or compiler:

.c -- C source file
.s -- Pma source file
.r -- Ratfor source file
.f -- Fortran 66 source file
.p -- Pascal source file

Therefore, if your current directory contains the files "f1.s", "f2.c", "f3.r", "f4.f", and "f5.p" and you execute the command "compile f1.s f2.c f3.r f4.f f5.p", 'compile' will call the appropriate language processors for each file and load the resulting binary versions together. Note that even though there are both C and Pascal files listed, their special libraries would *not* be loaded.

Every path name that you specify *must* include its associated suffix. Otherwise, 'compile' will decide that it is not a file, but an argument to pass on to the loader.

The following options will be used by the indicated compiler

when it processes those pathnames having the corresponding name extensions. Options to be passed on to the compilers should be enclosed in quotes, so that they will stay grouped together. For instance:

```
compile -m c junk.c -C'-a -Dindex=strchr' stuff.r
-R'-a -g' -o junk
```

Otherwise, the shell will split them up, and most of the options will go to the loader, and do something unexpected, instead of to the intended compiler.

-C <'cc' options>
Use the 'cc' options specified when compiling C modules.

-R <'rp' options>
Use the 'rp' options specified when preprocessing any Ratfor modules.

-F <'fc' options>
Use the 'fc' options specified when compiling Fortran modules. These options will affect Ratfor programs as well.

-S <'pmac' options>
Use the 'pmac' options specified when assembling PMA modules. These options will *not* affect C programs, since the C compiler no longer uses PMA to compile its programs.

-P <'pc' options>
Use the 'pc' options specified when compiling Pascal modules.

The options should not occur more than once; if they do, the last one will be used. Unrecognized options will be passed on to the loader.

Messages

"Usage: compile ..." for an invalid option to the '-m' flag, if no arguments are given, or no files are listed (only options).

Examples

```
compile -m c sort.c stuff.p
compile prog1.r prog2.p low_level.s -l vswtmath -o prog
```

Bugs

Does no sanity checking on the arguments passed to the

individual compilers, nor on what is passed on to the loader.

Cannot be used to call 'bind'.

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

cc (1), rp (1), fc (1), pc (1), pmac (1), ld (1), ucc (1), bind (3), *User's Guide for the Georgia Tech C Compiler*

Usage

```
copy
```

Description

'Copy' is Kernighan and Plauger's copy command from chapter two of *Software Tools*. It simply copies its standard input to its standard output until end-of-file.

Examples

```
file1> copy >file2
list_file> copy
copy >data
copy
```

See Also

cat (1), cp (1), print (1), fcopy (2)

Usage

copyout

Description

'Copyout' opens a file in the spool queue and diverts the user's command output into the file. This diversion can be stopped by logging out or issuing a 'como' command.

'Copyout' is intended for use by 'batch' to produce a batch job listing, but it may accidentally find use in other situations.

Examples

copyout

Files

//spoolq/prt??? for spool output

Bugs

Diverting a screen editor session to the line printer is very messy.

See Also

batch (1), como (1)

Usage

```
cp [-m] [-p] [-s [<depth>] ] <from> [ <to> ]
```

Description

'Cp' copies files or directories from one place in the file system to another. The single required argument '<from>' specifies the source file or directory. The '<to>' argument, which is optional, may be used to specify the destination file or directory. Omitting this argument produces the same effect as specifying the pathname of the current working directory. The precise result of any invocation of 'cp' depends on whether or not the destination is an existing directory and, in the case where the source is a directory, whether the "m" and/or "s" options is specified. For a more detailed explanation of the semantics of the "s" option, see the Reference Manual entries for the 'chat', 'del' and 'lf' commands. The various cases are elaborated below.

If the destination is an existing directory, the source is normally copied *into* that directory, retaining its original name. If the source is also a directory, its contents may be merged into the destination directory by specifying the "m" option. Otherwise, the source directory will be copied as a subdirectory of the destination directory.

If the destination is not an existing directory, the destination file is exactly as specified by the '<to>' pathname.

If the "p" option is specified, any directories created in the process of copying are given the same passwords as their counterparts in the source. If the option is not specified, these directories are given default passwords. (At installations running the Ga. Tech version of Primos, the defaults are the user's login name for the owner password and zeroes for the non-owner password; at installations running standard Primos, the defaults are blanks for the owner password and zeroes for the non-owner password.)

In all cases, the protection, date-modified and read/write lock attributes of the copied files are set identically to those of their source counterparts.

Examples

```
cp file //dir/file
cp file //dir
cp file
cp file1 file2
cp old_dir new_dir
cp -p old_dir new_dir
```

Files

None.

Messages

"Usage: cp ..." for illegal argument syntax.
"<source>: can't open" if source file can't be opened for reading.
"<destination>: can't create" if destination can't be created.
"<source>: copy incomplete" if an error occurred while moving the contents of the source file to the destination.
"<destination>: non-empty directory" when the source is an ordinary file and the destination is a non-empty directory.

Bugs

Works only on disk files.

Cannot copy specific ACL's or access categories

See Also

cat (1), chat (1), cn (1), copy (1), del (1), lf (1)

Usage

```
crypt [ <key> ]
```

Description

'Crypt' encrypts data from its first standard input based upon an encryption key supplied as an argument, and writes the result on its first standard output.

'Crypt' uses a reversible "exclusive-or" algorithm so that cipher text encrypted with a given key may be decoded using the same key.

If the <key> is omitted from the command, 'crypt' turns off the terminal echo and prompts for the key from the terminal.

Examples

```
sensitive_data> crypt bogus-key >safe_data
secret_message> crypt turkey
```

Messages

"Key: " for a missing key

Usage

cset [-i <int> | -k <key> | -m <mnemonic>] [-o (i | k | m)]

Description

'Cset' is a command that lists various information about the ASCII character set. The following arguments may be used to select a certain ASCII character for display:

-i Information is listed for the character whose integer value is <int>. If <int> is in the range 0 through 127 inclusive the character that will actually be listed is integer value <int> + 128 since Prime convention is mark parity for all ASCII characters, otherwise <int> must be in the range 128 to 255 inclusive. <Integer> may be entered in any radix using the <radix>r<int> format.

-k Information is listed for the character whose keycode matches <key>. Keycodes are the actual characters typed to enter the character: the character itself if it is simply an upper or lower case character, or an up arrow (^) to represent the control key followed by the character typed while holding the control key down. The only exception is for the rubout key, which is represented as ^#.

-m Information is listed for the character whose mnemonic matches <mnemonic>. Mnemonics are standard ASCII mnemonics in upper or lower case.

If none of the above options are present, information for all ASCII characters is listed.

The following argument may be used to select the output format:

-o If the string following this argument begins with an "i" (in upper or lower case), then the output will be the base 10 integer value of each character selected for display by the above arguments. If the string following this argument begins with a "k", then the output will be the keycodes corresponding to the selected characters. And if the string following this argument begins with an "m", then the output will be the mnemonics for the selected characters.

If this argument is omitted, output will consist of the integer value of the selected characters in bases 10, 8, and 16, together with the keycode and mnemonic associated with

those characters.

Examples

```
cset
cset -k ^p
cset -m del -o i
cset -i 8r200 -o m
```

See Also

ctomn (2)

Usage

```
csubc <input file>
         [-l [<listing file>]]
         [-z <CSUBS option>]
```

Description

'Csubc' serves as the Subsystem interface to the Prime DBMS Cobol subschema compiler (CSUBS). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and output files as needed, and then produces a Primos CSUBS command and causes it to be executed.

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. If the "-l" option is specified without a file name following it or is not specified, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.csub", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l".

The input filename must be a disk file name (conventionally ending in ".csub").

In summary, then, the default command line for compiling a file named "file.csub" is

```
csubc file.csub -l file.l
```

which corresponds to the CSUBS command

```
csubs -i *>file.csub -l *>file.l
```

Examples

```
csubc file.csub
csubc payroll.csub -l l_payroll
csubc funnyprog.csub -z"-newopt"
```

Messages

"Usage: csubc ..." for invalid option syntax.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"Sorry, the listing file must be a disk file" if the listing

file was directed to a device file.

Bugs

'Csubc' pays no attention to standard ports.

There is no way to avoid getting a listing file.

See Also

ddlc (1), cobc (1), cdmlc (1), ld (1), bind (3)

Usage

ctime

Description

'Ctime' prints the user's elapsed CPU time since login (in seconds) on standard output 1.

Examples

ctime

See Also

profile (1), time (1), clock (1)

Usage

cto [<string>]

Description

'Cto' copies its first standard input to its first standard output, terminating either at end of file or the first occurrence of <string>. In order to be recognized, <string> must appear on a line by itself. This termination line is not copied. If no argument is specified, <string> defaults to "-EOF".

'Cto' is useful in shell files for terminating programs that read from the command stream. It is virtually a necessity for generating end-of-file on terminals that cannot generate a control-c character.

Examples

```
>> cto | x
paron file1 file2
delete file1
-EOF
```

See Also

cat (1), copy (1), slice (1)

Usage

date

Description

'Date' prints the Gregorian date in the form mm/dd/yy on standard output one.

Examples

```
date
echo "Run at" [time] "on" [date]
```

See Also

day (1), time (1), date (2)

Usage

day [<dd> | <mm>/<dd> | <mm>/<dd>/<yy>]

Description

'Day' prints the name of the day of the week (e.g. Monday, Tuesday, Wednesday, etc.) on standard output one. The name is printed in lower case with the first character capitalized.

Should no arguments be given, the name of the current day is printed. Optionally, a day in the current month, in a different month but the current year, or in a different month and year may be given as an argument, and the day associated with that date will be printed.

Examples

```
day
echo Today is [day] [date]
day 30
day 01/01/99
```

Bugs

Argument format restricts usefulness to the Twentieth century.

See Also

date (1), time (1), date (2)

Usage

dbg { <DBG option> } <program> { <arguments> }

Description

'Dbg' allows the user to access the facilities of the Primos source level debugger (DBG) while still in the Subsystem. <Program> is a Subsystem program that has been linked by 'ld' with the "-d" option (i.e., it is a segment directory). 'Dbg' sets up the standard input and output ports and <arguments> for access by the program and then executes DBG with a call to Primos routine CP$.

Examples

```
dbg -vfyi -vfyp prog.r> new_rp >prog.f
dbg test.o -s -t 3
```

Messages

"command too long" for too many DBG options to fit on a Primos command line.

Bugs

If DBG bombs (as it has been known to do), the Subsystem must be reinitialized with the sequence "dels all;dels 6002" and then "swt".

Ratfor programs must be debugged using the Fortran names and line numbers (yuk!).

When DBG terminates (with the "q" command) it exits to PRIMOS. Typing "ren" will return back to the subsystem.

See Also

fc (1), f77c (1), pc (1), plgc (1), ld (1)

Usage

```
ddlc <input file>
          [-l [<listing file>]]
          [-z <SCHEMA option>]
```

Description

'Ddlc' serves as the Subsystem interface to the Prime DBMS schema compiler (SCHEMA). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and output files as needed, and then produces a Primos SCHEMA command and causes it to be executed.

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. If the "-l" option is specified without a file name following it or is not specified, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.ddl", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l".

The input filename must be a disk file name (conventionally ending in ".ddl").

In summary, then, the default command line for compiling a file named "file.ddl" is

```
ddlc file.ddl -l file.l
```

which corresponds to the SCHEMA command

```
schema -i *>file.ddl -l *>file.l
```

Examples

```
ddlc file.ddl
ddlc payroll.ddl -l l_payroll
ddlc funnyschema.ddl -z"-newopt"
```

Messages

"Usage: ddlc ..." for invalid option syntax.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"Sorry, the listing file must be a disk file" if the listing

file was directed to a device file.

Bugs

'Ddlc' pays no attention to standard ports.

There is no way to avoid getting a listing file.

See Also

cdmlc (1), csubc (1), fdmlc (1), fsubc (1)

Usage

```
declare { <identifier> [ = <value> ] }
```

Description

'Declare' is the primary method of creating shell variables with local (i.e., to the command file) scope. Its arguments are the names of the variables to be declared; they are declared at the current lexical level and assigned the specified values. If a value is not specified for a variable, it is given the empty string as a value. Value may contain unprintable characters in a mnemonic format. The format is '<' ascii_mnemonic '>'. To set dummy to a dash followed by a control-g and then another dash one would say:

```
declare dummy = "-<bel>-".
```

The quotes are needed to prevent the shell from interpreting the '<' and '>' signs as I/O redirectors. Variables declared within a command file exist as long as that command file is active; when its execution is complete, they disappear. If a variable of the same name is already declared at that level, its value is not changed.

Variables may also be created by the 'set' command.

Examples

```
declare name address telephone_number
declare terminal_type
declare i = 1 bel = "<bel>"
declare nobel = "@<bel>"
```

Bugs

Does not complain about multiple declarations of a variable within a given scope.

See Also

forget (1), set (1), vars (1), save (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

declared <variable_name> [<level_offset>]

Description

'Declared' tests for the existence of a shell variable named <variable_name>. If the variable exists, 'declared' prints "1"; otherwise it prints "0".

If <level_offset> is omitted, 'declared' examines all lexic levels for <variable_name>. Otherwise, only the level specified (<current_level> - <level_offset>) is searched. (See 'arg' for a more complete discussion of the <level_offset> mechanism.)

Examples

```
if [declared se_params]
   se_params
else
   echo ""
fi
```

See Also

vars (1), arg (1), set (1), forget (1), save (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

define [-(f | m)] {<input_file>}

Description

'Define' is a text substitution facility used to replace defined identifiers by their definitions. 'Define' takes the file(s) specified in the argument list, processes **define** statements and **undefine** statements, and places the output on its standard output file. 'Define' also processes **include** statements. For more information on **define** and **undefine** statements see the *Ratfor Programmer's Guide*.

In addition to the way that Ratfor handles the 'define' statement, this processor will allow the user to prevent premature evaluation of a given string by enclosing it in brackets, similar to 'macro' (please see the Reference Manual entry for the 'macro' command).

The following options are available:

-f Suppress automatic inclusion of standard definitions file. Macro definitions for the manifest constants used throughout the Subsystem reside in the file "=incl=/swt_def.r.i". 'Define' will process these definitions automatically, unless the "-f" option is specified.

-m Map all identifiers to lower case. When this option is selected, 'define' considers the upper case letters equivalent to the corresponding lower case letters, except inside quoted strings.

The remainder of the command line is used to specify the names of the input file(s). If no input file is specified, 'define' will expect input from standard input. Output will be sent to standard output.

Examples

```
define file1.r
file> define -f
define -m file1 file2 file3
```

Files

=incl=/swt_def.r.i for standard Subsystem macro definitions

Messages

"missing left paren in define"

"non-alphanumeric name in define"
"missing right paren in define"
"missing parameter in definition"
(two commas in a row)
"non-numeric parameter not allowed"
"too many parameters"
(more than 32 parameters)
"missing comma in parameter list"
"missing comma in parameter list"
(no comma between the parameter list or the name and the definition)
"invalid file name in include"
"includes nested too deeply"
(more than five levels deep)
"can't open include file"
"definition too long"
(more then 400 characters long)
"missing right paren after definition"
"missing left paren after undefine"
"non-alphanumeric name in undefine"
"missing right paren after undefine"
"line too long"
"unexpected EOF"

See Also

macro (1), rp (1), *Ratfor Programmer's Guide*

Usage

```
del { -<opt>{<opt>} } { -n | <path> }
   <opt>    ::=   d | f | s[<depth>] | v
   <depth>  ::=   [ <positive integer> ]
```

Description

'Del' is a general purpose file deleter. When invoked with a list of one or more pathnames as arguments, it attempts to remove each file named. The list of pathnames may be preceded by zero or more control arguments, each consisting of a hyphen, followed by one or more of the following letters:

-d when specified in combination with the "s" option (see below), this option causes 'del' to delete a UFD named as an argument after having deleted its contents. (Normally, 'del' deletes the UFD's contents and leaves the UFD itself intact.)

-f when specified, causes 'del' to attempt to manipulate the protection attributes of each file that it is about to delete to insure that the file is, in fact, deletable. Note, however, that this can only be done when the file resides in a directory that is public, owned by the user of 'del', or protected with an acl so that the user has protect privileges (ie - can change the acl protection on the object). For objects in password directories, the protection bits are modified to give the user all privileges as an owner. For ACL protected objects, the protecting object is modified to give the user "delete", "list", and "use" privileges.

-s when specified, causes 'del' to traverse the subtree of the file system descending from a UFD or segment directory named as an argument, attempting to delete each file it finds along the way. If a decimal number immediately follows the "s", then 'del' will descend to no more than that many levels below the named directory in its traversal. This option requires at least one of the arguments <path> or "-n" be specified (since directories must be deleted by name). Normally, when the named directory is a UFD, the directory itself is not deleted -- only its contents are. But if the "d" option (see above) is specified, the UFD too will be deleted. Users are exhorted to USE THIS OPTION WITH UTMOST CAUTION.

-v when specified, 'del' will print the pathname of each file before attempting to delete it, and will wait for a one-line response from the user. Only

if the line begins with a "Y" or a "y" will the file be deleted; otherwise, the file will be left intact.

If the string "-n" appears in the place of a <path> argument, 'del' will read arguments from its first standard input port until end-of-file is encountered. It is assumed that each line to be read contains a single path name, starting in column 1.

Examples

```
del lkj
del -dfs segdir subufd
del -vs1
del -s [cd -p]
files %junk | del -n
```

Messages

"Usage: del ..." for bad arguments
"<path>: in use" for files open by other users
"<path>: protected" for undeletable files
"<path>: not found" for non-existent files
"<path>: delete protected" for files with delete protection turned on
"<path>: can't delete" for unexpected file system error
"<path>: can't attach" for pathnames that can't be followed
"<path>: directory not empty" for trying to delete a non-empty directory
"<path>: directories nested too deeply" when directories are nested more deeply that 'del' can handle
"<path>: error reading directory" for unexpected error in reading a segment directory
"<path>: bad pathname" for non-existent files
"delete current directory by name only" when no arguments are given

Bugs

When deleting an ACL directory with the "-d" and "-f" option, if the top level directory cannot be deleted, the ACL protection attributes may be left changed.

A file cannot be deleted with the force option if there is more than one level of protection between the object to be deleted and the object protecting it. For example, if a file is protected by a directory which is protected by an access category then specifying the "-f" option when attempting to delete the file will not work.

See Also

cp (1), lf (1), mkdir (1), sacl (1), remove (2), create (2), gfdata (2), sfdata (2), sprot$ (6)

Usage

```
detab { -t <tab character>          |
        -r <replacement string>     |
        <column number>             |
        +<increment> }
```

Description

'Detab' expands tab characters on its first standard input file into an equivalent number of replacement characters on its first standard output file.

The tab character may be specified by an argument following the "-t" option; if not so specified, the ASCII TAB (ctrl-i) is assumed. Similarly, the string from which replacement characters are taken may be specified using the "-r" option. Replacement characters are taken as needed from the string, starting with the first and wrapping around to the beginning when the end of the string is reached. If no replacement string is specified, a single blank is assumed.

Any number of tab stops may be set by specifying the desired column number as an argument. If the "+<increment>" construct is used, stops will be set at intervals of <increment> columns, starting with the most recently set stop. Thus, the argument sequence

```
10 +5
```

would set stops in columns 10, 15, 20, etc. In the absence of any other specification, default stops are set in every fourth column, starting with column five.

Examples

```
file1> detab 21 36 41 66 -r " ." >file2
cat subr1 subr2 subr3 | detab +3 >prog.r
assembler.s> detab -t \ 10 20 35 >asm.s
```

Messages

"Usage: detab ..." for incorrect arguments.

See Also

entab (1)

Usage

diff [-{b | c | d | r | s | v}] [<old_file> [<new_file>]]

Description

'Diff' compares the contents of two files and reports on the differences between them. The default behavior is to describe the insert, delete, and change operations that must be performed on <old_file> to convert its contents into those of <new_file>.

Both file name arguments are optional; if the second is omitted, the first standard input is used for <new_file>; if neither argument appears, the first and second standard input are used for <old_file> and <new_file> respectively.

The options currently available are:

-b Perform a word-for-word binary comparison. 'Diff' will compare corresponding words of the two input files; if any differences are found, or if one file is shorter than the other, 'diff' prints the message "different" and exits. If the files are the same, 'diff' produces no output. When the "-v" option (see below) is specified, 'diff' prints an octal representation of the words that differ along with their offset from the beginning of the file, and notifies the user if one file is shorter than the other.

-c Perform a simple line-by-line comparison. 'Diff' will compare successive lines of the input files; if any corresponding lines differ, or if one file is shorter than the other, 'diff' prints the message "different" and exits. If the files are the same, 'diff' produces no output. When the "-v" option (see below) is specified, 'diff' prints the lines that differ along with their line number in the input file, and notifies the user if one file is shorter than the other.

-d List the "differences" between the two files, by highlighting the insertions, deletions, and changes that will convert <old_file> into <new_file>. This is the default option. If the "verbose" option "-v" (see below) is specified, unchanged text will also be listed.

-r Insert text formatter requests to mark the <new_file> with revision bars and deletion asterisks. This option is particularly

useful for maintenance of large documents, like Subsystem Reference Manuals.

-s Output a "script" of commands for the text editor 'ed' that will convert <old_file> into <new_file>. This is handy for preparing updates to large programs or data files, since generally the volume of changes required will be much smaller than the new text in its entirety.

-v Make output "verbose". This option applies to the "-b", "-c" and "-d" options discussed above. If not selected, 'diff' produces "concise" output; if selected, 'diff' produces more verbiage.

Examples

```
diff myfile1 myfile2
diff rf.r nrf.r | pg
diff -b /ca/bin/rp /cb/bin/rp
diff -c afile maybe_the_same_file
diff -v rf.r nrf.r | sp
diff -r old_manual.fmt new_manual.fmt | fmt
diff -s old new >>update_old_to_new
```

Messages

"<file>: can't open" if either <new_file> or <old_file> is not readable.
"Usage: diff ..." for illegal options.

Bugs

The algorithm used has one quirk: a line or a block of lines which is not unique within a file will be labeled as an insertion (deletion) if its immediately adjacent neighbors both above and below are labeled as insertions (deletions).

Fails on very large files (> 10000 lines) when using the "-d" option.

See Also

common (1),
Heckel, P., "A Technique for Isolating Differences Between Files", *Communications of the ACM*, vol 21, no 4 (April 1978), 264-268.

Usage

dnum [<disk_number>]

Description

If given a disk number as an argument, 'dnum' will print a short description of the corresponding disk partition (controller type, number of heads, first head number, etc.). If the argument is missing, 'dnum' will prompt the user for the required information and generate the corresponding disk number.

Examples

```
dnum 21060
Controller 4004 storage module disk
   controller 0, unit 0
   first head: 4, number of heads: 4
```

Messages

Many; 'dnum' is interactive.

Bugs

When given a cartridge module disk number as an argument, 'dnum' always considers it a storage module and gives incorrect results for the fixed surfaces.

drop (1) --- drop characters from a string (APL-style) 03/20/80

Usage

drop (<length> | -<length>) <string>

Description

'Drop' performs the function of the APL dyadic drop operator. The absolute value of the first argument is the number of characters to be dropped. If the number is positive, they are dropped from the front of the string; if negative, they are dropped from the end of the string. The result is printed on standard output one.

If more characters are dropped than exist in the string, a null string is printed.

Examples

```
drop 2 [filename]
cat [drop -2 source]
```

See Also

take (1), substr (1), stake (2), sdrop (2), substr (2)

Usage

```
dump { ls | linked_string | sv | shell_variable
     | fd [ <num> ] | file_desc [ <num> ]
     | cm | swt_common }
```

Description

'Dump' is intended to print a semi-readable dump of the various Subsystem data areas. It dumps any of four different data areas, based on its arguments. Following are descriptions of the four different dumps.

"Ls" or "linked_string" prints the command interpreter's linked string storage space in a readable format. This option produces a symbolic listing consisting of a series of entries of the form

```
address -> item
```

where "address" represents an index into the linked string storage space, and "item" is either (1) a quoted string, representing the characters in memory at the given address (e.g. "peruse"); (2) the word LSNULL followed by a size in parentheses, indicating available space in the storage area (e.g. LSNULL (1600)); or (3) the pointer "->" followed by an address, representing a pointer or link to another place in the storage area.

"Sv" or "shell_variable" dumps the contents of the hash table that stores shell variables and their contents. For each active lexic level (currently active command file), it produces a symbolic dump of the five hashed lists used for variable storage. Each item in each list consists of a variable name followed by an equals sign (=) and the variable's value. Both name and value are followed by indexes into the linked string storage area.

"Fd" or "file_desc" dumps the Subsystem file descriptor <num>. If <num> is missing, all open file descriptors are dumped.

"Cm" or "swt_common" dumps the remaining Subsystem common areas.

Examples

```
dump ls sv
dump fd 3 fd 5
dump cm fd
```

See Also

shtrace (1), dumpls (2), dumpsv (2), dmpcm$ (6), dmpfd$ (6), linked string routines ('ls?*' (4))

Usage

```
e  [ <filename> { <se options> } ]
```

Description

'E' is a shell file used to invoke the proper editor ('ed' or 'se') for a given terminal. In addition, 'e' will remember the name of the last file edited, and reuse that name if none is specified on the command line.

'E' will make use of the variables 'f' and 'se_params' if the user has declared them at the terminal level. 'F' is used to store the name of the last file edited; consequently, the user need only type the command "f" to have that name printed. 'Se_params' is used to store personally-preferred instructions for the initialization of 'se', such as the choice between absolute and relative line numbers, case mapping, etc. It may be any sequence of legal 'se' arguments, separated by blanks. Furthermore, additional screen editor options may be selected by including them on the 'e' command line after the file name argument. If either 'f' or 'se_params' is not declared at lexic level 0, 'e' will assume that they are empty.

'E' selects the proper editor to invoke by calling the 'term_type' command to see if the screen editor supports the current terminal type.

Examples

```
e time_sheet
e
```

Messages

None from 'e', but several may result from the editors or the shell.

Bugs

Since 'se' knows about users' terminal types, and since 'se' now reads personal commands in =home=/.serc, this command is pretty much obsolete.

See Also

ed (1), se (1), if (1), term_type (1)

Usage

echo { <arbitrary string> }

Description

'Echo' simply prints its arguments, separated by blanks, on standard output. It is frequently used to make announcements from shell programs or to quickly produce very short data files. If no arguments are specified, 'echo' produces no output whatsoever.

Before printing them, 'echo' scans its arguments for "@t" and "@n" character sequences and converts them to tabs and newlines, respectively.

Examples

```
echo "split@nthis@nline"
echo "Good morning"
```

See Also

error (1)

Usage

ed [<pathname>]

Description

'Ed' is the Subsystem version of the *Software Tools* text editor 'edit'. This entry contains a short summary of the editor's commands and allowable pattern elements; for a full description, along with a tutorial, see the *Introduction to the Software Tools Text Editor*.

Note that the commands accepted by 'ed' are also accepted by the Subsystem screen editor 'se'.

Commands to 'ed' consist of zero or more "line number expressions" followed by a single-character mnemonic, possibly followed by additional parameters. The following table outlines the allowable line number expression syntax:

Elements of Line Number Expressions

Form	*Value*
integer	value of the integer (e.g., 44).
.	number of the current line in the buffer.
$	number of the last line in the buffer.
^	number of the previous line in the buffer (same as .-1).
-	number of the previous line in the buffer (same as ^).
/pattern[/]	number of the next line in the buffer that matches the given pattern (e.g., /February/); the search proceeds to the end of the buffer, then wraps around to the beginning and back to the current line. The trailing "/" is optional.
\pattern[\]	number of the previous line in the buffer that matches the given pattern (e.g., \January\); search proceeds in reverse, from the current line to line 1, then from the last line back to the current line. The trailing "\" is optional.
>name	number of the next line having the given markname (search wraps around, like //).

<name	number of the previous line having the given markname (search proceeds in reverse, like \\).
expression	any of the above operands may be combined with plus or minus signs to produce a line number expression. Plus signs may be omitted if desired (e.g., /parse/-5, /lexical/+2, /lexical/2, $-5, .+6, .6).

The text patterns used in line number expressions (and in global commands and the substitute command, discussed below) take the form of limited regular expressions. Each such regular expression is composed of a sequence of ordinary characters and special metacharacters, called "pattern elements." The following table outlines the allowable pattern elements.

Summary of Pattern Elements

Element	*Meaning*
%	Matches the null string at the beginning of a line. However, if not the *first* element of a pattern, is treated as a literal percent sign.
?	Matches any single character other than newline.
$	Matches the newline character at the end of a line. However, if not the *last* element of a pattern, is treated as a literal dollar sign.
[<ccl>]	Matches any single character that is a member of the set specified by <ccl>. <Ccl> may be composed of single characters or of character ranges of the form <c1>-<c2>. If character ranges are used, <c1> and <c2> must both belong to the digits, the upper case alphabet or the lower case alphabet.
[~<ccl>]	Matches any single character that is *not* a member of the set specified by <ccl>.
*	In combination with the immediately preceding pattern element, matches zero or more characters that are matched by that element.
@	Turns off the special meaning of the immediately following character. If that character has no special meaning, this is treated as a literal "@".
{<pattern>}	Tags the text actually matched by the sub-pattern specified by <pattern> for use in the replacement part of a substitute command.
&	Appearing in the replacement part of a substitute command, represents the text actually matched by the pattern part of the command. If "&" is the only character in the replacement part, however, then it represents the replacement part used in a previous substitute command.
@<digit>	Appearing in the replacement part of a substitute command, represents the text actually matched by the tagged sub-pattern specified by <digit>.

Finally, the following table lists the commands that may be used for the actual creation and modification of text:

Editor Command Summary

Range	*Syntax*	*Function*
.	a[:text]	Append Inserts text after the specified line. Text is inserted until a line containing only a period and a newline is encountered. In 'se', if the command is followed immediately by a colon, then whatever text follows the colon is inserted without entering "append" mode. The current line pointer is left at the last line inserted.
.,.	c[:text]	Change Deletes the lines specified and inserts text to replace them. Text is inserted until a line containing only a period and a newline is encountered. In 'se', if the command is followed immediately by a colon, then whatever text follows the colon is inserted without entering "append" mode. The current line pointer is left at the last line inserted.
.,.	d[p]	Delete Deletes all lines between the specified lines, inclusive. The current line pointer is left at the line after the last one deleted. If the "p" is included, the new current line is printed.
none	e[!] [filename]	Enter Loads the specified file into the buffer and prepares for editing. Automatically invoked if a filename is specified as an argument on the command line used to invoke the editor. The current line pointer is positioned at the first line in the buffer. An error message is generated if the editing buffer contains text that has not been saved. The enter command may be resubmitted after the error message, in which case it will be obeyed. The "enter now" command "e!" may be used to avoid the error message.

none	f [filename]	File Print or change the remembered file name. If a name is given, the remembered file name is set to that value; otherwise, the remembered file name is printed.
.,$	g/pat/command	Global on pattern Performs the given command on all lines in the specified range that match a certain pattern.
none	h[stuff]	Help In 'se', provides access to online documentation on the screen editor. "Stuff" may be used to select which information is displayed.
.	i[:text]	Insert Inserts text before the specified line. Text is inserted until a line containing only a period and a newline is encountered. In 'se', if the command is immediately followed by a colon, then whatever text follows is inserted without entering "append" mode. The current line pointer is left at the last line inserted.
^,.	j[/stuff[/]][p]	Join The specified lines are joined into a single line. You may specify in "stuff" what is to replace the newlines that previously separated the lines. The default is a single blank. If you use the default, 'ed' automatically prints out the result. If the "p" option is used, the resulting line (which becomes the new current line) is printed. Thus "j" and "jp" are equivalent to "j/ /p". In general, 'ed' and 'se' will supply trailing delimiters for you. So "j/" is the same as "j//", i.e. replace the newline(s) with nothing (delete them).
.,.	km	marK The specified lines are marked with 'm' which may be any single character other than a newline. If 'm' is not present, the lines are marked with the default name of blank. The current line pointer is never changed.

none	l	Locate "l" will print the first line of the file =installation=. This is so that one can tell what machine he is using from within the editor. This is particularly useful for installations with many machines that can run the editor, where the user can switch back and forth between them, and become confused as to where he is at a given moment.
.,.	m<line>[p]	Move Moves the specified block of lines after <line>. <Line> may not be omitted. The current line pointer is left at the last line moved. If the "p" is specified, the new current line is also printed.
.,.	n[m]	Name If 'm' is present, the last line in the specified range is marked with it and all other lines having that mark name are given the default mark name of blank. In 'ed', if 'm' is not present, the mark name of each line in the range is printed; in 'se' the names of all lines in the range are cleared.
none	o[stuff]	Option Editing options may be queried or set. "Stuff" determines which options are affected. In 'ed', options "d", "g", "k", and "p" are available. See below for a full discussion of what the options do.
.,.	p	Print Prints all the lines in the given range. In 'se', as much as possible of the range is displayed, always including the last line; if no range is given, the previous page is displayed. The current line pointer is left at the last line printed.
none	q[!]	Quit Exit from the editor. An error message is generated if the editing buffer contains text that has not been saved. The quit command may be resubmitted after the error message, in which case it will be obeyed. The "quit now" command "q!" may be

used to avoid the error message.

. r [filename] Read
Insert the contents of the given file after the specified line. The current line pointer is left at the last line read.

.,. s[/pat/sub[/][g][p]] Substitute
Substitutes "sub" for each occurrence of the pattern "pat". If the optional "g" is specified, all occurrences in each line are changed; otherwise, only the first occurrence is changed. The current line pointer is left at the last line in the range in which a substitution was made. This line is also printed if the "p" is used. In 'ed', if you leave off the trailing slash, the result of the substitute will be printed automatically. Thus "s/junk/stuff" is entirely equivalent to "s/junk/stuff/p". If you type an "s" by itself, without a pattern and replacement string, 'ed' will behave as though you had typed "s//&/p", i.e. substitute the previous replacement pattern for the previous search pattern, and print.

.,. t[/from/to[/][p]] Transliterate
The range of characters specified by 'from' is transliterated into the range of characters specified by 'to'. The last line on which something was transliterated is printed if the "p" option is used. The last line in the range becomes the new current line. Again, if you leave off the trailing delimiter, 'ed' will print the result of the transliteration. In addition, like the "s" command, both the 'from' and 'to' parts are saved; "t//&/" will perform the same transliteration as the last one, and "t" is the same as "t//&/". The "&" is special if it is the only character in the 'to' part, otherwise it is treated as a literal "&". In Unix mode (for 'se' only), use "%" instead of "&". See *Software Tools* and the help on 'tlit' for some examples of character transliterations.

.	u[d][p]	Undo The specified range of lines is replaced by the last range of lines deleted. If the "d" is used, the restored text is inserted after the last line in the specified range. The current line pointer is set at the last line that was restored; this line is also printed if the "p" is specified.
.,.	v	oVerlay In 'ed', each line in the given range is printed without its terminating newline and a line of input is read and added to the end of the line. If the first and only character on the input line is a period, no further lines are printed. In 'se', "overlay mode" is entered and the control characters may be used to modify text anywhere in the buffer. A control-v may be used to quit overlay mode. A control-f may be used to restore the current line to its original state and terminate the command.
1,$	w['+'|'!'] [filename]	Write Writes the portion of the buffer specified to the named file. The current line pointer is not changed. If "+" is given, the portion of the buffer is appended to the file; otherwise the portion of the buffer replaces the file. In 'se' only, if "!" is present, an existing file specified in the command is over-written without comment. If "filename" is not present, the specified lines will be written to the current file name specified on the status line.
1,$	x/pat/command	eXclude on pattern Performs the command on all lines in the given range that do not match the specified pattern.
.,.	y<line>[p]	copY Makes a copy of all the lines in the given range, and inserts the copies after <line>. As with the "m" command, <line> may not be omitted. The current line pointer is set to the new copy of the last line in the

range; this line is printed if the "p" is present.

.,. zb<left>[,<right>][<char>] draw Box
In 'se' only, a box is drawn using the given <char> (blank by default, allowing erasure of a previously-drawn box). Line numbers are used to specify top and bottom row positions of the box. <Left> and <right> specify left and right column positions of the box. If second line number is omitted, the box degenerates to a horizontal line. If right-hand column is omitted, the box degenerates to a vertical line.

. =[p] Equals
The number of the specified line is printed. The line itself is also printed if the "p" option is used. The current line pointer is not changed.

none ? Query
In 'ed' only, a verbose description of the last error encountered is printed.

1,$!mcommand Exclude on markname
Similar to the 'x' prefix except that 'command' is performed for all lines in the range that do not have the mark name 'm'.

1,$ 'mcommand Global on markname
Similar to the 'g' prefix except that 'command' is performed for all lines in the range that have the mark name 'm'.

. : Print next page
In 'ed', 23 lines beginning with the current line are printed (equivalent to ".,.+23p"). In 'se', the next page of the buffer is displayed and the current line pointer is placed at the top of the window.

. ~[<Software Tools Command>] Escape to the shell
If present, the <Software Tools Command> is passed to the shell to be executed. Otherwise, an interactive shell is created. After either the command or the shell exits, 'ed'

prints a "~" to indicate that the shell escape has completed. If the first character of the <Software Tools Command> is a "!", then the "!" is replaced with the text of the previous shell command. An unescaped "%" in the <Software Tools Command> will be replaced with the current saved file name. If the shell command is expanded, both 'ed' and 'se' will echo it first, and then execute it.

Until EPFs are supported, when using 'ed', do not use the shell to execute external commands. Internal commands (like 'cd') are OK. This does not apply to 'se'.

For a deeper discussion of using the shell from within a program, see the help on the 'shell' subroutine.

The values associated with editor options should immediately follow their respective key letters, without intervening blanks between the option letter and the option value. The options are as follows:

Option	*Action*
d[<dir>]	selects the placement of the current line pointer following a "d" (delete) command. <dir> must be either ">" or "<". If ">" is specified, the default behavior is selected: the line following the deleted lines becomes the new current line. If "<" is specified, the line immediately preceding the deleted lines becomes the new current line. If neither is specified, the current value of <dir> is displayed.
g	controls the behavior of the "s" (substitute) command when it is under the control of a "g" (global) command. By default, if a substitute inside a global command fails, 'ed' will not continue with the rest of the lines which might succeed. If "og" is given, then the global substitute will continue, and lines which failed will not be affected. Successive "og" commands will toggle this behavior. An explanatory message is written to the terminal.
k	Indicates whether the current contents of your edit buffer has been saved or not by printing either a "saved" or "not saved" message on your terminal.

p[/string[/]] sets the prompt to be used (useful for the user who is disturbed by 'ed's quiet behavior). The prompt can be set by the command "op/string[/]", which sets the prompt to "string". The trailing delimiter is optional. If no string is given, the prompt is set to "* ". An empty string ("op//") restores 'ed's no prompting behavior. Successive "op" commands will toggle prompting mode. The "op" option has an entirely different meaning in 'se'; see the help on 'se' for details.

Examples

```
ed
ed file
ed_input> ed file
```

Files

```
=temp=/ed?* for scratch file
=temp=/script?* for checkpoint file
=home=/ed.logout for saving the buffer on a LOGOUT$ condition
```

Messages

```
"fatal scratch file read error"
"fatal scratch file write error"
"fatal scratch file seek error"
"can't open scratch file"
"?" for miscellaneous errors
```

See Also

se (1), *Software Tools*, *Introduction to the Software Tools Text Editor*

Usage

ek [<erase character> <kill character>]

Description

'Ek' allows the user to select his erase (character delete) and kill (line delete) characters. If 'ek' is called with two arguments, the erase and kill characters are set. Each of the arguments may be a single character, or an ASCII mnemonic for an unprintable character. If 'ek' is called without arguments, the current erase and kill characters are printed.

Erase and kill characters are part of the user's permanent profile, and so will be remembered from session to session.

Examples

```
ek 1 2
ek BS DEL
ek
```

Messages

"Usage: ek ..." for improper number of arguments

Bugs

"ek e k" will produce extremely undesirable effects.

See Also

term (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

```
if <value>
   then
      { <command> }
   elif <value>
      { <command> }
fi
```

Description

'Elif' is used as a short form of the 'else' statement with an 'if' statement. It does not cause another nesting level of control statements and is useful in implementing case-like structures. Unfortunately, <value> must be a constant in the 'elif' statement where an else-if pair allows <value> to be a function call. This severely limits its usefulness since the value will be known at the time it is used.

Examples

```
if [eval [line] = 10]
   then
      set term = consul
elif 7                  # always will execute, so same as an else
   then
      set term = regent
fi
```

Messages

"Missing 'fi'" if end-of-file is seen before a 'fi' is encountered.

Bugs

Redirectors placed before the 'fi' will prevent 'else' from detecting it.

Some might consider it a bug that 'elif' takes a constant, instead of being able to use the result of a function call.

See Also

if (1), then (1), elif (1), fi (1), case (1)

Usage

```
if <value>
   then
      { <command> }
   else
      { <command> }
fi
```

Description

'Else' is used in conjunction with the 'if' command to introduce the negative portion of a conditional statement. Paradoxically, it is executed only as control falls through from the then-part of the conditional; its action is to skip to the first unmatched 'fi' command.

The else-clause of a conditional is always optional.

Since 'else' works as well from the terminal as it does from a command file, typing "else" as a command will cause the command interpreter to skip input until it sees a 'fi' command or end-of-file.

Examples

```
if [eval [line] = 10]
   then
      set term = consul
   else
      set term = unknown
fi


if [eval [take 2 [time]] ">" [deadline]]
   then
      echo "Time out."
   else
      process_job
fi
```

Messages

"Missing 'fi'" if end-of-file is seen before a 'fi' is encountered.

Bugs

Redirectors placed before the 'fi' will prevent 'else' from detecting it.

See Also

if (1), then (1), fi (1), case (1)

Usage

```
entab { -t <tab character> |
        <column number>    |
        +<increment> }
```

Description

'Entab' converts multiple blanks on its first standard input into an equivalent number of blanks and tab characters on its first standard output. The tab character may be specified as an argument with the "-t <tab character>" argument sequence; otherwise, the ASCII TAB (ctrl-i) is used.

Any number of tab stops may be set by specifying the desired column numbers as arguments. If the +<increment> construct is used, stops will be set at intervals of <increment> columns, starting with the most recently set stop. Thus, the argument sequence

```
10 +5
```

would set stops in columns 10, 15, 20, etc. In the absence of any other specification, default stops are set in every fourth column, starting with column five.

Examples

```
prog.f> entab 7 >compressed_prog.f
term_paper> entab -t \
```

Messages

"Usage: entab ..." for incorrect arguments.

See Also

detab (1)

Usage

error [-<error_code>] { <arbitrary_string> }

Description

'Error' is used in shell programs to announce the presence of an error condition and return an error code. The option argument specifies the error code returned; the default is 1000 (identical to the value returned by the subprogram 'error'). The arguments specified are written to ERROUT, separated by spaces and terminated by a NEWLINE.

Examples

error File not found.
error -1 "Attention: your program has just been destroyed"

Bugs

Probably should understand escape sequences, like 'echo'.

See Also

echo (1), error (2), seterr (2)

Usage

```
case <value>
   when <alternative1>
      { <command> }
   when <alternative2>
      { <command> }
   ...
   out
      { <command> }
esac
```

Description

'Esac' is a do-nothing command used to mark the end of a case statement. It may be searched for by the 'case', 'when', or 'out' commands. Every 'case' command must be followed by a matching 'esac' command.

'Esac' may be used to regain control of a terminal after execution of a 'when' or 'out' command.

Examples

```
case [time]
   when 12:00:00
      echo "LUNCHTIME!!!"
   when 17:00:00
      echo "t i m e  t o  g o  h o m e . . ."
   out
      echo "Back to Work."
esac
```

Bugs

Redirectors before 'esac' prevent its recognition by 'when', 'out', and 'case'.

See Also

case (1), when (1), out (1), if (1)

Usage

eval { <expression_element> }

Description

'Eval' is used to evaluate arithmetic expressions involving 32-bit integers and shell variables. The expression to be evaluated is given in the arguments, and may be spread out over as many arguments as desired. The value of the expression is printed on standard output one.

'Eval' supports the following operators:

+	addition
-	subtraction and unary minus
*	multiplication
/	division
%	modulus
<<	logical left shift
>>	logical right shift
**	exponentiation
<	less than (returns 1 or 0)
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
~=	not equal to
&	bitwise logical and
|	bitwise logical or
~	bitwise logical complement

Operator priority, from highest to lowest, is as follows:

```
-   (unary minus)
**
*       /     %     <<    >>
+       -
<       <=    =     ~=    >=    >
~
&
|
```

Parentheses may be freely used to specify the desired order of evaluation.

Shell variables may appear in expressions; their values will be substituted when necessary. As always, shell function calls may be included as part of the command line, and will be processed before 'eval' sees the expression.

Care should be taken in using characters recognized by the shell as meta-characters (e.g. parentheses, vertical bar, greater than sign). For this reason, it is probably wise to enclose the expression in quotes.

Examples

```
eval 10 - 14 + 37**2
set i = [eval i + 1]
cat file_stack[eval sp-1]
```

Messages

"Bad element in expression" for missing or unrecognizable expression element.
"<var>: domain error" for reference to non-numeric shell variable.
"<var>: value error" for reference to undefined shell variable.

See Also

cmp (1), declare (1), forget (1), set (1), hp (1)

Usage

exit [<levels>]

Description

'Exit' causes execution of one or more currently active command files to cease. It is somewhat similar to the PL/I "return" statement in that it simulates a normal termination of at least one environment (scope).

When invoked, 'exit' positions the <levels> most recently activated command files to end-of-file and dumps the command interpreter's internal command buffer. Thus, when the command interpreter next attempts to fetch a command, it sees <levels> successive ends-of-file and cleans up the associated environments. If <levels> is omitted, only one level is terminated.

'Exit' is most often used to terminate a command file when some error condition defined by the user occurs (for example, an attempt to use the command file by an unauthorized user).

Examples

```
if 1
   echo "Sorry, you are not allowed to use this program."
   exit
fi
```

Bugs

May behave irrationally if <levels> is too large.

See Also

error (1), if (1), sh (1), goto (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

```
f77c {-<option>[<level>]} <input file>
         [-b [<binary file>]]
         [-l [<listing file>]]
         [-z <F77 option>]
   <option> ::= c | d | e | f | g | h | i | k | m |
                o | q | r | s | t | u | v | w | x
```

Description

'F77c' serves as the Subsystem interface to the Primos Fortran 77 compiler (F77). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos F77 command and causes it to be executed.

Options

The general structure of an 'f77c' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-c(0) [0..1] - Case.

Level 0 forces case to be insignificant in identifiers. Upper case identifiers are considered the same as lower case identifiers.

Level 1 cause case to significant in identifiers. Upper case identifiers are considered different from lower case identifiers.

-d(0) [0..2] - Debugging control.

Level 0 prevents all debugging information from being included in the generated code. A program so compiled may not be used with the source level debugger.

Level 1 allows limited debugging information to be included in the generated code, but does not interfere with optimization.

Level 2 causes complete debugging information to be included in the generated code and inhibits optimization. This option cannot be used with full

optimization (-o3).

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(0) [0..1] - Offset map.

Level 0 inhibits the generation of a storage offset map.

Level 1 cause the generation of a map listing the storage offset of each program variable.

-g(0) [0..1] - Logical precision.

Level 0 causes the compiler to allocate 16 bits (1 word) for each logical variable or constant.

Level 1 causes the compiler to allocate 32 bits (2 words) for each logical variable or constant.

-h(0) [0..1] - Huge (multi-segment) arrays.

Level 0 insures that dummy arrays and array parameters will not be treated as multi-segment arrays.

Level 1 causes references to dummy arrays and array parameters to generate code that will work even if the arrays are larger than one segment (64K words) in length. This option is allowed only when the "-m" option is at level 2.

-i(0) [0..1] - Integer precision.

Level 0 causes the compiler to assign 16 bits (1 word) to each integer variable or constant.

Level 1 causes the compiler to assign 32 bits (2 words) to each integer variable or constant.

-k(0) [0..1] - Compilation statistics.

Level 0 inhibits the display of compilation statistics on the terminal.

Level 1 causes the display of compilation statistics on the terminal.

-m(2) [2..3] - Addressing mode.

Level 2 implies 64V addressing mode. At present this is the only addressing mode fully supported under the Subsystem.

Level 3 implies 32I addressing mode. Code in this addressing mode will not execute on a Prime 400.

-o(2) [0..3] - Optimization control.

Level 0 turns off all optimizations.

Level 1 turns on pattern replacement optimizations.

Level 2 turns on pattern replacement and strength reduction optimizations.

Level 3 turns on all optimizations (pattern replacement, strength reduction, and removal of invariants in DO-loops). This option cannot be used with full debugging (-d2).

-q(1) [0..1] - Suppress warning messages.

Level 0 inhibits the display of compiler warning messages.

Level 1 allows the display of compiler warning messages.

-r(0) [0..1] - Range checking.

Level 0 inhibits run-time checking of subscripts and substrings.

Level 1 causes the compiler to insert code for the run-time checking of subscripts and substrings.

-s(1) [0..1] - Storage allocation control.

Level 0 requires that all subprogram variables be allocated statically (the usual case for implementations of Fortran, although not required by the standard).

Level 1 requires that all subprogram variables not named in SAVE declarations or DATA statements be allocated dynamically on the run-time stack. This permits recursion and more efficient use of memory, and is the normal mode of usage under the Subsystem.

-t(0) [0..1] - DO-loop trip control.

Level 0 causes all DO loops to be of the zero or more iteration (Fortran 77 standard) type.

Level 1 causes all DO loops to be of the one or more iteration (Fortran 66 standard) type.

-u(1) [0..1] - Undeclared variable checking.

Level 0 prevents the compiler from flagging undeclared variables as errors.

Level 1 causes the compiler to report undeclared variables as errors. This enforces (one hopes) better programming practices and reduces the number of hard-to-find semantic errors in programs.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-w(0) [0..1] - Generate floating round instructions.

Level 0 does not generate floating round (FRN) instructions.

Level 1 cause a floating round (FRN) instruction to be generated before every floating store (FST) instruction in the code produced by the F77 compiler. This option improves the accuracy of single precision floating point calculations at some slight run-time performance expense.

-x(1) [0..2] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing containing all variables referenced in executable statements and omitting those that are declared but never referenced.

Level 2 causes the compiler to generate a full cross-reference of all variables.

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.f77", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.f77", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".f77", ".f" or ".df") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.f77" is

```
f77c -c0d0e1f0g0h0i0k0m2o2q1r0s1t0u1v1w1x1 _
     file.f77  -b file.b  -l /dev/null
```

which corresponds to the F77 command

```
f77 -i *>file.f77 -b *>file.b -l no -ints -logs
```

Examples

```
f77c file.f77
f77c -ig dmach.f77
f77c -x dmach.f77 -b b_dmach -l l_dmach
f77c -m3 i_mode_prog.f77 -z"-newopt"
```

Messages

"Usage: f77c ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to

<upper bound>" if an out-of-range level number is specified.

"missing input file name" if no input filename could be found.

"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.

"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.

"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'f77c' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'F77c' pays no attention to standard ports.

See Also

f77cl (1), ld (1), init$f (2), bind (3)

Usage

f77cl <program name> [<'ld' options>] [/ <'f77c' options>]

Description

'F77cl' is a shell file that invokes the Primos Fortran 77 and the Primos segmented loader. If 'f77cl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".f77", although in <program name> it may be specified with or without the ending ".f77". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'f77c' will be called with the <'f77c' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
f77cl myprog.f77
f77cl myprog subs.b subs2.b -l mylib
f77cl myprog / -ok -l mylist
```

Messages

"<program name>.f77: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

f77c (1), ld (1), init$f (2), bind (3)

Usage

```
fc {-<option>[<level>]} <input file>
        [-b [<binary file>]]
        [-l [<listing file>]]
        [-z <FTN option>]
   <option> ::= d | e | f | h | i | k | m | o | p | q |
                r | s | t | u | v | w | x
```

Description

'Fc' serves as the Subsystem interface to the Primos Fortran 66 compiler (FTN). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos FTN command and causes it to be executed.

Options

The general structure of an 'fc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-d(0) [0..2] - Debugging control.

Level 0 prevents all debugging information from being included in the generated code. A program so compiled may not be used with the source level debugger.

Level 1 allows limited debugging information to be included in the generated code, but does not interfere with optimization.

Level 2 causes complete debugging information to be included in the generated code and inhibits optimization. (Cannot be used when the "-o" option is specified with a level greater than zero.)

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(1) [0..1] - Floating point instruction restriction.

Level 0 causes the compiler to avoid generating certain types of floating point instructions that are not available on all Prime machines. This level is allowed only when the "-m" option is at levels 0 or 1.

Level 1 allows the compiler to use the entire floating point instruction set.

-h(0) [0..1] - Huge (multi-segment) arrays.

Level 0 insures that dummy arrays and array parameters will not be treated as multi-segment arrays.

Level 1 causes references to dummy arrays and array parameters to generate code that will work even if the arrays are larger than one segment (64K words) in length. This option is allowed only when the "-m" option is at level 2.

-i(0) [0..1] - Integer precision.

Level 0 causes objects declared to be of type "integer" to be assigned to 16 bit storage locations.

Level 1 causes objects declared to be of type "integer" and all integer constants to be assigned to 32 bit storage locations. This is occasionally useful in transporting Fortran code written on or for other systems. Beware of interactions with Primos and Subsystem support routines which normally require 16-bit parameters.

-k(0) [0..1] - Compilation statistics.

Level 0 inhibits the display of compilation statistics on the terminal.

Level 1 causes the display of compilation statistics on the terminal.

-m(2) [0..2] - Addressing mode.

Level 0 implies 32R addressing mode. Large arrays ("-h1"), dynamic storage allocation ("-s1"), and debugging ("-d1" or "-d2") may not be used in this mode.

Level 1 implies 64R addressing mode. Large arrays ("-h1"), dynamic storage allocation ("-s1"), and debugging ("-d1" or "-d2") may not be used in this mode.

Level 2 implies 64V addressing mode. At present this is the only addressing mode fully supported under the Subsystem.

-o(1) [0..2] - Optimization control.

Level 0 turns off all optimizations.

Level 1 turns on "safe" optimizations (strength reduction and removal of invariants in DO-loops). This option cannot be used with full debugging ("-d2").

Level 2 turns on "unsafe" optimizations (same as level 1, but applied even to loops that may make use of the Fortran extended-range feature). This option cannot be used with full debugging ("-d2"). Note: Ratfor generates GOTO instructions to implement its internal procedures. If a Ratfor internal procedure is called from within a DO loop, this option *cannot* be used for the program.

-p(0) [0..1] - Entry control block allocation.

Level 0 disallows mixing of procedures and entry control blocks.

Level 1 allows mixing of procedures and entry control blocks. Selection of this option is valid only when the "-m" option is at level 2.

-q(0) [0..1] - Quirk control.

Level 0 disallows the use of certain statements and declarations designed for use by systems programmers.

Level 1 allows the use of these statements and declarations. A side effect of selecting this level is that the compiler flags undeclared variables, regardless of the level of the "-u" option.

-r(0) [0..1] - Instruction reach control.

Level 0 causes the compiler to generate short instructions for all variable references.

Level 1 causes the compiler to generate long-reach instructions for all variable references. This option is valid only when the "-m" option is at level 0 or 1.

-s(1) [0..1] - Storage allocation control.

Level 0 requires that all subprogram variables be allocated statically (the usual case for implementations of Fortran, although not required by the standard).

Level 1 requires that all subprogram variables not named in SAVE declarations or DATA statements be allocated dynamically on the run-time stack. This permits recursion and more efficient use of memory, and

is the normal mode of usage under the Subsystem. Dynamic allocation cannot be used unless the addressing mode is 64V (-m2).

-t(0) [0..1] - Run-time trace control.

Level 0 causes no run-time trace code to be emitted.

Level 1 causes the compiler to emit trace code that prints statement numbers when they are encountered and records assignments to variables. Warning: this option can produce reams of output!

-u(1) [0..1] - Undeclared variable checking.

Level 0 prevents the compiler from flagging undeclared variables as errors.

Level 1 causes the compiler to report undeclared variables as errors. This enforces (one hopes) better programming practices and reduces the number of hard-to-find semantic errors in programs.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-w(0) [0..1] - Generate floating round instructions.

Level 0 does not generate floating round (FRN) instructions.

Level 1 cause a floating round (FRN) instruction to be generated before every floating store (FST) instruction in the code produced by the FTN compiler. This option improves the accuracy of single precision floating point calculations at some slight run-time performance expense.

-x(1) [0..2] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing containing all variables referenced in executable statements and omitting those that are declared but never referenced.

Level 2 causes the compiler to generate a full cross-reference of all variables.

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.f", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.f", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".f", ".ftn", or ".df") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.f" is

```
fc -d0e1f1h0i0k0m2o1p0q0r0s1t0u1v1w0x1 _
     file.f  -b file.b  -l /dev/null
```

which corresponds to the FTN command

```
ftn -i *>file.f -b *>file.b -l no -64v -dcl -dynm -opt
```

Examples

```
fc file.f
fc -i -u0 dmach.f
fc -x dmach.f -b b_dmach -l l_dmach
fc -m1 r_mode_prog.f -z"-debase -nofp"
```

Messages

"Usage: fc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'fc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Fc' pays no attention to standard ports.

See Also

fcl (1), ld (1), rfl (1), init$f (2), bind (3)

Usage

fcl <program name> [<'ld' options>] [/ <'fc' options>]

Description

'Fcl' is a shell program that invokes the Primos Fortran 66 compiler and the Primos segmented loader. If 'fcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".f", although in <program name> it may be specified with or without the ending ".f". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'fc' will be called with the <'fc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
fcl myprog.f
fcl myprog subs.b subs2.b -l mylib
fcl myprog / -o -l mylist
```

Messages

"<program name>.f: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

fc (1), ld (1), init$f (2), bind (3)

Usage

```
fdmlc <input file>
          [-b [<output file>]]
          [-l [<listing file>]]
          [-z <FDML option>]
```

Description

'Fdmlc' serves as the Subsystem interface to the Prime DBMS Fortran DML preprocessor (FDML). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and output files as needed, and then produces a Primos FDML command and causes it to be executed.

The "-b" option is used to select the name of the file to receive the output Fortran code from the preprocessor. If a file name follows the option, then that file receives the output. If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".df". For example, if the input filename is "prog.f", the output file will be "prog.df"; if the input filename is "foo", the output file will be "foo.df".

The "-l" option is used to select the name of the file to receive the listing generated by the preprocessor. If a file name follows the option, then that file receives the listing. If the "-l" option is specified without a file name following it or is not specified, a default filename is constructed from the input filename by changing its suffix to ".dl". For example, if the input filename is "gonzo", the listing file will be "gonzo.dl"; if the input filename is "bar", the listing file will be "bar.dl".

The input filename must be a disk file name (conventionally ending in ".f", ".f77", or ".ftn").

In summary, then, the default command line for compiling a file named "file.f" is

```
fdmlc file.f -b file.df -l file.dl
```

which corresponds to the FDML command

```
fdml -i *>file.f -b *>file.df -l *>file.dl
```

Examples

```
fdmlc file.f
fdmlc payroll.f -b b_payroll -l l_payroll
fdmlc funnyprog.f -z"-newopt"
```

Messages

"Usage: fdmlc ..." for invalid option syntax.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to output on the terminal or line printer spooler.
"Sorry, the listing file must be a disk file" if the listing file was directed to a device file.

Bugs

'Fdmlc' pays no attention to standard ports.

There is no way to avoid getting both a listing and output file.

See Also

ddlc (1), f77c (1), fc (1), fsubc (1), ld (1), bind (3)

Usage

fdmlcl <program name> [<'ld' options>] [/ <'fc' options>]

Description

'Fdmlcl' is a shell file that invokes the Prime DBMS Fortran DML preprocessor, the Primos Fortran 66 compiler and the Primos segmented loader. If 'fdmlcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".f", although in <program name> it may be specified with or without the ending ".f". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'fc' will be called with the <'fc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
fdmlcl myprog.f
fdmlcl myprog subs.b subs2.b -l mylib
fdmlcl myprog / -ok -l mylist
```

Messages

"<program name>.f: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

fc (1), ld (1), bind (3)

Usage

```
fdmp  { -<opt>{<opt>}   |
        +<start>        |
        -<end> }  [ <pathname> ]
   <opt> ::= b | c | d | h | o
```

Description

'Fdmp' writes on standard output a dump of the named file (standard input if the file name is omitted) in one or more of five formats as specified by the <opt> arguments. The formats are:

-b The file is interpreted as a sequence of octal bytes.

-c The file is interpreted as a sequence of ASCII characters. Non-printable characters are represented by a control sequence consisting of a caret ("^") followed by the corresponding printable character. Thus, an ETX (ctrl-c) would be represented by "^c". The single exception is DEL which is represented as "^ ".

-d The file is interpreted as a sequence of signed decimal integers.

-h The file is interpreted as a sequence of hexadecimal integers.

-o The file is interpreted as a sequence of octal integers.

In the absence of any other specification, "-o" (octal) is assumed.

For each mode requested, one line of output is produced for each group of eight words in the file. The file offset, in octal, of the first word in the group is prepended to the first line of output for each group.

The portion of the file that is dumped may be controlled with the "+<start>" and "-<end>" arguments. <start> and <end> represent the decimal addresses of the first and last words of the file to be dumped. (The first word has an address of zero.) The two arguments may be used in any combination. If the start address is unspecified, word zero is assumed. Likewise, if no ending address is given, the dump proceeds until end of file is encountered.

If no file name is specified as an argument, standard input one is read, allowing 'fdmp' to be used in a pipeline.

Examples

```
fdmp -bc -127 textfile
weird_program | fdmp
```

Messages

"Usage: fdmp ..." for incorrect arguments.

Usage

```
ffind <string> { -{<option>} } { <file_spec> }
   <option> ::=  c | i | l | o[<occurrences>] | v | x
   <file_spec> ::= <filename> | -[<stdin_number>] |
                        -n(<stdin_number>|<filename>)
```

Description

'Ffind is a filter that selects lines matching a given string from the named files (or standard input if no files are specified) and copies them to standard output. Unlike 'find', 'ffind' cannot match the standard Subsystem patterns but will match only a *literal* string. The algorithm used (Knuth, Morris, and Pratt) is very fast and is typically four to fives times faster than 'find'.

The available options are:

-c If the "c" option is used, only a count of the lines that matched (differed) is printed.

-i If the "i" option is used, the case of the string and the text of the search file(s) is ignored.

-l If the "l" option is used, each line printed is preceded by its relative line number within the file from which it was read.

-o If the "o" option is used, 'ffind' will quit searching the current file after <occurrences> matching (differing) lines have been found in it, and will continue with the next file. If "o" is specified but <occurrences> is omitted, only the first occurrence is found.

-v If the "v" option is specified, each line of output is labelled with the name of the input file from which the line was read.

-x If the "x" option is used, only lines that do not match the string are printed.

The remaining command line arguments are taken as names of files to be searched for the string. The full syntax of the <file_spec> argument is described in the entry for 'cat' (1).

Examples

```
lf -c | ffind .r
guide -p sh | ffind the -ci
```

Messages

"Usage: ffind ..." for bad arguments.
"file: can't open" for unreadable files.

See Also

cat (1), change (1), find (1)

Knuth, D.E., J.H. Morris, Jr., and V.R. Pratt (1977). "Fast pattern matching in strings." SIAM Journal on Computing 6 (No. 2): 240-267.

Usage

```
if <value>
   then
      { <command> }
   else
      { <command> }
fi
```

Description

'Fi' marks the end of a conditional statement. It is a do-nothing command that may be searched for by either 'if' or 'else' in the process of skipping commands. Each 'if' command must be followed by a matching 'fi' command.

If a terminal is locked up due to an unmatched 'if' or 'else', the 'fi' command may be used to regain control.

Examples

```
if [eval [line] ">" 33]
   then
      set type = phantom
   else
      set type = terminal
fi

if 1
   echo "Sorry, you can't use this program."
   goto exit
fi
```

Bugs

I/o redirectors placed before 'fi' render it unrecognizable to 'if' and 'else'.

See Also

if (1), then (1), else (1), case (1)

Usage

```
field  [-f[<width>]]  { <column>           |
                        <column>-<column>  |
                        c<column>          |
                        s<string> }
```

Description

'Field' is designed for manipulation of data in formatted fields. It is a filter that selects data from certain fields of standard input, processes it, and copies it to standard output.

To visualize the action of field, consider the following scenario: Imagine a blank-filled output line. Cut out data from an input line according to column specifications. Paste this data onto the output line at the current column position. Move the current column position to the end of the data just pasted on.

Field provides this "cut and paste" operation as its most basic function. The argument forms <column> (meaning data in the single column given) and <column>-<column> (meaning all data between the given columns, inclusive) transfer fields of data from input line to output line. The argument form s<string> inserts an arbitrary string (called a "padding string") at the current position in the output line. The last argument form (c<column>) resets the current position in the output line to any desired column.

If the "-f" (fixed-length output) option is selected, field will blank-fill output lines to a fixed length as specified by <width>. If <width> is omitted, a value of 72 is assumed. In the default mode (no "-f"), trailing blanks are stripped off.

Field was first designed to ease the problem of stripping sequence numbers from Cobol programs, and still finds most of its work at the same sort of task. It is, however, also useful for arranging multiple key fields before sorting with 'sort'.

Examples

```
cobol_prog> field 1-72 >prog.cob
file> field 5-10 s" " 1-80 | sort | field 8-87 >sorted_file
data_file> field -f80 1-80 >padded_data
```

Messages

"Usage: field ..." for incorrect argument syntax
"<arg>: too many padding strings" for storage area overflow

"<arg>: column out of range" for bad column number
"<arg>: too many fields" for field storage area overflow
"<arg>: bad column syntax" for non-numeric column

See Also

sort (1), lam (1), change (1)

Usage

```
file <pathname> {<option>}
<option> ::= -d | -[n]e | -p twrtwr | -[n]r |
             -s | -u | -[n]w | -[n]z
```

Description

'File' tests the specified pathname for certain conditions. 'File' only operates on one pathname per call and can only test for all specified conditions true (the and-product of all conditions). If no conditions are given, 'file' assumes the "-e" option. All other tests must be specifically turned on. The output of 'file' is a "1" or "0" depending on whether the conditions were all true or one or more was false.

'File' is most commonly used with the 'if' command.

The options available are:

-d	file type is DAM (direct access)
-[n]e	test for the [non] existence of <pathname>
-p twrtwr	test for specific protection bits on
-[n]r	test for [no] read permission on <pathname>
-s	file type is SAM (sequential access)
-u	file type is UFD (directory)
-[n]w	test for [no] write permission on <pathname>
-[n]z	test <pathname> for [non] zero length

Examples

```
if [file [arg 1] -ne]
   echo [arg 1] does not exist
   exit
fi
```

Messages

"Usage: file ..." for illegal argument syntax.

"<pathname>: cannot test conditions" if 'filtst' returned an error in trying to test the pathname.

Primos file system errors will be noted if found.

Bugs

Should accept multiple pathnames.

Should probably have an option to test for 'or' of arguments as well as 'and' of arguments.

Accepts only an obsolete syntax for the file protection argument.

See Also

if (1), chat (1), lf (1), find$ (2)

Usage

```
files [ <pattern> { <'lf'_argument> } ]
```

Description

'Files' is a shell file that invokes the 'lf' command and filters its output through 'find' to select the names that match the specified pattern. The pattern may be any expression that is acceptable (as a pattern) to 'find'. By default, the files in the current working directory are the ones whose names are examined; however, an alternate directory may be specified as a second argument. If no arguments are specified, files produces the same results as an "lf -c" command.

Examples

```
del -v [files ".b$"]

pr [files ".r$"]

files .r$ =src=/lib/swt/src | change .r$ |$
   files .d$ =doc=/man/s2 | change .d$ | common -1
```

Messages

Various messages may be produced by 'lf' and 'find'.

See Also

find (1), lf (1), amatch (2), makpat (2)

Usage

```
find <pattern> { -{<option>} } { <file_spec> }
   <option> ::=  c | i | l | o[<occurrences>] | v | x
   <file_spec> ::= <filename> | -[<stdin_number>] |
                        -n(<stdin_number>|<filename>)
```

Description

'Find' is a filter that selects lines matching a given pattern from the named files (or standard input if no files are specified) and copies them to standard output. The pattern supplied as the first argument is a regular expression with the full set of options found in the editor. (See *Introduction to the Software Tools Text Editor* in the *Software Tools Subsystem User's Guide* for details.)

The available options are:

-c If the "c" option is used, only a count of the lines that matched (differed) is printed.

-i If the "i" option is used, the case of the pattern and the text of the search file(s) is ignored.

-l If the "l" option is used, each line printed is preceded by its relative line number within the file from which it was read.

-o If the "o" option is used, find will quit searching the current file after <occurrences> matching (differing) lines have been found in it, and will continue with the next file. If "o" is specified but <occurrences> is omitted, only the first occurrence is found.

-v If the "v" option is specified, each line of output is labelled with the name of the input file from which the line was read.

-x If the "x" option is used, or if the first character of the pattern is a tilde ("~"), only lines that do not match the pattern are printed.

The remaining command line arguments are taken as names of files to be searched for the pattern. The full syntax of the <file_spec> argument is described in the entry for 'cat' (1). Most frequently, it will take the form of a Subsystem pathname.

'Find' is frequently used for processing output from 'lf' before performing some operation on a number of files, and for stripping out "uninteresting" lines from data to be further processed by other tools.

Examples

```
lf -c | find .r$
lf -c | find .r$ | find call -lv -n
find CALL -lv [lf -c | find .f$]
find "format" -c rf.r ed.r
```

Messages

"Usage: find ..." for bad arguments.
"illegal pattern" for bad pattern syntax.
"file: can't open" for unreadable files.

See Also

cat (1), change (1), ffind (1), files (1), se (1), makpat (2), amatch (2), match (2), *Introduction to the Software Tools Text Editor*

Usage

fmt [-s | -p<start_page>[-<end_page>]] { <filename> }

Description

'Fmt' is an extended version of Kernighan and Plauger's 'format' text formatter.

Input to 'fmt' consists of text intermixed with formatting requests and function calls. Formatting requests are identified by a special character (called the "control character", normally a period) appearing in the first column of a line of input. Such requests are used to change margins, text justification, underlining and boldfacing, etc. Function calls appear within square brackets, and are used to change number registers, query the status of certain internal formatter variables, and effect partial word boldfacing and underlining.

For a complete description of 'fmt', along with a tutorial and numerous examples, the reader is referred to the *Software Tools Text Formatter User's Guide*.

If the "-s" option is specified, 'fmt' will pause at the top of each page of output, to allow the user to insert paper manually. To continue output, the user should type a control-c.

The "-p" option may be used to limit the pages output by 'fmt'. Only the given range of pages will be printed. If the ending page number is omitted, all remaining text will be printed.

The files named on the command line are used as sources of input. The effect is the same as if the contents of all the named files had been concatenated before processing. Note: the filename "-" may be used to indicate the first standard input.

The following tables summarize currently-implemented formatting requests and function calls (and their variants).

Summary of Commands

Command Syntax	Initial Value	If no Parameter	Cause Break	Explanation
.#	-	-	no	Introduce a comment.
.ad c	both	both	no	Set margin adjustment mode.
.am xx	-	-	no	Add additional text to the body of a

				previously defined macro.
.bf N	N=0	N=1	no	Boldface N input text lines.
.bp +N	N=1	next	yes	Begin a new page.
.br	–	–	yes	Force a break.
.c2 c	`	`	no	Set no-break control character.
.cc c	.	.	no	Set basic control character.
.ce N	N=0	N=1	yes	Center N input text lines.
.de xx	–	ignored	no	Begin definition or redefinition of a macro.
.dv <stream>	–	end '.dv'	no	Temporarily divert the output stream to a "filename" or to a temporary file designated by an integer "N" (to be later read by a ".so N" command) until a 'dv' command with no arguments is seen.
.ef /l/c/r/	blank	blank	no	Set even-numbered page footing.
.eh /l/c/r/	blank	blank	no	Set even-numbered page heading.
.en xx	–	ignored	no	End macro definition.
.eo +N	N=0	N=0	yes	Set even page offset.
.er text	–	ignored	no	Write a message to the terminal.
.ex	–	–	yes	Exit immediately to the Subsystem.
.fi	on	–	no	Turn on fill mode.
.fo /l/c/r/	blank	blank	no	Set running page footing.

.he /l/c/r/	blank	blank	no	Set running page heading.
.hy	on	-	no	Turn on automatic hyphenation.
.if <args>	-	ignored	maybe	Conditional execution of an input line.
.in +N	N=0	N=0	yes	Indent left margin.
.lm +N	N=1	N=1	yes	Set left margin.
.ls N	N=1	N=1	no	Set line spacing.
.lt +N	N=60	N=60	no	Set length of header, footer and titles.
.m1 +N	N=3	N=3	no	Set top margin before and including page heading.
.m2 +N	N=2	N=2	no	Set top margin after page heading.
.m3 +N	N=2	N=2	no	Set bottom margin before page footing.
.m4 +N	N=3	N=3	no	Set bottom margin including and after page footing.
.mc <char>	BLANK	BLANK	no	Set margin character.
.mo +N	N=0	N=0	no	Set margin offset.
.na	-	-	no	Turn off margin adjustment.
.ne N	-	N=1	yes	Express a need for N contiguous lines.
.nf	-	-	yes	Turn off fill mode. (Also inhibits adjustment.)
.nh	-	-	no	Turn off automatic hyphenation.
.ns	on	-	no	Turn on 'no-space' mode.

.nx file	–	next arg	no	Move on to the next input file.
.of /l/c/r/	blank	blank	no	Set odd-numbered page footing.
.oh /l/c/r/	blank	blank	no	Set odd-numbered page heading.
.oo +N	N=0	N=0	yes	Set odd page offset.
.pl +N	N=66	N=66	no	Set page length.
.pn +N	N=1	ignored	no	Set page number.
.po +N	N=0	N=0	yes	Set page offset.
.ps N M	N=M=0	N=M=0	yes	Skip pages while (page number mod M) is less than N.
.rc c	BLANK	BLANK	no	Set tab replacement character.
.rm +N	N=60	N=60	yes	Set right margin.
.rs	–	–	no	Turn off 'no-space' mode.
.sb	off	–	no	Single blank after end of sentence.
.so <stream>	–	ignored	no	Temporarily alter the input source. "Stream can be a "–" to indicate standard input, a "filename," or an integer "N" corresponding to a temporary file created by a previous '.dv N' command.
.sp N	–	N=1	yes	Put out N blank lines.
.ta N ...	9 17 ...	all	no	Set tab stops.
.tc c	TAB	TAB	no	Set tab character.
.ti +N	N=0	N=0	yes	Temporarily indent left margin.
.tl 'l'c'r'	blank	blank	yes	Generate a three part title.

.ul N	N=0	N=1	no	Underline N input text lines.
.xb	on	-	no	Extra blank after end of sentence.

Functions

add	Add constant to number register (add <reg_number> <constant>)
bf	Boldface arguments on output
cu	Output arguments with a continuous underline
date	Current date; e.g., 11/27/84
day	Current day of the week; e.g., Tuesday
ldate	Current date: e.g., November 27, 1984
num	Evaluate number register (num <pre-inc/dec><reg_number><post-inc/dec>)
rn	Convert argument to a lower-case Roman numeral
RN	Convert argument to an upper-case Roman numeral
set	Set number register to value (set <reg_number> <constant>)
sub	Output the arguments as a subscript
sup	Output the arguments as a superscript
time	Current time of day; e.g., 02:16:12
ul	Underline the arguments on output
letter	Convert a number to its lower case equivalent
LETTER	Convert a number to its upper case equivalent
vertspace	Change the vertical spacing on NEC Spinwriter
even	Test if number is even
odd	Test if number is odd
cap	Capitalize text
small	Map all characters of text to lower case
plus	Add two numbers
minus	Subtract two numbers
header	Return the page header
evenheader	Return the even page header
oddheader	Return the odd page header
footer	Return the page footer
evenfooter	Return the even page footer
oddfooter	Return the odd page footer
cmp	Perform string comparison
icmp	Perform integer comparison
bottom	Return the number of the last printed line
top	Return the number of the first printed line

Variables

cc	Current basic control character
c2	Current no-break control character
in	Current indentation value
lm	Current left margin value
ln	Current line number on the page

ls	Current line-spacing value
lt	Length of titles
ml	Current macro invocation level
m1	Current margin 1 value
m2	Current margin 2 value
m3	Current margin 3 value
m4	Current margin 4 value
ns	True or false if no-space is in effect.
pl	Current page length value
pn	Current page number
po	Current page offset value
rm	Current right margin value
tc	Current tab character
ti	Current temporary indentation value
tcpn	Current page number, right justified in 4 character field

Special Characters

	bl	Phantom blank
	bs	Backspace
	alpha	lower-case Greek alpha
*	ALPHA	upper-case Greek alpha
	beta	lower-case Greek beta
*	BETA	upper-case Greek beta
*	chi	lower-case Greek chi
*	CHI	upper-case Greek chi
	delta	lower-case Greek delta
*	DELTA	upper-case Greek delta
	epsilon	lower-case Greek epsilon
*	EPSILON	upper-case Greek epsilon
	eta	lower-case Greek eta
*	ETA	upper-case Greek eta
	gamma	lower-case Greek gamma
	GAMMA	upper-case Greek gamma
	infinity	infinity symbol
	integral	integration symbol
*	INTEGRAL	large integration sign
*	iota	lower-case Greek iota
*	IOTA	upper-case Greek iota
*	kappa	lower-case Greek kappa
*	KAPPA	upper-case Greek kappa
	lambda	lower-case Greek lambda
	LAMBDA	upper-case Greek lambda
	mu	lower-case Greek mu
*	MU	upper-case Greek mu
	nabla	inverted delta (APL del)
	not	EBCDIC-style not symbol
*	nu	lower-case Greek nu
*	NU	upper-case Greek nu
	omega	lower-case Greek omega
	OMEGA	upper-case Greek omega
*	omicron	lower-case Greek omicron
*	OMICRON	upper-case Greek omicron
	partial	partial differential symbol
	phi	lower-case Greek phi

```
        PHI           upper-case Greek phi
        psi           lower-case Greek psi
        PSI           upper-case Greek psi
        pi            lower-case Greek pi
        PI            upper-case Greek pi
        rho           lower-case Greek rho
      * RHO           upper-case Greek rho
        sigma         lower-case Greek sigma
        SIGMA         upper-case Greek sigma
        tau           lower-case Greek tau
      * TAU           upper-case Greek tau
        theta         lower-case Greek theta
        THETA         upper-case Greek theta
      * upsilon       lower-case Greek upsilon
      * UPSILON       upper-case Greek upsilon
        xi            lower-case Greek xi
      * XI            upper-case Greek xi
        zeta          lower-case Greek zeta
      * ZETA          upper-case Greek zeta
      * downarrow     arrow pointing down
      * uparrow       arrow pointing up
      * backslash     back slash symbol
      * tilde         tilde symbol
      * largerbrace   large square right brace
      * largelbrace   large square left brace
      * proportional  proportional symbol
      * apeq          approximately equal to
      * ge            greater than or equal to
      * imp           implies
      * exist         there exists
      * AND           logical and
      * ne            not equal to
      * psset         proper subset
      * sset          subset
      * le            less than or equal to
      * nexist        there does not exist
      * univ          for every
      * OR            logical or
      * iso           congruence
      * lfloor        left floor
      * rfloor        right floor
      * lceil         left ceiling
      * rceil         right ceiling
      * small0        a small 0
      * small1        a small 1
      * small2        a small 2
      * small3        a small 3
      * small4        a small 4
      * small5        a small 5
      * small6        a small 6
      * small7        a small 7
      * small8        a small 8
      * small9        a small 9
      * scolon        semicolon
      * dquote        double quote
      * dollar        dollar sign
```

The special characters marked with an asterisk (*) are only available on the NEC **Spinwriter**, and so the output of 'fmt' *must* be post-processed with 'sprint'.

In particular, these characters require that the special Times-Roman/Mathematics type wheel be in the **Spinwriter**. This wheel, in order to accomodate the special characters, lacks certain of the regular ASCII graphics. These are substituted for by special functions. For example, [scolon] is used to produce a semi-colon.

Examples

```
fmt -p3-10 report | dprint
fmt report | os >/dev/lps/f
fmt -s contents tutorial index
```

Bugs

There should be some way to specify multiple ranges of pages to be printed.

See Also

os (1), dprint (3), sprint (3), *Software Tools Text Formatter User's Guide*

Usage

forget { <identifier> }

Description

'Forget' is used to destroy shell variables that were created by 'declare' or 'set'. The arguments supplied must be names of shell variables that are active at the current lexical level. The named variables will be removed from the command interpreter's symbol table.

Note that it is not necessary to explicitly destroy shell variables that are declared local to a command file; when the execution of the command file is completed, they will be destroyed automatically.

Examples

```
forget name address telephone_number
forget face
```

See Also

declare (1), set (1), vars (1), save (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

fos [<pathname> { <spool options> }]

Description

The shell program 'fos' executes the proper pipeline to produce a formatted document on the line printer. If called with any arguments, the first argument must be the file to be formatted, possibly followed by spooler options. If no arguments are supplied, then 'fos' will attempt to use the 'f' variable that is set by 'e' (if declared). If the 'f' variable is not declared, then an error message is printed. The spool options are any options acceptable to 'sp'.

'Fos' can take only one <pathname>, while 'fmt' can take many. To make 'fos' accept several names, one can type:

fos "file1 file2 file3"

Examples

fos report
fos book -c 1000

Messages

"Usage ..." for missing pathname argument and no 'f' variable

See Also

e (1), sp (1), fmt (1), os (1)

Usage

```
fsize  { -<option>{<option>} } { <pathname> | -n }
       <option> ::=  f | r | v | w
```

Description

'Fsize' prints the amount of disk space consumed by the file system objects specified as arguments. Any type of file system object may be specified (ordinary file, ufd, or segment directory); in the case of a ufd or segment directory, the entire file system subtree thereunder is considered in the size. 'Fsize' will determine the amount of space used not only for data but also for internal purposes (such as direct access indices).

If "-n" appears in place of a pathname, pathnames are read from the standard input. For more information on this syntax, see the entry for 'cat' (1).

The following options are available:

-f Force any object that can't be read to be readable by manipulating its protection bits and concurrent access lock. After the object has been sized, the original attributes are restored. The user must have owner status in the object's parent directory for this option to work. **WARNING:** This option should *not be used on objects whose protection and concurrency attributes are assumed by some running program to have a particular setting*. In particular, certain of the files used by Prime's database management system must not have their concurrency locks changed while the DBMS is active.

-r Print the size as a number of disk records (default).

-v Print the name of the object along with the size of the object. This is especially useful when pathnames are being read from the standard input.

-w Print the size as a number of 16 bit words.

If the "-n" idiom is not used and no pathnames are given on the command line, the program will size the current directory by default.

Examples

```
fsize
lf -c | fsize -f -n | stats -tashln
fsize -wv  //extra //lib //src
```

Messages

"Usage: fsize ..." for incorrect argument syntax.
"-r and -w are mutually exclusive" for requesting both "r" and "w".
"<pathname>: can't get record size" if the record size of the disk partition containing the specified object can't be determined.
"<pathname>: can't determine size" if the specified object can't be opened for reading.
"<pathname>: can't size directory contents" if the specified ufd can't be attached to.

Bugs

Gives inaccurate results for very large DAM files and DAM segment directories (those with multi-level indices).

Will not work if the MFD of the disk containing an object cannot be attached to with a password of "XXXXXX". The MFD must be read to determine the record size for the disk.

See Also

lf (1), hd (1)

Usage

```
fsubc <input file>
         [-l [<listing file>]]
         [-z <FSUBS option>]
```

Description

'Fsubc' serves as the Subsystem interface to the Prime DBMS Fortran subschema compiler (FSUBS). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and output files as needed, and then produces a Primos FSUBS command and causes it to be executed.

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. If the "-l" option is specified without a file name following it or is not specified, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.fsub", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l".

The input filename must be a disk file name (conventionally ending in ".fsub").

In summary, then, the default command line for compiling a file named "file.fsub" is

```
fsubc  file.fsub  -l file.l
```

which corresponds to the FSUBS command

```
fsubs -i *>file.fsub -l *>file.l
```

Examples

```
fsubc file.fsub
fsubc payroll.fsub -l l_payroll
fsubc funnyprog.fsub -z"-newopt"
```

Messages

"Usage: fsubc ..." for invalid option syntax.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"Sorry, the listing file must be a disk file" if the listing

file was directed to a device file.

Bugs

'Fsubc' pays no attention to standard ports.

There is no way to avoid getting a listing file.

See Also

ddlc (1), f77c (1), fc (1), fdmlc (1), ld (1), bind (3)

Usage

goto <label>

Description

'Goto' provides a means of altering the flow of control in command files. After the execution of a 'goto' command, the command interpreter will resume processing of the command file at the first command (network, to be precise) labelled with the given identifier. A node (and thus a network) may be labelled by preceding it with a colon and an identifier, e.g.

```
:exit echo Done.
```

'Goto' is normally used only within command files; however, it may be used from the terminal, with the restriction that control can only be transferred forward, never back.

Examples

goto exit

Messages

"goto could not find target label" for a missing label.
"Usage: goto <label>" if used without an argument.

Bugs

'Goto' does not understand compound nodes, so jumping into the middle of one may have unpredictable results. If the target label is preceded by any I/O redirectors, it will not be recognized.

See Also

if (1), case (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

```
group [-a | -o] {<group_list>}
```

Description

'Group' lists a user's currently active groups or tests for combinations of groups. With no arguments, 'group' lists all of a users currently active groups. The "-a" option causes 'group' to return a "1" if all the groups listed are active (i.e. returns the logical AND) and '0' otherwise. The "-o" option causes 'group' to return "1" if any of the listed groups are currently active (i.e. returns the logical OR) and a '0' otherwise. If a "<group_list>" is specified with no flag argument then 'group' assumes "-a".

Examples

```
group
group .guru
group -a .demo .system
group -o .test .strange
```

Messages

"can't retrieve group names" if an error in the call to GETID$ occurs.

See Also

Primos List_Group command

Usage

gtod

Description

'Gtod' prints the time of day together with the date, month, and year; in a format which is pleasing to humans.

Examples

```
gtod
echo Compiled at [gtod | quote]
```

See Also

date (1), time (1), day (1), ctime (1), date (2)

Usage

```
guide  { <option> | <item> }
   <option> ::=  -p
   <item> ::= <guide_name>
```

Description

Several of the more complicated or more frequently used Subsystem commands and libraries have additional documentation beyond that which is available from the reference manual. This documentation is in the form of a separate paper on each command or library, but these papers may be combined to form the *Software Tools Subsystem User's Guide*.

The command

guide <guide name>

prints the named guide in a format suitable for reading on a fast CRT terminal.

The command

guide -p <guide name>

prints the named guide in a format suitable for the line printer.

Copies of individual documents may be printed on one of the on-site line printers by giving the following command:

guide -p <guide name> | os >/dev/lps/f

where "<guide name>" is one of the following guide names:

cc

A copy of the *User's Guide for the Georgia Tech C Compiler*. This guide describes the necessary requirements for compiling programs written in C from the Subsystem. Refer to *The C Programming Language* by Brian Kernighan and Dennis Ritchie for specific details about the C programming language. **This guide is only available to customers who have also licensed the C language compiler package.**

ed

A copy of *Introduction to the Software Tools Text Editor* is printed. This paper includes a tutorial on the Subsystem's text editor that is highly recommended for beginning users, as well as a command summary and a

special section on the Subsystem screen editor.

fmt

A copy of the *Software Tools Subsystem Text Formatter User's Guide* is printed. This includes tutorial, reference, and applications information. One very useful appendix contains all text formatting commands, arranged alphabetically.

fs

A copy of *User's Guide to the Primos File System* is printed. This paper gives a brief introduction to the Primos file system as it applies to the use of the Subsystem. It explains the structure of the file system, provisions for security, and how users access files by name.

math

A copy of the *SWT Math Library User's Guide* is printed. This includes descriptions of the Prime floating point hardware, the SWT math library, and the tests used to validate the SWT library. Appendices contain useful programs to help determine where the exponent is located on your particular machine, determine the amount of loss of bits in a multiply operation, and calculate hexadecimal constants for use in mathematical routines. The addendum documents the routines which used to be in the old, locally supported, math library "vswtml."

mgr

A copy of the *Software Tools Subsystem Manager's Guide* is printed. This guide is useful for all Subsystem managers and anyone else interested in the installation, maintenance, and daily operation of the Subsystem.

ring

A copy of *Ring -- The Software Tools Subsystem Network Utility* is printed. This paper documents the structure and use of 'ring', a utility which makes it easier for the end user to deal with Primenet.

rp

Prints the *Ratfor Programmer's Guide*. This document includes a detailed description of the Ratfor programming language as well as instructions for its use under the Subsystem. It is essential for anyone hoping to do any significant amount of programming using the capabilities supplied by the Subsystem.

sh

A copy of *User's Guide for the Software Tools Subsystem Command Interpreter* is printed. This paper discusses the features of the Subsystem command interpreter, called the 'shell', on three levels: a tutorial introduction, a syntax and semantics reference, and a set of applications notes.

tutorial

A copy of *The Software Tools Subsystem Tutorial* is printed. This tutorial is intended as a user's first introduction to the Subsystem and covers such essentials as logging in and out, features of the command language, editing, online documentation and so forth. NEW USERS SHOULD READ THIS DOCUMENT FIRST.

v8.1

A copy of the *Software Tools Subsystem Version 8 to Version 8.1 Conversion Guide* is printed. This guide summarizes all user-visible changes that have been made between the Version 8 and Version 8.1 Subsystems.

v9

A copy of the *Software Tools Subsystem Version 8.1 to Version 9 Conversion Guide* is printed. This guide summarizes all user-visible changes that have been made between the Version 8.1 and Version 9 Subsystems.

vcg

A copy of *A Re-Usable Code Generator for Prime 50-Series Computers User's Guide*. 'Vcg' is a reusable general-purpose code generator that accepts an "intermediate form" and produces 64V-mode relocatable object code, or optionally, PMA. The V-mode code generator is the back-end for the Georgia Tech C Compiler. **This guide is only available to customers who have also licensed the C language compiler package.**

Examples

```
guide tutorial
guide -p rp | os >/dev/lps/f
```

Files

Most of those contained in =doc=/fguide.

See Also

Software Tools Subsystem Reference Manual

Usage

```
hd [ -n | -u | -v ] { <pack id> }
```

Description

'Hd' ("how's disk?") prints a summary of available disk space. Zero or more <pack id> arguments may be used to specify the disk partitions of interest. A <pack id> may be the packname of the partition (the name of the record availability table file) or its *octal* logical disk number in the range 0:76 or an asterisk indicating the disk containing the user's current directory. If no <pack id> arguments are given, information for all online disks, in order of increasing logical disk number, is provided.

The format of each line of output is as follows:

```
nn:  rrrrrr free pppppp% full ssss...
```

where 'nn' is the logical disk number in octal (if the disk number is known), 'rrrrrr' is the (decimal) number of records available on the partition, 'pppppp' is the percentage of total records on the partition that are currently in use, and 'ssss...' is the packname of the partition.

The "-n" and "-u" options may be used to determine the base record size for reporting the number of available records on storage module partitions (whose physical record size is 1024 words). If the "-n" option is specified, the number of physical records available is "normalized" to an equivalent number of 440 word records. If the "-u" option is specified, the number of available physical records is reported as is, without normalization. If neither option is given, "-u" is assumed.

If the "-v" option is used, 'hd' will also print the number of heads and total number of records in each partition.

Examples

```
hd
hd swtsys user_a 10
hd -v *
hd -n cc
```

Files

Record availability tables and master file directories on all reported disks.

Messages

"Usage: hd ..." for incorrect argument syntax.
"bad packname" for unrecognized packnames or out-of-range logical disk numbers.
"disk-rat unreadable" if the record availability table can't be opened for reading.
"disk-rat badly formatted" for a record availability table that does not conform to the standard format.
"mfd unreadable" if the master file directory on the partition can't be opened for reading.
"-n and -u are mutually exclusive" if both "-n" and "-u" options are specified.

Bugs

The name is not terribly mnemonic. It is a holdover from the long defunct Georgia Tech Burroughs B5500.

See Also

fsize (1), lf (1)

Usage

```
help  { <item> | <option> }
   <option> ::= -c | -d | -s | -f | -g | -i | -p | -u
```

Description

'Help' can be used to retrieve various types of information concerning Subsystem commands and library subprograms.

General information on the Subsystem can be had simply by typing the command "help" with no arguments:

```
help
```

More comprehensive information can be obtained with the form

```
help item item...
```

'Help' searches the *Software Tools Subsystem Reference Manual* for the named commands and subprograms and, if they are found, prints their manual entries. Any uniquely-named command or library subprogram may be found in this manner.

In the case of commands and subprograms that share a common name (e.g. 'print' or 'date') the ambiguity may be resolved by specifying the option "-c" to select the command or "-s" to select the subprogram. If neither "-s" or "-c" is specified, the default behavior is the same as for "-c".

General information not in the Reference Manual is accessed with the "-g" option; for example,

```
help -g bnf
```

gives a short explanation of the extended Backus-Naur Form (BNF) used to describe command syntax in the Reference Manual.

An index of all documented commands and library subprograms can be generated with the "-i" option. (This is an excellent way of getting an overview of what functionality the Subsystem has to offer.) Furthermore, if some particular function is desired, but the names of commands that perform that function are unknown, the "-f" option may be used to search the index for a given pattern. For example, the names and short descriptions of all commands and library subprograms dealing with character strings will be listed by the following command:

```
help -f string
```

(The "-f" option is an excellent way for a new user to track down commands and subprograms that are germane to the solution of a particular problem.) (An aside to experienced

users: the patterns following a "-f" option are standard Subsystem regular expressions, identical to those used in the text editors and the 'find' and 'change' commands.)

'Help' calls the 'page' subroutine in the Subsystem library to print a screenful of information at a time; any response that is acceptable to 'page' can be given as a response to 'help'. Please see the Reference Manual entry for the 'page' routine for more details ("help page" would be appropriate). In particular, a carriage return may be entered to continue to the next screenful of information. While the 'help' processor is presenting the text of a Reference Manual entry, it prompts with the a string of the form "<name> [<number>+] more ? ", where <name> is the name of the command or routine for which help is being provided, and <number> is the number of the page (screenful) being presented. When the end of the Reference Manual entry is reached, 'help' prompts with the string "<name> [<number>$] more ? ", with the dollar sign indicating that the end of the manual entry has been reached. By default, 'help' instructs the 'page' subroutine to use any special features your terminal may have, via the 'vth' terminal handling library. If you have a dumb terminal, or a hard-copy terminal, use the "-d" option to tell 'help' that it is using a "dumb" terminal.

For extracting Reference Manual entries to be spooled and printed, the "-p" option may be used to turn off the automatic pagination described above. When "-p" is specified, the Reference Manual entries selected are printed exactly as they are stored, with underlining/boldfacing intact, and indentation and page size unchanged. This output must be run through 'os' before being printed on the line printer. Example:

```
help -p help rp fmt | os >/dev/lps/f
```

If the user only desires to see the syntax for the command and not the description, then the "-u" option can be specified. This causes only the "Usage" section of the Manual entry to be retrieved for the given command. The 'usage' Subsystem command is a shell file that uses this option; the user normally does not need to specify it in a call to 'help'.

Examples

```
help
help -g bnf
help e se date time
help -s date -c print
help -i
help -f file string input output
help -p fmt | os >/dev/lps/f
help -u se
```

Files

=doc=/fman/s1/<command>.d for command documentation
=doc=/fman/s2/<subprogram>.d for subprogram documentation
=doc=/fman/s3/<command>.d for local command documentation
=doc=/fman/s4/<subprogram>.d for local subprogram documentation
=doc=/fman/s5/<command>.d for low-level command documentation
=doc=/fman/s6/<subprogram>.d for low-level subprogram documentation
=doc=/fman/contents for command and subroutine index
=temp=/tm?* for temporary files to store the Reference Manual entry for paging

Messages

"Sorry, no help is available for <command>" in case of missing or unreadable documentation file.
"Can't open index file =doc=/fman/contents" in case index file is missing or unreadable.
"<pattern>: bad pattern" in case there is a syntax error in a pattern following a "-f" option.
"cannot create scratch file for help entry" if a file in the =temp= directory could not be opened to store the help text for paging.
"cannot close scratch file" if the scratch file in =temp= could not be closed.

See Also

guide (1), pg (1), usage (1), page (2), *Software Tools Subsystem Reference Manual*

Usage

hist [on | off | save [<file>] | restore [<file>]]

Description

The Shell contains a mechanism (similar to Berkeley Unix's C-Shell) called a history mechanism. This is sort of dynamic macro facility that allows the user to specifiy a unique substring of a previous command and have the command recalled and re-executed or have portions of the command edited and inserted into the current command.

Up to 128 commands are saved in a circular command queue. Commands are seached for and retrieved from this queue.

The possible options to 'hist' do the following

on Turn the history mechanism on and reset the queue. If history is already enabled then this will clear whatever command history exists in the queue.

off Turn the history mechanism off. Any command history in the queue is lost.

save Save the current command history in the specified file. If no file is specified, the command history is saved in the file "=histfile=".

restore Restore the command history from a previous session from the specified file. If no file is specified, the command history is restored from the file "=histfile=".

'Hist' with no options produces a list of the current command history on STDOUT.

See the *User's Guide for the Software Tools Subsystem Command Interpreter* for a more detailed explanation of the history mechanism, and examples of its use.

Examples

```
hist
hist on
hist off
hist save
hist restore //jeff/bin/scum
```

See Also

User's Guide for the Software Tools Subsystem Command Interpreter

Usage

history

Description

'History' is a simple command file that keeps a history of changes to the Subsystem. It is intended for use by the Subsystem implementors to keep a record of changes and additions.

'History' is invoked with no arguments, since they are not used. The user invoking 'history' must be in the group '.guru' (users who are all powerful and all knowing) in order to be able to make changes in the Subsystem history. Since most systems do not have the '.guru' group, the local administrator should change =src=/std.sh/history.sh to use an appropriate test. Essentially, one should test if the user has permission to modify the history file.

To see what changes have been made to the Subsystem, use the command 'phist'.

Examples

history

Files

=doc=/hist/history for the history of the Software Tools Subsystem.

Messages

"Must be a guru to issue this command" if the user is not allowed to change the Subsystem history.

See Also

phist (1)

Usage

hp { <expression elements> }

Description

'Hp' is a desk calculator program using the Reverse Polish Notation familiar to all stack machine aficionados and users of Hewlett-Packard calculators. It accepts expressions composed of operands and operators from either its argument list or its first standard input and evaluates them.

If the expressions to be evaluated are given on the command line, 'hp' prints the resulting value automatically; otherwise, the user must request printing with the "p" or "P" commands.

An acceptable expression consists of a sequence of "constants" and "commands." Constants are numeric constants written in the style of Fortran, and are stored internally as double precision floating-point values. Commands are single characters that request an arithmetic, stack control, or control flow operation. The following commands are currently implemented:

p print the value on the top of the stack.

P print all the values currently on the stack.

d delete the top value on the stack (throw it away).

D empty the stack completely (throw all stacked data away).

+ add top two items on the stack, place sum on the stack.

- subtract top of stack from next to top, place difference on the stack.

* multiply top two items on the stack, place product on the stack.

/ divide next to top of stack by top of stack, place quotient on the stack.

^ evaluate (next to top of stack) to the (top of stack) power, place result on the stack.

< if next to top of stack is less than top of stack, place a 1 on the stack; otherwise, place a 0 on the stack.

= if next to top of stack equals top of stack, place a 1 on the stack; otherwise, place a 0 on the

stack.

`>` if next to top of stack is greater than top of stack, place a 1 on the stack; otherwise, place a 0 on the stack.

`&` if next to top of stack is nonzero and top of stack is nonzero, place a 1 on the stack; otherwise, place a 0 on the stack.

`|` if next to top of stack is nonzero or top of stack is nonzero, place a 1 on the stack; otherwise, place a 0 on the stack.

`~` if top of stack is nonzero, replace it with a 0; if it is zero, replace it with a 1 (logical negation).

Examples

```
hp 32.75 4.5 *
hp
1 2 3 4 5 6 7P++++++pd
3.1416 2.7183^ 2.7183 3.1416^>p
```

Messages

"stack underflow" if an attempt is made to perform an operation with insufficient operands on the stack.
"<char>: unrecognized command" if an character not corresponding to any command appears in an expression.

See Also

eval (1), stacc (1)

Usage

```
if <value>
   then
      { <command> }
   else
      { <command> }
fi
```

Description

'If' allows users of the Shell's programming facilities to execute commands conditionally, after the fashion of the Algol 68 conditional clause.

The <value> after the if command may be any string; if it is zero, empty, or missing altogether, it is interpreted as false; otherwise, it is interpreted as true. If <value> is true, then the commands after the keyword 'then' are executed; otherwise, the commands after the keyword 'else' are executed. In either alternative, any commands (including nested if commands) may be used.

The keyword 'then' is optional. The keyword 'else' and its associated commands may be omitted if no action is desired when <value> is false. The keyword 'fi' is mandatory.

'If' is not restricted to use in command files, and so may produce puzzling results when used incorrectly from a terminal. These can always be handled by typing a 'fi' command or by generating end-of-file to the command interpreter.

Examples

```
if [nargs]
   set params = [args]
fi

if [eval i ">=" 10]
   then
      goto exit
   else
      set i = [eval i + 1]
      goto loop
fi

if [flag]; then; echo "Success!"
else; echo "Failure..."
fi
```

Messages

"Missing 'fi'" if end-of-file is reached on command input before a matching 'fi' was found.

Bugs

Redirectors in front of the 'else' will prevent it from being recognized.

Typing "if" on someone's terminal will cause the Shell to ignore any command they type until an EOF or an unmatched 'fi' is typed.

See Also

then (1), else (1), fi (1), case (1), goto (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

include

Description

Many Ratfor programs use the Ratfor "include" statement to include a frequently used body of code, such as the standard definition file "=incl=/swt_def.r.i", as part of the source input. This is useful for saving disk space, but is sometimes inconvenient if the programmer wishes to see the entire text of his program. The 'include' command is provided to make this possible. 'Include' copies its standard input to its standard output, while looking for lines that begin with "include", followed by a file name, possibly enclosed in quotes (" or '). If such a line is found, the contents of the named file are inserted in its place and copying continues as before. Files to be included may be nested to a depth of 5.

Examples

prog.r> include | pr

Messages

"Can't open include" if include file could not be found.

See Also

rp (1), macro (1)

Usage

index <string> <character>

Description

'Index' is a version of the PL/I index function. The string specified as the first argument is searched for an occurrence of the character specified as the second argument; if the character is found, 'index' prints its location in the string (first character in the string is at position 1) on standard output. If the character is not found, zero is printed.

'Index' is equivalent to the 'index' subprogram available in the standard Software Tools Subsystem library.

Examples

```
index "abcdefghijklmnopqrstuvwxyz" a
index [upalf] a
take [index [string] " "]] [string]
```

Bugs

None, unless you consider the argument order a bug.

See Also

take (1), drop (1), substr (1), index (2)

Usage

installation

Description

'Installation' merely prints the Subsystem installation name on standard output.

The installation name resides in the file "=extra=/installation" and may be changed at the discretion of the Subsystem manager.

Examples

```
echo Run at [installation]
```

Files

=extra=/installation

Usage

iota [<lower_limit>] <upper_limit> [-f <format>]

Description

'Iota' is derived from the monadic APL operator of the same name; it prints a series of consecutive integers on standard output. The <upper_limit> and optional <lower_limit> arguments specify the range of integers to be printed. The default <lower_limit> is one.

The <format> argument is a standard format string, identical to that accepted by 'encode' or 'print'. Its presence allows the user to select fill characters, field width, and other parameters associated with the printing of integers.

Examples

```
iota 10
stack(i)
iota [most_recent] 1
iota -5 5 -f "*4,-16,0i"
```

Messages

"Usage: iota ..." for invalid argument syntax.

Bugs

If sharp signs ("#") are included in the format string, 'iota' will die of a pointer fault in 'encode'.

See Also

parscl (2), encode (2), print (2)

Usage

```
isph
```

Description

'Isph' allows a shell file to test and see if its invoker is a phantom. It writes a "1" to standard output if the invoker is a phantom, and a "0" it it is being run from a terminal.

Examples

```
if [isph]
   then
      error "screen editor must be run at a terminal"
   else
      se -a my_prog
fi
```

Usage

join [<delimiter>] [-l<nlines>]

Description

'Join' reads its first standard input, replaces all NEWLINEs with the <delimiter> string, and writes the result on its first standard output. The <delimiter> argument may be specified as any arbitrary string. If it is omitted, a single blank is assumed. If the "-l<nlines>" construct is specified, a maximum of <nlines> input lines will be joined into each output line.

Examples

```
files .r | join -l10 | change % "ar -u arch " | sh
file1> join "|" >file2
```

Usage

kwic [-d [<discard list>]]

Description

'Kwic' is the key-word-in-context program from *Software Tools*. It is a filter, taking lines of text from its standard input, rotating them so that each word in the sentence appears at the beginning of a line, and marking the original position of the NEWLINE with a "fold character" (currently a grave accent with zero parity bit).

If the "-d" option is used, 'kwic' will read a sorted list of words, either from the file specified by <discard list>, or from standard input two if the file name is omitted. If the first word in a rotated line is found in the list, the line will not be written out. The discard file should contain one word per line, in lower case. (Before searching the list, 'kwic' converts the search key to lower case.)

The output from 'kwic' is typically sorted with 'sort' then "un-rotated" with 'unrot' to produce the finished key-word-in-context index.

Examples

```
text> kwic | sort | unrot >index
headers> kwic -d discard_list >headers.k
headers> discard_list> kwic -d >headers.k
```

Messages

"<file>: cannot open" if discard list cannot be read.
"<word>: discard list too long" if there are too many words in the discard list.

See Also

sort (1), unrot (1), *Software Tools*

Usage

```
lacl {<option>} {<file_spec>}
      <option> ::=  -(a | b | c | l | p | t | v)
```

Description

The 'lacl' command will list information about the Access Control Lists protecting any file system object. If no pathname is specified, 'lacl' will print ACL information on the current directory. For a more comprehensive description of ACL's, see the help for the 'sacl' command. For a full description of <file_spec>, see the help on 'cat'.

Options recognized by 'lacl':

-a List the access pairs describing the ACL for the object. This is the default action if no options are specified, and the "-a" must be specified if you wish to display the pairs when also specifying the "-t" and "-b" options.

-b Give the pathname of the object protecting the named item. The pathname is the same as the object for specific ACLs, or the name of the acat involved for access category protection. The pathname may also be of an ancestor directory in the case of default specific ACLs. If the "-p" option is also given the "-b" is ignored.

-c Print the access pairs one per column instead of all on the same line.

-l Long format listing. Acts as if the options "-a -b -c -v -t" were all given.

-p List the priority ACL in effect for the logical disk partition on which the object resides.

-t Give the type of the ACL protecting the object. The type is either "specific", "default specific", "acat", "default acat", "object is an acat", or "priority".

-v Verbose form -- echo the pathname of the object being checked and include separator characters if the "-b", "-l", or "-t" options have also been selected.

Examples

```
lacl -p -v /0/mfd /1/mfd

lf -fc =vars= | lacl -abv -n
```

lacl

Messages

"Usage: lacl [-l] [-b [-a]] [-p] [-t] [-c] [-v] {<pathname>}" for improper command usage.
"Cannot list acl for <pathname>" for various file system errors or insufficient access rights.

See Also

lf (1), sacl (1), gfdata (3)

Usage

```
lam {-i<string> | <filename>}
```

Description

'Lam' is used to combine multiple input streams into one output stream by placing corresponding lines from each input stream end-to-end. For example, if STDIN1 contains

```
line #
line #
```

and STDIN2 contains

```
1
2
```

then the result of the command "lam" will be

```
line #1
line #2
```

If an input stream is shorter than the others, its contribution to the output is null once it reaches EOF.

The "-i" arguments may be used to insert arbitrary strings into the output stream, either before the lamination, after it, or between any two files. The string to be inserted must follow the "-i" immediately; it may not be placed in the following argument.

If no arguments are given on the command line, standard input 1 is laminated to standard input 2, i.e. "lam" is equivalent to "lam /dev/stdin1 /dev/stdin2". Otherwise, at least one file name argument must be supplied on the command line.

Examples

```
file1> file2> lam >lamination
lam col1 -i\ col2 -i\ /dev/stdin1 | detab -t \
lam -i">>" file >junk
```

See Also

cat (1), tee (1), common (1), field (1), join (1), diff (1), take (1), drop (1)

Usage

```
ld [-(a|b|d|f|h|n|p|u|w)] { <binary file>
                          -c <segment number>
                          -e <segment number>
                          -g <segment name>
                          -l <library file>
                          -m [ <map options> ]
                          -i
                          -t
                          -s <loader command> }
                        [ -o <output file> ]
```

Description

'Ld' is used to call the Primos loader (SEG) from the Subsystem.

The following global options indirectly affect the production of loader commands:

-a Modify the load sequence to include run-time support for Pascal programs. This option may be used with '-b' and '-p' for mixed-language programs.

-b Modify the load sequence to include run-time support for C programs. (The load of the C main program is triggered by the appearance of the first binary file or library.) This option may be used with '-a' and '-p' for mixed-language programs. Besides loading the C run-time library "ciolib", this option automatically loads the SWT math library, "vswtmath", and the shared shell library, "vshlib".

-d Produce a SEG-compatible segment directory rather than P300 memory image. This option must be used with the source-level debugger (DBG) or when more than 64K of memory must be initialized when a program is loaded (usually Fortran programs with block data subroutines).

-f Generate a full load map after commands are complete. The name of the map file will be the same as the name of the output file with the ".o" suffix (if any) replaced by ".m". This option performs the same action as the options "-t -m" at the end of the argument list.

-h Suppress the inclusion of the "mix" command in the load sequence, so that procedure and

linkage will be loaded in different segments.

-n Do not include the high-memory common blocks or load the default libraries unless the '-i' and '-t' options are encountered. This allows the loading of non-Subsystem programs or the insertion of additional loader commands at the beginning and end of the load.

-p Modify the load sequence to include run-time support for PL/I subset G programs. This option may be used with '-a' and '-b' for mixed-language programs.

-u Generate a load map of undefined symbols after the default libraries have been loaded.

-w Modify the load sequence to include run-time support for Prime C programs.

The following local options are examined in the order presented and directly produce commands to the loader:

<binary file> specifies a binary code file to be loaded.

-c <segment number> cause subsequent common blocks to be loaded in the specified segment. By default, common blocks are loaded into segment 4001 (Fortran, Ratfor) or segment 4000 (PL/I G).

-e <segment number> specifies the default segment number for a load using the "-v" option. The segment numbers used for the <binary file>, -l <library file>, and -t directives are affected. This option normally has use only when a shared, multi-segment program is being loaded.

-g <segment name> causes up to 28 characters specified as <segment name> to be used for the names of the segments produced from a load using the "-v" option. The default <segment name> is "..". This option normally has use only when a multi-segment, shared program is being loaded.

-l <library file> specifies a library file to be loaded.

-s <Primos loader command> allows arbitrary loader commands to be inserted in the command stream

-m <map options> presents a map command to the

loader. If <map options> is omitted, the first "<binary file>.m" is assumed. (If <binary file> ends with ".b", the "b" is replaced with an "m".)

-i causes the inclusion of the initial sequence of Subsystem program loader commands (the definition of Subsystem common block locations and default segment for user common blocks) to be included, regardless of the "-n" global option.

-t causes the inclusion of the terminal sequence of Subsystem program load commands (the default library loads) to be included, regardless of the "-n" global option. If the "-n" option is not specified, the sequence of commands will be included at this point, so that loader commands may be inserted after the libraries have been loaded. This option may be used with the "-m" option to generate a full load map.

-o <output file> specifies the output file for the results of the load. If omitted, the first "<binary file>.o" is assumed. (If <binary file> ends with ".b", the "b" is replaced with an "o".)

Commands are presented to the loader in the order in which they are encountered in the command line, except for "-o", which appears only at the end of the command stream.

Examples

```
ld -du rf.b -t -m
ld sol.b -l vthlib -o sol
ld sh.b -s "sy kp$swt 165035" -o sh
```

Bugs

'Ld' pays no attention to standard ports.

If the "-d" option is not present, 'ld' must be able to create files in the current directory.

All files specified must be disk files.

See Also

fc (1), pc (1), plgc (1), f77c (1), pmac (1), x (1), rfl (1), xcc (1), xccl (1), bind (3)

Usage

```
length  [ <string> ]
```

Description

'Length' may be used to determine the length (in characters) of a string or of all the lines in standard input. If an argument is specified on the command line, its length is printed on 'length's first standard output port; otherwise, lines are read from the first standard input port until end-of-file, with the length of each being printed after reading. When lines are taken from standard input, the NEWLINE at the end of each line is not included in the printed length.

Examples

```
length [login_name]
lf -c | length | stats -hlq
```

See Also

substr (1), take (1), drop (1), rot (1), length (2), substr (2), stake (2), sdrop (2)

Usage

```
lf { -<option>{<option>} } { <pathname> }
   <option> ::=  a | c | d | f | k | l | n
                 | o | q | r | t | u | v
                 | <sort>m | <sort>p | <sort>w | s<depth>
   <sort>   ::=  [ \ | / ]
   <depth>  ::=  [ <positive integer> ]
```

Description

'Lf' prints information about files. Its primary function is to list the names of all files within specified directories; however, other information is also available under control of the options. Each option is specified by a single letter, as follows:

a list files whose names begin with the character '.'. These files are occasionally used for long-term storage of data, and many people prefer that they not appear in directory listings, so they are not listed by default.

c force all output to be left justified in a single column at the left margin. This option is generally used for producing output that is to be processed further by other programs.

d treat the directories named in the argument list as ordinary files. 'Lf' normally prints information about the contents of named directories; this option causes it to print information about the directories themselves.

f print full pathname (or as much of it as is known) for each file name printed. This option is frequently used when the output of 'lf' is to be processed by other programs, since it makes the output filenames independent of the current position in the file system.

k print the value of the read/write lock associated with the file. This lock is used to control concurrent access to the file by multiple users. Possible values are:

sys system default value is used. At most installations this is equivalent to "n-1" (see below).

n-1 allow multiple readers or one writer.

n+1 allow multiple readers and one writer.

n+n allow multiple readers and multiple writers.

l select options k, m, o, p, t, u, and w. See below for details on these options.

m print time and date of last modification for each file listed.

n inhibit sorting of output. See below for a discussion of sorting.

o print the owner password for each file listed. Note that the only files that have passwords are directories; a field of blanks is printed for non-directory files.

p print the protection mode of each file listed. The protection mode is represented as a string consisting of two fields separated by a "/". The characters to the left of the "/" indicate the mode that applies to users with owner access to the file, while those to the right indicate the mode that applies to users with non-owner access to the file. The three types of access are "truncate" (or "delete"), "write" and "read", represented by the characters "t", "w" and "r" respectively. The presence of any of these characters in the protection mode string indicates that the associated type of access is allowed. If all three types of access are allowed, the character "a" appears instead of "twr".

q print the non-owner password for each file listed. Note that the only files that have passwords are directories; a field of blanks is printed for non-directory files.

r reverse sort. The direction of sorting is reversed. Thus, an ascending sort becomes descending, and vice versa. See below for a discussion of sorting.

s print information about an entire subtree of the file system. When this option is used, 'lf' interprets each <pathname> argument as the name of the root node of a subtree of the file system. Each file in the named directory is listed; when a subdirectory is encountered, 'lf' descends into it and recursively lists its contents. This process is repeated until all files and directories in the subtree have been listed. The depth to which 'lf' will descend in traversing the subtree may be limited by appending a positive integer to the "s"; 'lf' will then descend no more than that many levels below the named directory. The other options may be used to select the information printed for each file. The current level of descent is indicated by indentation; 'lf' indents three spaces each time it descends into a subdirectory. Indentation may be suppressed by specifying the "c" option.

t print the file type of each file listed. The file type is a three character string with possible values as follows:

sam the file is organized according to the Sequential Access Method.

dam the file is organized according to the Direct Access Method.

sgs the file is a segment directory organized as a "sam" file.

sgd the file is a segment directory organized as a "dam" file.

ufd the file is a (user file) directory.

spc the file is a "special" file, such as the master file directory (mfd), the disk record availability table (dskrat), the system boot file (boot) or the bad-record file (badspt).

??? the file type cannot be ascertained.

u print the status of the "dumped" and "modified" flags associated with each file listed. The "dumped" flag may be set by a program such as a file archiver to indicate that a backup copy of the file exists. The "modified" flag is set by Prime's single user operating system (DOS) whenever the file is modified to indicate that the modification date is inaccurate. (DOS doesn't maintain modification dates.) Primos automatically resets both flags whenever the file is modified. The "dumped" flag is represented by a "d" and the "modified" flag by an "m". If a flag is on, its associated character is printed; otherwise, a blank is displayed.

v for each directory named in the argument list, print a header containing the directory's pathname before listing its contents.

w print the file size (in 16-bit words) for each file listed. The number printed is the number of data words contained in the file; it does not include, for example, the cumulative sizes of files contained within segment directories or UFDs.

If no <pathname> arguments are given, 'lf' assumes you want a listing of the contents of the current working directory.

The default listing format without the "s" option is multi-column sorted alphabetically by name across columns; if, however, the "n" option is selected, no sorting takes place. With the "s" option, sorting is never done and the default

format is one file per line with incremental indentation based on nesting level. In all cases, the "c" option forces one file per line, starting in column 1.

When neither the "n" nor "s" option is used, 'lf' may be asked to sort the output on any of three fields other than the file name: date of last modification ("m" option), protection mode ("p" option), or file size ("w" option). To request one of these, a slash ("/") or backslash ("\") may be prepended to the appropriate option letter; slash causes an ascending sort, backslash, a descending one. Thus, to sort the output in order of most recent modification, one might use

```
lf -\m
```

Examples

```
lf -/wp
lf -m =nbin=
lf -l //allen //dan //perry
lf -s =src=
lf -csf /0 /1 /2 /3 /4 /5 | find pascal
```

Messages

"<pathname>: not found" if specified file or directory doesn't exist.

Bugs

Sorting is performed on at most one field. The ability to sort by protection mode is not very useful.

See Also

chat (1), lacl (1), tscan$ (6)

Usage

line

Description

'Line' prints the user's process id in decimal on standard output.

Examples

```
cat >=temp=/t$[line]
line
to [caller] Process i.d. is [line]
```

Bugs

A better name for this command would be 'pid'.

See Also

login_name (1), term_type (1), date (2)

Usage

```
link [-{f | m}] {-n<filename> | <filename>}
```

Description

'Link' creates a **linkage** statement for the files specified as arguments in the command line. An identifier needs to be in a **linkage** statement if it is longer than six characters and it meets one of the following conditions:

1) The identifier is in an external statement.
2) The identifier is the name of a named common block.
3) The identifier is a subroutine name.
4) The identifier is a function name.

The **linkage** statement produced by 'link' includes all identifiers which are of one of the four types above, regardless of the number of characters in the identifier. Because of this, 'link' creates a list of all external symbols for the modules of a given program as well as a **linkage** statement.

The following options are available:

f Suppress automatic inclusion of standard definitions file. Macro definitions for the manifest constants used throughout the Subsystem reside in the file "=incl=/swt_def.r.i". 'Rp' will process these definitions automatically, unless the "-f" option is specified.

m Map all identifiers to lower case. When this option is selected, 'link' considers the upper case letters equivalent to the corresponding lower case letters, except inside quoted strings.

n Read file names from an input file until EOF is reached. 'link' observes the convention that a "-n" argument implies that file names are to be read from an input file until EOF is reached, rather than simply from the argument list. "-n" implies the standard port STDIN, "-n2" implies STDIN2, "-n3" implies STDIN3, and "-nfilename" implies the named file.

The remainder of the command line is used to specify filenames which are part of the program for which the **linkage** statement is being created.

Examples

```
link -nrpfiles
```

```
link xref.r xref.sort xref.out
```

Files

```
=temp=/tm?* for internal temporaries
=incl=/swt_def.r.i for standard Subsystem macro definitions
```

Messages

See the *User's Guide for the Ratfor Preprocessor* for more information on linkage statements.

See Also

rp (1), sep (1), gfnarg (2)

Usage

locate [-cmd | -sub] { <module_name> }

Description

'Locate' returns the pathname(s) of the source code for a command or subprogram on standard output.

Examples

```
locate -sub getlin
locate lf
```

Files

=src=/misc/srcloc

Bugs

Does not complain if source is not found.

Usage

log [<log file>]

Description

'Log' is used to make entries in one of a number of user logs. When used, 'log' appends the current date, time, and day-of-week to the specified log file and then appends to it the contents of standard input one, up to the next occurrence of end-of-file.

If <log file> is present, it must be the name of a file in the user's variables storage directory; the named file is used as the log file. If absent, the file "u.log" is used.

'Log' is frequently used to make records suitable for use in preparing end-of-the-month time sheets and project diaries.

Examples

```
log time_sheet
log projlog
log
```

Files

=varsdir=/<log_file> for log file.

Bugs

The restriction of having the log file reside in =varsdir= could be considered a bug.

Usage

```
login_name
```

Description

'Login_name' prints the user's login name on standard output one.

Examples

```
cat >//[login_name]/time_sheet
cto >//[login_name]/[arg 1]
login_name
```

See Also

line (1), date (2)

Usage

lorder <object_file>

Description

'Lorder' takes the given object code file and rearranges the object modules to allow loading in one pass by the loader.

Examples

```
lorder =lib=/vthlib
lorder mylib
```

Files

<object_file>.lib is generated

Messages

"Usage: lorder ..." for no object code file arguments

Bugs

Does not complain if more than one object code file is specified, but will only process the first one specified.

See Also

bmerge (5), brefs (5), tsort (1)

Usage

```
lps ( <cancel> | <list> )
<cancel> ::= -c { <seq> }
<list>   ::= { -{d|m|q} | -a <dest> | -p <paper> } { <pack> }
```

Description

'Lps' allows the user to cancel entries from his home spool queue, or to list the contents of any spool queue in the system.

To cancel entries, the "-c" argument is followed by the entry numbers to be cancelled. If an entry does not belong to the user, 'lps' prints an error message and leaves the entry intact.

In the absence of the "-c" argument, 'lps' lists the contents of the spool queues on the specified disk packs, interpreting the remaining arguments as listing constraints as follows:

-a list only entries that will be printed at the specified destination.

-d list only entries that are deferred.

-m list only entries that belong to the current user.

-p list only entries that will be printed on the specified type of paper.

-q print a "quiet" listing that contains no heading and lists only the sequence number, user name, size, destination and file name of each entry selected. (Note: this option merely controls the format of the listing and has nothing to do with which entries are selected.)

If multiple constraints are specified, only entries that satisfy all constraints are listed. If no constraints are specified, the entire queue is listed.

Examples

```
lps
lps -c 1 5 prt10
lps -a lpb -p narrow -q  sa sb sc
```

Files

```
/<pack>/spoolq/q.ctrl queue control file
//spoolq/prt???  print files
```

Messages

"Usage: lps ..." for improper command syntax.
"Can't find SPOOLQ directory on disk <pack>" if the specified disk partition is inaccessible or does not contain a spooler queue.
"Can't read queue on disk <pack>" if the spooler queue on the specified disk can't be opened for reading.
"<seq>: bad sequence number" for illegal syntax in specifying a print file.
"<print_file>: in use" for trying to cancel a print file that is either being printed or still being written.
"<print_file>: can't cancel" when an unexpected error occurs while cancelling a print file.
"<print_file>: not found" for trying to cancel a nonexistent print file.
"<print_file>: not yours" for attempting to cancel someone else's print file.

See Also

sp (1), pr (1)

Usage

macro [-e]

Description

'Macro' is Kernighan and Plauger's macro preprocessor from Chapter 8 of *Software Tools*. 'Macro' is an exceedingly powerful program; it is theoretically possible to use it as a general programming system. A complete description of its capability is beyond the scope of this document, but a few samples are presented here to help the user become proficient in its usage.

'Macro' is a filter; it takes input from its standard input file, expands all macros it encounters, and places the output on its standard output file. This behavior strongly encourages its use in pipelines.

The basic format of a macro definition is:

```
define(macro-name, replacement-text)
```

"Macro-name" is an identifier, i.e. a sequence of letters or digits beginning with a letter. "replacement-text" is a (possibly empty) sequence of characters, which may be specially interpreted by 'macro'.

The "-e" option allows for the escaping of characters that "macro' would normally use as delimiters (e.g. commas, right parenthesis, etc.). To escape a character, it must be preceded by the escape character "@". 'Macro' discards the escape character and treats the escaped character as a normal character with no special meaning. Since 'macro' discards the escape character, in order to get a literal "@" it must be escaped ("@@").

Macro arguments are referred to by a construct of the form "$<integer>" in the replacement text. The <integer> must be a digit from 0 to 9, inclusive. (Digits 1-9 represent the first through the ninth arguments; digit 0 represents the name of the macro itself). For example, the following macro could be used to skip blanks and tabs in a string, starting at a given position:

```
define(skipbl,
   while ($1 ($2) == ' 'c | $1 ($2) == TAB)
      $2 = $2 + 1
   )
```

Here are a few examples of the use of this macro:

```
skipbl(line, i)
skipbl(str, j)
```

In order to prevent premature evaluation of a string, the string may be surrounded by square brackets. For example, suppose we wished to redefine an identifier. The following sequence will not work:

```
define(x,y)
define(x,z)
```

This is because "x" in the second definition will be replaced by "y", with the net result of defining "y" to be "z". The correct method is

```
define(x,y)
define([x],z)
```

The square brackets prevent the premature evaluation of "x".

'Macro' provides several "built-in" functions. These are given below:

divert(filename) or divert(filename,append) or divert
"Filename" is opened for output and its file descriptor is stacked. Whenever 'macro' produces output, it is directed to the named file. If the second argument is present, output is appended to the named file, rather than overwriting it. If both arguments are missing, the current output file is closed and output reverts to the last active file.

dnl or dnl(commentary information)
As suggested by Kernighan and Plauger, 'dnl' may be used to delete all blanks and tabs up to the next NEWLINE, and the NEWLINE itself, from the input stream. There is no other way to prevent the NEWLINE after each 'define' from being passed to the output. Any arguments present are ignored, thus allowing 'dnl' to be used to introduce comments.

ifelse(a,b,c,d)
If "a" and "b" are the same string, then "c" is the value of the expression; otherwise, "d" is the value of the expression. Example: this macro returns "OK" if i is defined to be "1", "ERR" otherwise:

```
define(status,ifelse(i,1,OK,ERR))
```

include(filename)
"Filename" is opened and its file descriptor is stacked. The next time 'macro' requests input, it receives input from the named file. When end-of-file is seen, 'macro' reverts to the last active input file (the one containing the last include) and picks up where it left off.

incr(n)
increment the value of the integer represented by "n", and return the incremented value. For instance, the

following pair of defines set MAXCARD to 80 and MAXLINE to 81:

```
define(MAXCARD,80)
define(MAXLINE,incr(MAXCARD))
```

substr(s,m,n)
: return a substring of string "s" starting at position "m" with length "n". substr(abc,1,2) is ab; substr(abc,2,1) is b; substr(abc,4,1) is empty. If "n" is omitted, the rest of the string is used: substr(abc,2) is bc.

undefine(name)
: 'Undefine' is used to remove the definition associated with a name. Note that the name should be surrounded by brackets, if it is supplied as a literal, otherwise it will be evaluated before it can be undefined. Example: undefine([x]), undefine([substr])

Examples

See *Software Tools*.

Files

None used by 'macro' itself; the builtins 'include' and 'divert' may be used for limited file manipulation.

Messages

Extensive. See *Software Tools*.

Bugs

Blanks are not allowed between the macro name and its argument list.

See Also

rp (1), include (1), *Software Tools*

Usage

```
mail [ -p ] { <login name> }
```

Description

'Mail' is the user's interface to the Subsystem postal service.

If invoked with arguments, 'mail' first verifies that each is the login name of a Subsystem user, reads standard input one until end-of-file, and then appends the message thus read to the mailbox files of all users named. All letters are postmarked with the sender's login name and the time and date of the mailing.

If no argument is present on the command line, the user's own mailbox file is displayed. If the "-p" option is not present and standard output is directed to the user's terminal, letters are printed one CRT screenful at a time. (The user may skip or re-examine the mail at this point; see manual entries for 'pg' (1) and 'page' (2) for further information.) If anything was in the mailbox, 'mail' then asks the question, "Save mail?". If the response begins with the letter "n", the mail is discarded; otherwise, the contents of the mailbox are appended to the file named by the template "=mailfile=" (Subsystem default is =varsdir=/.mail).

Examples

```
mail
mail spaf
   (message follows, terminated by end-of-file (Control-C))
mail perry dan myers
   (message follows, terminated by end-of-file (Control-C))
```

Files

```
=mail=/<login_name> for mailboxes
=mailfile= for mail save file
```

Messages

"Usage: mail ..." for invalid arguments.
"Save mail?" to ask if mail should be saved.
"can't create temporary file" if a temporary file can't be created to hold the letter for distribution.
"can't open <user>'s mailbox" if the mail delivery file for <user> can't be opened.

Bugs

Mail messages are neither secure nor private.

See Also

to (1)

Usage

mkdir <pathname> [-o <owner>] [-n <non_owner>]

Description

'Mkdir' is used to create a new directory. The pathname given as the first argument is the pathname of the directory; all nodes but the last must exist prior to the invocation of 'mkdir'. The "-o" and "-n" keyword arguments may be used to specify the owner and non-owner passwords to be given to the new directory. If they are omitted, default values are assumed as follows: at installations running the Georgia Tech version of Primos, the user's login name is used for the owner password and the non-owner password is set to zero; at installations running unmodified Primos, the owner password is set to blanks and the non-owner password is set to zero.

Examples

```
mkdir subsys
mkdir subdir -o allen
mkdir //may-78/twob -n secret
```

Messages

"Usage: mkdir ..." for missing directory name or bad arguments

"<pathname>: can't create" if directory already exists or the path to it cannot be followed

See Also

lf (1), passwd (3), del (1)

Usage

mklib <file>

Description

'Mklib' runs the Primos EDB program to convert the relocatable object code output from FTN, PMA, or other compilers contained in the file named <file>.b into a library format file in the file named <file>.

For example,

mklib swtlib

would convert the contents of the file named "swtlib.b" into library format and write the result on the file named "swtlib".

Examples

mklib swtlib

Messages

Several possible messages from EDB.

See Also

Primos EDB command

Usage

mktree { <pathname> }

Description

'Mktree' converts Subsystem pathnames into standard Primos treenames. If arguments are supplied, each is interpreted as a pathname and the results of conversion are printed (one per line) on standard output. If no arguments are supplied, pathnames are read (one per line) from standard input until EOF, with the conversion results again being printed one per line on standard output.

Examples

```
mktree //bozo/file
x spool [mktree [arg 1]]
```

See Also

mktr$ (6), mkpa$ (2)

Usage

```
mt [<unit>] [-p<pos>] [-(r|w) [<cvt>] [<blk>] {<file_spec>}] [-v]
   <unit>      ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
   <pos>       ::=   [+|-]<file number>[/<block number>]
   <cvt>       ::=   -c (a[scii] | b[inary] | e[bcdic])
   <blocking>  ::=   -b <record size>[/<blocking factor>]
```

Description

'Mt' is a program designed to provide a general purpose magnetic tape handling facility to users of the Subsystem. It supports three basic types of operation: tape positioning, reading files from tape, and writing files to tape. It is also possible to perform both a positioning operation and a read or write operation in a single invocation.

The first argument may be used to specify a particular tape drive. The allowable values are integers from 0 through 7, although a particular installation may not support that many drives. If no unit is specified on the command line, unit 0 is assumed. Whatever unit is used, it should have been previously assigned by the user with the Primos ASSIGN command.

The remaining arguments select one of the basic operations to be performed on the specified drive. The available options are described in the following paragraphs.

-p The "-p" option may be used to accomplish either relative or absolute positioning of the tape. The argument following the "-p" consists of an optional plus or minus sign followed by either a <file number> or a <file number> and a <block number> separated by a slash (/).

If the plus or minus sign is present, relative positioning is selected; the <file number> specifies the offset from the current file of the target file. Thus "+1" would position the tape to the beginning of the file immediately following the current one, while "-1" would position the tape to the immediately preceding file. If a block number is present, the specified number of blocks are skipped in the same direction. As a special case, if a minus sign is present and both the <file number> and <block number> are zero, the tape is positioned to the beginning of the current file.

If no sign is present, absolute positioning is selected; the <file number> is taken as the target file's ordinal position on the tape, where the first file has position 1, and the <block number> is taken as the ordinal position of the desired block within the target file.

In positioning the tape, 'mt' only considers physical tape marks; it specifically does not recognize any kind of labels in determining where a file begins and ends.

-r The "-r" option causes 'mt' to read files from the specified tape drive. The "-r" may optionally be followed by conversion and blocking specifications. <Cvt> specifies what kind of character set conversion is to be performed on the data read from the tape: "-c a" indicates that the characters on the tape are in ASCII, "-c e" indicates that they are in EBCDIC, and "-c b" indicates that they are arbitrary binary codes and are not to be interpreted as characters at all. If no <cvt> is specified, ASCII is assumed. The Prime convention for text files is to store characters with the most significant bit set to 1, whereas most ASCII encoded tapes are written with this bit set to 0. 'Mt' automatically turns this bit on when reading ASCII tapes, and turns it off when writing them.

<Blk> specifies how the physical blocks from the tape will be broken up into lines before being written out. This argument is significant only if the specified conversion is ASCII or EBCDIC; binary records are written out as-is, regardless of whatever <blk> specification may be in effect. If omitted, a default value of "80/10" is used; that is, 80 bytes per line, 10 lines per physical tape block. Although this implicitly suggests that physical tape blocks are 800 bytes long, 'mt' will read any size tape block (up to 6K bytes for ASCII and EBCDIC conversion, up to 12K bytes for binary conversion) and divide it into lines according to the specified <record size>. For ASCII and EBCDIC tapes, each line is stripped of trailing blanks and terminated with a NEWLINE character before being written out to its final destination.

-v The "-v" option is used to make 'mt' verbose, it will tell you how many blocks it read from or wrote to the tape.

-w The "-w" option is syntactically identical to the "-r" option. The <cvt> specification may be used to specify what character set will be used in writing the tape, and the <blk> specification determines the size of the blocks written. 'Mt' writes fixed size tape blocks, the size of which is determined by the product of <record size> and <blocking factor>. If the specified conversion is ASCII or EBCDIC, input lines that are shorter than <record size> are padded out to that length with blanks after having their NEWLINE character removed. As with "-r", binary blocks are not divided into lines. In any case, if the end of the input file is reached before a complete block has been constructed, the remaining bytes are filled with zeros (for binary conversion) or blanks (for ASCII or EBCDIC con-

version).

The remaining command line arguments are taken as names of files to be read from or written to the tape. The full syntax of the <file_spec> argument is described in the entry for 'cat' (1). Most frequently, it will take the form of a Subsystem pathname.

Examples

```
mt -p 1
mt 1 -w tape_file
mt -r -ce -b120/30 file1 file2 file3
cat file | mt -w
```

Messages

"Usage: mt ..." for incorrect argument syntax.
"syntax: -b <record size>[/<blocking factor>]" for incorrect blocking arguments.
"syntax: -c (a[scii] | b[inary] | e[bcdic])" for incorrect conversion arguments.
"syntax: -p [+|-]<file number>[/<block number>]" for incorrect positioning arguments.
"maximum block size is <max> bytes" if the requested block size exceeds the maximum.
"units are <low> to <high>" if an illegal unit number is specified.
"drive is not ready" if the specified unit is not ready.
"drive is off line" if the specified unit is not on line.
"tape is at end of reel" if the tape mounted on the specified unit is positioned beyond the end-of-tape marker.
"tape is in mid-file" if an attempt is made to write on a tape that is neither at the load point or at a file mark.
"tape is write protected" if an attempt is made to write on a tape that has no write ring.
"<file>: bad file name" if <file> begins with a dash.
"<file>: can't create" if <file> can't be opened for writing.
"<file>: can't open" if <file> can't be opened for reading.
"<file>: <num> blocks read from tape" when using the "-v" option.
"<file>: <num> blocks written to tape" when using the "-v" option.
"Block <n>: <error status> Unrecovered" for an unrecovered tape i/o error on the <n>th block, resulting from <error status>.
"beginning of file" if an attempt is made to do backward relative block positioning beyond a file mark.
"beginning of tape" if an attempt is made to do backward relative positioning beyond the load point.
"end of file" if an attempt is made to do forward relative

block positioning beyond a file mark.
"end of tape" if an attempt is made to position beyond the end-of-tape marker.

See Also

cat (1), Primos MAGNET command, Primos t$mt

Usage

nargs [<level_offset>]

Description

'Nargs' prints the number of arguments supplied on a command line at some higher level of command file/function call nesting. It is most often used in a function call within a command file to determine the number of arguments supplied to that same command file.

As with the 'arg' and 'args' commands, <level offset> may optionally be specified to indicate the number of higher nesting levels to skip before counting. In keeping with its most frequent mode of usage, the default value is one, so that the nesting level corresponding to the function call is ignored.

Examples

```
nargs 0
echo [nargs]
```

See Also

arg (1), args (1), getarg (2)

Usage

news [-p] { -i | <item_number> }

Description

'News' gives Subsystem users access to the Software Tools Subsystem news service. It has three basic functions:

1. To print an index of currently active news items.

   The "-i" option is available to perform this function. The command "news -i" will print the index. Each entry in the index is of the form:

   <item_number> <date> <time> <headline>

   The <item_number> is an integer which may be used to select specific articles to be printed (see below). The <date> and <time> are the date and time at which the item was published. (See the documentation for the 'publish' command.) The <headline> is a short description of the contents of the news item.

2. To print selected news items.

   For each <item_number> (see above) specified in its argument list, 'news' will print a corresponding news item on standard output. Available news items may be determined by looking at the index generated by the "news -i" command.

3. To print the news delivered to a subscriber.

   Users may "subscribe" to the news service by using the 'subscribe' command. Whenever a subscriber logs in to the Subsystem (either at Primos login or through the 'swt' command), he is informed if any news item has been published since he last checked with the news service. If news is available, he should type the command "news", without arguments. Recent news items will be printed, one CRT screenful at a time. (The user may skip or re-examine the news at this point; see manual entries for 'pg' (1) and 'page' (2) for further information.) The user is then asked whether or not he wishes to save his news. The correct response is "n" or "N" for "no"; anything else causes the news to be saved. News not saved may still be retrieved through the usual channels outlined in steps 1 and 2 above.

If the user does not specify the "-p" option and standard output is directed to his terminal, 'news' will display the requested articles one page at a time. Otherwise, 'news' will produce its output in a continuous stream.

Examples

```
news -p -i
news 22 23 24
news
```

Files

=news=/articles/art<number> for archived articles
=news=/index for article index
=news=/delivery/<login_name> for delivery to subscribers
=news=/subscribers for a list of subscribers

Messages

"Usage: news ..." for invalid argument syntax.
"article <number> could not be found" for unknown article number.

See Also

publish (1), retract (1), subscribe (1), pg (1), page (2)

Usage

os { -l <page length> | -x }

Description

'Os' is a filter that may be used to convert backspaces (such as those produced by the formatter for underlining and boldfacing) into standard Fortran line printer carriage control codes.

If the output of 'os' is spooled, the Fortran forms control mode must be in effect. Use of the "f" option on the 'sp' command or the "f" option in a "/dev/lps" pathname (e.g. "/dev/lps/f") will enable Fortran forms control.

If the "-x" option is included, 'os' will attempt to generate output for a Printronix printer. We are told that these printers can overprint only a single line, and the characters on that line can only be underscores. Under "-x", 'os' emits only the overstriking that can be performed on these printers.

'Os' will generate a page-eject at the bottom of each page (to keep the pages correct in case of a paper jam). The <page_length> is the number of lines per output page. If <page_length> is omitted, 'os' assumes 66 (standard paper).

Examples

```
fmt report | os | sp / f
junk> os >/dev/lps/f/bjunk
```

Messages

"Usage: os ..." for invalid argument syntax.

See Also

sp (1), fos (1)

Usage

```
case <value>
   when <alternative1>
      { <command> }
   when <alternative2>
      { <command> }
   ...
   out
      { <command> }
esac
```

Description

'Out' is used to flag the default alternative in a 'case' command sequence. It should appear after any command sequences introduced by 'when' commands, and will be selected by the 'case' command if and only if none of the alternatives specified by 'when' commands are taken.

'Out' is usually executed only if control falls through from the commands under the control of a 'when'. In this instance, commands are skipped until an unmatched 'esac' command is found.

Use of 'out' from a terminal may cause input to be ignored until end-of-file or the typing of an 'esac' command.

Examples

```
case [line]
   when 12
      set location = REMOTE
   out
      set location = LOCAL
esac
```

Messages

"Missing 'esac'" if end-of-file is encountered before an 'esac' command.

Bugs

'Out' is a holdover from the ALGOL 68 case-clause syntax.

See Also

case (1), when (1), esac (1), if (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

pause ([for] <interval> [<units>] | until <time>)

Description

'Pause' causes a user's traffic with the system to cease for a fixed interval of time or until a specific wall clock time.

In the first usage format, <interval> is the number of time units to pause, expressed as a positive decimal integer. It must be less than 32768. <Units> specifies the time unit. It may be:

"seconds" for seconds,
"minutes" for minutes,
"hours" for hours,

or omitted, in which case "seconds" is assumed. Abbreviations consisting of any initial substring of the above units are allowed. The word "for" may be included to enhance readability; its presence or absence is otherwise insignificant.

In the second format, traffic will be suspended until the system clock registers the time of day specified by <time>. <time> may be expressed in almost any common format. One guideline should be observed, however: a colon must be used to separate hours from minutes and minutes from seconds.

Examples

pause 5 seconds
pause for 2 hours
pause until 3pm
pause until 18:45:30

Messages

"Usage: pause ..." for invalid argument syntax.

See Also

sema (1), date (2), Primos sleep$

Usage

```
pc {-<option>[<level>]} <input file>
          [-b [<binary file>]]
          [-l [<listing file>]]
          [-z <PASCAL option>]
   <option> ::= c | d | e | f | h | k | m | n |
                o | q | r | s | u | v | w | x
```

Description

'Pc' serves as the Subsystem interface to the Primos Pascal compiler (PASCAL). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos PASCAL command and causes it to be executed.

Options

The general structure of an 'pc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-c(0) [0..1] - Case.

Level 0 forces case to be insignificant in identifiers. Upper case identifiers are considered the same as lower case identifiers.

Level 1 cause case to significant in identifiers. Upper case identifiers are considered different from lower case identifiers.

-d(0) [0..2] - Debugging control.

Level 0 prevents all debugging information from being included in the generated code. A program so compiled may not be used with the source level debugger.

Level 1 allows limited debugging information to be included in the generated code, but does not interfere with optimization.

Level 2 causes complete debugging information to be included in the generated code and inhibits optimization. (Cannot be used when the "-o" option is

specified with a level greater than zero.)

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(2) [0..3] - Symbol table map and offset map control.

Level 0 inhibits the generation of either a symbol table map or a storage offset map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 1 causes the generation of a map listing the storage offset of each program variable, but still inhibits the generation of a a symbol table map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 2 causes the generation of a map listing the symbol names appearing in the program, but inhibits the generation of a storage offset map.

Level 3 causes the generation of both the symbol table and storage offset maps.

-h(0) [0..1] - Huge (multi-segment) arrays.

Level 0 insures that dummy arrays and array parameters will not be treated as multi-segment arrays.

Level 1 causes references to dummy arrays and array parameters to generate code that will work even if the arrays are larger than one segment (64K words) in length.

-k(0) [0..1] - Compilation statistics.

Level 0 inhibits the display of compilation statistics on the terminal.

Level 1 causes the display of compilation statistics on the terminal.

-m(2) [2..3] - Addressing mode.

Level 2 implies 64V addressing mode. At present this is the only addressing mode fully supported under the Subsystem.

Level 3 implies 32I addressing mode. Code in this addressing mode will not execute on a Prime 400.

-n(1) [0..1] - Nesting level indicator.

Level 0 inhibits the printing of the nesting level of each statement on the listing.

Level 1 causes the printing of the nesting level of each statement.

-o(1) [0..1] - Optimization control.

Level 0 turns off all optimizations.

Level 1 turns on optimizations. This option cannot be used with full debugging (-d2).

-q(1) [0..1] - Suppress warning messages.

Level 0 inhibits the display of compiler warning messages.

Level 1 allows the display of compiler warning messages.

-r(0) [0..1] - Range checking.

Level 0 inhibits run-time checking of subscripts and substrings.

Level 1 causes the compiler to insert code for the run-time checking of subscripts and substrings.

-s(0) [0..1] - Check for use of non-standard features.

Level 0 allows all features of Prime Pascal.

Level 1 generates a syntax error for the use of any feature not in the proposed ANSI standard.

-u(0) [0..1] - Generate external procedure definition.

Level 0 does not generate an external procedure definition.

Level 1 generates an external procedure definition.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-w(0) [0..1] - Generate floating round instructions.

Level 0 does not generate floating round (FRN) instructions.

Level 1 cause a floating round (FRN) instruction to be generated before every floating store (FST) instruction in the code produced by the PASCAL compiler. This option improves the accuracy of single precision floating point calculations at some slight run-time performance expense.

-x(1) [0..1] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing. (Cannot be used when the "-f" option is specified with a level less than two.)

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.p", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.p", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".p" or ".pascal") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a

file named "file.p" is

```
pc -c0d0e1f2h0k0m2n1o1q1r0s0u0v1w0x1 _
     file.p  -b file.b  -l /dev/null
```

which corresponds to the PASCAL command

```
pascal -i *>file.p -b *>file.b -l no
```

Examples

```
pc file.p
pc -kf dmach.p
pc -x dmach.p -b b_dmach -l l_dmach
pc -m3 i_mode_prog.p -z"-newopt"
```

Messages

"Usage: pc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'pc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Pc' pays no attention to standard ports.

See Also

ld (1), pcl (1), file$p (2), geta$p (2), init$p (2), bind (3)

Usage

pcl <program name> [<'ld' options>] [/ <'pc' options>]

Description

'Pcl' is a shell file that invokes the Primos Pascal compiler and the Primos segmented loader. If 'pcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".p", although in <program name> it may be specified with or without the ending ".p". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'pc' will be called with the <'pc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
pcl myprog.p
pcl myprog subs.b subs2.b -l mylib
pcl myprog / -ok -l mylist
```

Messages

"<program name>.p: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

pc (1), ld (1), init$p (2), bind (3)

Usage

```
pg [-e] [-v] [-s <screensize>] [-m <message>] {<file_spec>}
<file_spec> ::= <filename> | -[<stdin_number>] |
                -n(<stdin_number> | <filename>)
```

Description

'Pg' is a filter which displays the contents of a disk file in paginated form. It allows skipping pages forward and backward as well as searching for patterns within the file. 'Pg' is primarily intended for viewing a file on a high speed CRT, but it may be used from any terminal.

'Pg' displays the named files (see 'cat' for further information on <file spec>s) by calling the library routine 'page', which accepts the following responses:

f <path>	Display the file whose pathname is <path>.
f	Redisplay the original file.
h	Print a command summary.
l<lines>	Set screen size to specified number of lines. Display starts over on page 1.
n	Proceed to next file (exit if on last file).
p<pages>	Display given number of pages (default 1), prompting only after the end of the range.
q	Proceed to next file (exit if on last file).
x	Exit immediately from 'pg'.
y	Advance to the next page (proceed to next file if on last page).
ctrl-c	Exit immediately from 'pg'.
newline	Advance to the next page (proceed to next file if on last page).
<page>	Display specified page number.
-<pages>	Back up given number of pages (default 1).
^	Redisplay previous page.
.	Redisplay current page.
+<pages>	Advance given number of pages (default 1).
$	Display the last page.
/<pat>[/]	Display the next page containing <pat>.
\<pat>[\]	Display the previous page containing <pat>.

The pattern <pat> is a regular expression with the full set of options found in the editor. The file is searched circularly from the current position for the next page that contains the specified pattern. As in the editor, the trailing delimiter is optional. (See *Introduction to the Software Tools Text Editor* in the *Software Tools Subsystem User's Guide* for details.)

By default, 'pg' prompts after each page with a string of the form

```
file [n+]?
```

and after the last page with a string of the form

file [n$]?

if the '-e' command line argument is not specified. "File" is the name of the file being displayed, and "n" is the page number within the file. If the '-e' argument is specified, 'pg' will not issue a prompt after the final page of a file, but instead it proceeds to the next file in the argument list (if any). The '-m <message>' argument sequence may be used to specify a prompt string different from the default; this string is used as both the intermediate and final prompt. For details on how this string is interpreted, see the entry for 'page' in section 2.

'Pg' normally displays each file using the 'vth' subroutine package to manage the screen. If the current terminal type is not one of those that 'vth' supports, or if the '-v' argument is specified, then 'pg' displays each file using ordinary sequential output.

The user can inform 'pg' of the number of lines on his terminal screen with the '-s <screensize>' command line argument. If 'vth' output is used, 'pg' takes advantage of the fact that 'vth' knows the size of the screen, and uses all available lines to display the file. In this case the '-s' argument is ignored. If 'vth' output is not used, and the '-s' argument is omitted, 'pg' uses a default value of 23 lines.

Examples

```
pg -s 5 file
fmt english | pg
help -i | pg -m "continue or quit? "
```

Messages

"Usage: pg ..." for invalid argument syntax.

Bugs

The "h" command output is not paged.

See Also

cat (1), copy (1), print (1), page (2), vt?* (2), *Introduction to the Software Tools Text Editor*

Usage

ph { <command> }

Description

The 'ph' command allows the Subsystem user to execute Subsystem commands in the background while continuing with other work at his terminal. The phantom user feature of the Primos operating system is used to implement this command and Primos must have been configured at startup for phantom users.

'Ph' has two usage formats:

In the first format, the commands to be executed are given as arguments. Care should be taken when using this format to enclose in quotes any commands that contain the following characters:

( ) [ ] { } # , > |

since these meta-characters will otherwise be interpreted by the shell applied to the 'ph' command itself.

In the second format, commands are read from standard input up to the next occurrence of end of file. This format allows 'ph' to be used at the end of a pipeline.

In either case, 'ph' builds a script of commands that will be used to drive the phantom process.

Assuming no errors were encountered, 'ph' responds by printing the phantom's process id on standard output.

Examples

```
ph rf se.r
ph "rf rf.r; fc rf.f"
commands> ph
```

Files

=varsdir=/ph<user_number><sequence> for phantom input file

Messages

"=temp= missing" if unable to follow pathname of phantom script.
"No free phantoms" if Primos refuses to initiate phantom.

"Can't create phantom temp" if unable to create file to hold phantom script.

Bugs

A note on portability: 'ph' takes advantage of a Georgia Tech modification to the Primos operating system that duplicates both current and home directories in the environment of the phantom (the normal procedure is to duplicate only the current directory). In systems that do not have this feature, the first command to be executed by the phantom should be a 'cd' command to attach to the desired directory.

Only 4 phantoms may be concurrently in progress on behalf of any single user.

Due to Primos restrictions, phantoms cannot be started while the user is attached to a remote disk.

See Also

sh (1), x (1), batch (1), Primos phant$

Usage

```
phist { -b <author> | -f <date> | -s <subject> | -q }
      [-i <input file>]
<date> ::= <day> | <month>/<day> | <month>/<day>/<year>
```

Description

The purpose of 'phist' is to print selected portions of a history file. The history file chosen by default, "=doc=/hist/history", chronicles the ongoing development and maintenance of the Software Tools Subsystem by its implementors at Ga. Tech. It consists of a series of dated entries, each of which contains the name of the author, a list of commands or files affected, and a description of the modification.

When invoked without arguments, 'phist' simply prints out the entire history file; but several optional argument sequences can be employed to sift out the interesting entries. The "-b <author>" argument sequence may be specified to restrict the entries printed to those written by a given author. The syntax of <author> is the same as that defined for patterns in the Software Tools Subsystem text editors (see the *Introduction to the Software Tools Text Editor* for details).

The "-s <subject>" argument sequence tells 'phist' that only those entries concerning the specified subject should be printed. <Subject> may also be an arbitrary pattern.

The "-f <date>" sequence allows the user to tell 'phist' that he only wants to see entries written on or after a specific date. The format of <date> has three options: if a single integer is specified, it designates a day of the current month; if two integers separated by a slash are specified, they designate a month and day of the current year; finally, if three integers separated by slashes are specified, they designate a specific month, day and year.

If the "-q" option is specified, 'phist' will only print the heading of each selected entry (i.e., the date, author and subject of the entry) and omit the explanatory text. Otherwise, the entire entry is printed.

If the "-i <input file>" is specified, 'phist' takes its input from <input file>, rather than from "=doc=/hist/history".

Examples

```
phist
phist -s %se
phist -f 12/19
```

```
phist -f 1/31/79 -s stacc -b allen
```

Files

=doc=/hist/history for the history of the Software Tools Subsystem.

Messages

"history file not available" if =doc=/hist/history does not exist or is not readable.
"history file contains apocryphal information" if the history file is incorrectly formatted.
"<author>: bad author pattern" if the string following "-b" is not a legal pattern.
"<subject>: bad subject pattern" if the string following "-s" is not a legal pattern.
"<date>: bad date" if the string following "-f" is not recognizable as a date.
"Usage: phist ..." for incorrect argument syntax.

See Also

history (1), *Software Tools Subsystem User's Guide*

Usage

```
plgc {-<option>[<level>]} <input file>
         [-b [<binary file>]]
         [-l [<listing file>]]
         [-z <PL1G option>]
   <option> ::= c | d | e | f | h | k | m | n |
                o | p | q | r | s | v | w | x
```

Description

'Plgc' serves as the Subsystem interface to the Primos PL/I subset G compiler (PL1G). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos PL1G command and causes it to be executed.

Options

The general structure of an 'plgc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-c(0) [0..1] - Case.

Level 0 forces case to be insignificant in identifiers. Upper case identifiers are considered the same as lower case identifiers.

Level 1 cause case to significant in identifiers. Upper case identifiers are considered different from lower case identifiers.

-d(0) [0..2] - Debugging control.

Level 0 prevents all debugging information from being included in the generated code. A program so compiled may not be used with the source level debugger.

Level 1 allows limited debugging information to be included in the generated code, but does not interfere with optimization.

Level 2 causes complete debugging information to be included in the generated code and inhibits optimization. (Cannot be used when the "-o" option is

specified with a level greater than zero.)

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(2) [0..3] - Symbol table map and offset map control.

Level 0 inhibits the generation of either a symbol table map or a storage offset map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 1 causes the generation of a map listing the storage offset of each program variable, but still inhibits the generation of a a symbol table map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 2 causes the generation of a map listing the symbol names appearing in the program, but inhibits the generation of a storage offset map.

Level 3 causes the generation of both the symbol table and storage offset maps.

-h(0) [0..1] - Huge (multi-segment) arrays.

Level 0 insures that dummy arrays and array parameters will not be treated as multi-segment arrays.

Level 1 causes references to dummy arrays and array parameters to generate code that will work even if the arrays are larger than one segment (64K words) in length.

-k(0) [0..1] - Compilation statistics.

Level 0 inhibits the display of compilation statistics on the terminal.

Level 1 causes the display of compilation statistics on the terminal.

-m(2) [2..3] - Addressing mode.

Level 2 implies 64V addressing mode. At present this is the only addressing mode fully supported under the Subsystem.

Level 3 implies 32I addressing mode. Code in this addressing mode will not execute on a Prime 400.

-n(1) [0..1] - Nesting level indicator.

Level 0 inhibits the printing of the nesting level of each statement on the listing.

Level 1 causes the printing of the nesting level of each statement.

-o(1) [0..1] - Optimization control.

Level 0 turns off all optimizations.

Level 1 turns on optimizations. This option cannot be used with full debugging (-d2).

-p(0) [0..1] - Quick call of internal subroutines.

Level 0 causes all internal subroutines to be called with the normal procedure call (PCL) mechanism.

Level 1 causes internal subroutines to be "quick called" (shortcalled) whenever possible. This option cannot be used with full debugging (-d2).

-q(1) [0..1] - Suppress warning messages.

Level 0 inhibits the display of compiler warning messages.

Level 1 allows the display of compiler warning messages.

-r(0) [0..1] - Range checking.

Level 0 inhibits run-time checking of subscripts and substrings.

Level 1 causes the compiler to insert code for the run-time checking of subscripts and substrings.

-s(1) [0..1] - Constant copying for subroutine calls.

Level 0 inhibits the copying of constants into temporary variables for passing as subroutine parameters.

Level 1 causes the compiler to copy constants into temporary variables before calling subroutines.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-w(0) [0..1] - Generate floating round instructions.

Level 0 does not generate floating round (FRN) instructions.

Level 1 cause a floating round (FRN) instruction to be generated before every floating store (FST) instruction in the code produced by the PL1G compiler. This option improves the accuracy of single precision floating point calculations at some slight run-time performance expense.

-x(1) [0..1] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing. (Cannot be used when the "-f" option is specified with a level less than two.)

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.plg", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.plg", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".plg" or ".pl1g") or the device

"/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.plg" is

```
plgc -c0d0e1f2h0k0m2n1o1p0q1r0s1v1w0x1 _
     file.plg  -b file.b  -l /dev/null
```

which corresponds to the PL1G command

```
pl1g -i *>file.plg -b *>file.b -l no
```

Examples

```
plgc file.plg
plgc -kf dmach.plg
plgc -x dmach.plg -b b_dmach -l l_dmach
plgc -m3 i_mode_prog.plg -z"-newopt"
```

Messages

"Usage: plgc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'plgc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Plgc' pays no attention to standard ports.

See Also

ld (1), plgcl (1), geta$plg (2), init$plg (2), bind (3)

Usage

plgcl <program name> [<'ld' options>] [/ <'plgc' options>]

Description

'Plgcl' is a shell file that invokes the Primos PL/I subset G compiler and the Primos segmented loader. If 'plgcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".plg", although in <program name> it may be specified with or without the ending ".plg". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'plgc' will be called with the <'plgc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
plgcl myprog.plg
plgcl myprog subs.b subs2.b -l mylib
plgcl myprog / -ok -l mylist
```

Messages

"<program name>.plg: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

plgc (1), ld (1), init$plg (2), bind (3)

Usage

```
plpc {-<option>[<level>]} <input file>
         [-b [<binary file>]]
         [-l [<listing file>]]
         [-z <PLP option>]
   <option> ::= e | f | q | v | x
```

Description

'Plpc' serves as the Subsystem interface to the Primos PL/P compiler (PLP). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos PLP command and causes it to be executed.

Options

The general structure of an 'plpc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(0) [0..1] - Offset map.

Level 0 inhibits the generation of a storage offset map.

Level 1 cause the generation of a map listing the storage offset of each program variable.

-q(1) [0..1] - Suppress warning messages.

Level 0 inhibits the display of compiler warning messages.

Level 1 allows the display of compiler warning messages.

-v(1) [1..2] - Listing verbosity.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-x(1) [0..1] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing.

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.plp", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.plp", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".plp") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.plp" is

```
plpc -e1f0q1v1x1 _
     file.plp  -b file.b  -l /dev/null
```

which corresponds to the PLP command

```
plp -i *>file.plp -b *>file.b -l no
```

Examples

```
plpc file.plp
plpc -f dmach.plp
plpc -x dmach.plp -b b_dmach -l l_dmach
plpc -e0 r_mode_prog.plp -z"-newopt"
```

Messages

"Usage: plpc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'plpc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Plpc' pays no attention to standard ports.

See Also

ld (1), plpcl (1), bind (3)

Usage

```
plpcl <program name> [<'ld' options>] [/ <'plpc' options>]
```

Description

'Plpcl' is a shell file that invokes the Primos PL/P compiler and the Primos segmented loader. If 'plpcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".plp", although in <program name> it may be specified with or without the ending ".plp". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'plpc' will be called with the <'plpc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
plpcl myprog.plp
plpcl myprog subs.b subs2.b -l mylib
plpcl myprog / -xv -l mylist
```

Messages

"<program name>.plp: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

plpc (1), ld (1), bind (3)

Usage

```
pmac {-<option>[<level>]} <input file>
          [-b [<binary file>]]
          [-l [<listing file>]]
          [-z <PMA option>]
   <option> ::= v | x
```

Description

'Pmac' serves as the Subsystem interface to the Primos macro assembler (PMA). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos PMA command and causes it to be executed.

Options

The general structure of an 'pmac' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing containing the machine code representation of each instruction.

Level 2 generates a full source code listing that includes the code generated by all macro calls.

-x(1) [1..2] - Cross-reference listing control.

Level 1 causes the compiler to generate a cross-reference listing containing all variables referenced in executable statements and omitting those that are declared but never referenced.

Level 2 causes the compiler to generate a full cross-reference of all variables.

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.s", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.s", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".s" or ".pma") or the device "/dev/tty", in which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.s" is

```
pmac -v1x1 file.s -b file.b -l /dev/null
```

which corresponds to the PMA command

```
pma -i *>file.s -b *>file.b -l no
```

Examples

```
pmac file.s
pmac -x dmach.s -b b_dmach -l l_dmach
pmac -v2 macroprog.s -z"-newopt"
```

Messages

"Usage: pmac ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.

"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'pmac' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Pmac' pays no attention to standard ports.

See Also

ld (1), pmacl (1), bind (3)

Usage

pmacl <program name> [<'ld' options>] [/ <'pmac' options>]

Description

'Pmacl' is a shell file that invokes the Primos Macro Assembler and the Primos segmented loader. If 'pmacl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".s", although in <program name> it may be specified with or without the ending ".s". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'pmac' will be called with the <'pmac' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
pmacl myprog.s
pmacl myprog subs.b subs2.b -l mylib
pmacl myprog / -x -l mylist
```

Messages

"<program name>.s: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

pmac (1), ld (1), bind (3)

Usage

```
pr {<file_spec> | -h <heading>} [/ <sp_opts>]

<file_spec> ::= <filename> | -[<stdin_number>] |
                -n(<stdin_number> | <filename>)
```

Description

'Pr' is used to print paginated listings of text files on the line printer.

Files to be printed are specified by <file_spec>s; see 'cat' for further information on the semantics of this construct.

Spooler control options may appear on the command line, but must be separated from file names by an argument consisting only of a slash. See 'sp' and the library routine 'open' for further information on <sp_opts>.

Examples

```
lf -c | find .r | sort | pr -n
pr file1 file2 file3
cat part1 part2 | pr -
pr form / p/narrow/
```

Files

```
//spoolq/prt???  for spool file
//spoolq/q.ctrl for queue control file
```

Messages

See 'print'

See Also

print (1), sp (1), cat (1)

Usage

primos

Description

'Primos' allows users to use the Primos command interpreter without terminating the Subsystem. The 'primos' command pushes a new listener level of the Primos command interpreter with a call to the Primos routine COMLV$. The Primos command interpreter then prompts with "OK," (or whatever prompt has been set). The user can then execute any Primos commands (that do not disturb segments '4040 and '4041) in the normal fashion. Executing the Primos command REN (re-enter), or executing the command "swt" will cause the Subsystem to close any Primos file units that were left open by Primos commands and continue where it left off.

Examples

primos

See Also

stop (1), x (1), Primos comlv$

Usage

```
print {<option>} {-h <heading>|<file_spec>}
<option> ::= -i <indent> | -j | -l <length> |
             -m <margin> | -p

<file_spec> ::= <filename> | -[<stdin_number>] |
                -n[<stdin_number> | <filename>]
```

Description

'Print' is an enhanced version of Kernighan and Plauger's 'print' program from *Software Tools*. It produces paginated listings with page headings on its standard output and is well suited for printing text files on a hard-copy terminal or a line printer.

Options are available to control the format of the listing as follows:

-i A "-i" followed by an integer causes 'print' to prepend the specified number of blanks to each output line, indenting the listing from the left margin.

-j The "-j" option causes 'print' to put out a form-feed character (FF) at the end of each page. Normally, 'print' puts out blank lines to get to the top of the next page.

-l A "-l" followed by an integer causes 'print' to change its idea of how many lines there are on a page to the specified number. By default, 66 lines per page are assumed.

-m A "-m" followed by an integer may be used to set the number of blank lines that are left at the top and bottom of each page. The default setting is 6 lines (one inch). The heading produced at the top of each page is centered in this group of lines.

-p Selecting the "-p" option is equivalent to selecting "-j" and "-i 5". This option is designed for use when the output is directed to a line printer.

If no <file_spec> arguments are specified, 'print' prints standard input. Otherwise, 'print' prints the files selected by each <file_spec>. For further information on the options available in the <file_spec> construct, see the Reference Manual entry for 'cat'.

'Print' produces a header for each page of output, consisting of the name of the file being printed, the time and date of printing, and the current page number in the file. The "file name" field of the header may be changed to an

arbitrary string by using the "-h" option followed by the desired header text. A "-h" affects all <file_spec>s to its right, up until the next "-h". If a "-h" followed by an empty string ("") is specified, 'print' reverts to using the name of the file in the header.

Examples

```
file> print
print file >neat
files .r$ | print -p -n >/dev/lps
eight_lines_per_inch> print -l 88 -i10
```

Messages

"<file-name>: can't print" if file could not be read
"Usage: print ..." for improper argument syntax

See Also

pr (1), sp (1), cat (1)

Usage

profile [-d <dictionary>] [<profile>]

Description

'Profile' formats the information recorded by a profiled Ratfor program (one compiled with "rp -p") and prepares a report.

Two input files are used. The first contains a dictionary of the subroutines in the traced program and is produced by 'rp' when the program is compiled (with the "-p" option). The name of the dictionary file may be specified explicitly after the "-d" argument; otherwise, "timer_dictionary" is assumed.

The second file contains the actual profile data that are recorded when the traced program is run. Its name may also be specified as an argument; "_profile" is assumed otherwise.

Profile analyzes the two data files and produces a report on standard output, containing the following information:

- Number of times each routine was called
- Real time spent in each routine
- Percentage real time spent in each routine
- CPU time spent in each routine
- Percentage CPU time spent in each routine
- Milliseconds spent in each routine per call
- Paging time spent in each routine
- Percentage paging time spent in each routine

Note that profile can only be used to summarize execution of Ratfor programs compiled with the "-p" option, or Fortran programs in which the necessary trace calls have been included by hand.

Examples

```
profile | sp
profile -d dict1 prof_info
```

Files

"timer_dictionary" for default dictionary.
"_profile" for default profile data.

Messages

"Usage: profile ..." for invalid argument syntax.

Bugs

If the profiled program exits without calling the profile exit routine (e.g. by calling 'error' rather than using 'stop', from Ratfor) no profile data file will be created.

The system clock only has a resolution of 1/330 second, so 'profile' may not be accurate in timing short routines.

Procedure call overhead is charged to the calling routine rather than to the called routine.

See Also

rp (1), st_profile (1), t$entr (6), t$exit (6), t$time (6)

Usage

publish <path_name> { <path_name> }

Description

'Publish' is the recommended means of publishing an article in the Software Tools Subsystem news service. The contents of the files given as arguments (there must be at least one) are entered into the news service archive and sent to all news service subscribers.

Each file named is published as a separate news item. The first non-blank line of each file should be the "headline." The headline may be left-justified or centered. The headline is placed in the index entry for an item, along with the time and date of publishing and the item number. (The item number is used for retrieving specific news items; see the help for the 'news' command.)

'Publish' deletes leading and trailing blank lines and always insures that there is a blank line following the headline. Because of this, output from the text formatter is suitable for publication if it contains no underlining or boldfacing.

WARNING: When news has a large circulation, 'publish' will take a significant amount of time to do its job. DO NOT interrupt it, or you may prevent some users from obtaining a copy in their news box. In the event that 'publish' is interrupted, use "retract -q" to remove the article and then publish it again.

Examples

```
publish new_york_times
publish first second
```

Files

=news=/articles/art<number> for archived articles
=news=/index for article index
=news=/delivery/<login_name> for delivery to subscribers
=news=/subscribers for the subscription list

Messages

"<article>: cannot open" for not being able to access article file.
"<article>: empty file" for trying to publish an empty file.
"Headline too long: <headline>" for trying to use a head-

line that will not fit in the index.

"can't open archive copy file" for not being able to open =news=/article/art<number>.

"cannot make delivery" for not being able to open delivery file.

"can't open index file" for not being able to open index file.

See Also

news (1), subscribe (1), retract (1)

Usage

pword

Description

'Pword' changes a user's login password. A Primos login password consists of up to 16 letters, numbers, and the following special characters: '#', '$', '&', '*', '-', '.', and '/'. Null passwords (consisting of no characters) may or may not be allowed depending on the specific system.

'Pword' turns off terminal echo (to prevent someone from peeking) and requests the old password. It then requests the new password. The new password is requested a second time to verify that the user is changing his password to the correct string. If the two new passwords differ in any way then an error message is printed and the users password is left unchanged. 'Pword' then calls the Primos routine CHG$PW to change the user's password. Any errors are interpreted and printed on the terminal.

Examples

```
pword
Old password: old.password
New password: new.password$
Reenter new password for verification: new.password$
```

Messages

"One of the passwords was illegal" if a password containing an illegal character is entered.

"The old password did not match the actual password" if the old password entered did not match the actual old password of the account.

"Disk is write protected. See system administrator" if the disk on which the passwords reside is write protected.

See Also

Primos CHANGE_PASSWORD command, Primos chg$pw

Usage

quota [-s <quota limit>] [-v] {<file_spec>}

Description

It is possible to set an upper limit to the number of disk records that may be used in a directory. This command may be used to read or set the quota limits on any directory. Use of the 'quota' command without the "-s" argument will result in a display of the form:

a/b (c)

where 'a' is the total number of records currently used in the directory and all of its descendants, 'b' is the current quota, and 'c' is the time-record product; the time record product is a measure of how many records have been in use over time in this directory and may be used in accounting.

Use of the "-s" option will set the quota for the named directory. The argument after the "-s" must decode to a positive-valued long integer. If the value is zero then quota limits are removed from the directory.

Note that *no* error is reported if the user should set the maximum quota to a value less than the number of records currently used. Should this event occur, no files or directories may be created in the directory, nor may any existing files be expanded.

See the help on 'cat' for a full description of the meaning of <file_spec>.

Examples

```
quota /u(a b c)/spaf -s 0
quota foobar/junk
```

Messages

"Usage: quota ..." for improper arguments.
"<pathname>: can't get quota information" for various file system errors or lack of access rights.
"<pathname>: not a quota directory"; self-explanatory.
"improper quota value" for invalid value of <quota limit>.
"<pathname>: can't set quota" for various file system errors or lack of access rights.

See Also

cat (1), gfdata (2), sfdata (2)

Usage

quote

Description

'Quote' supplies one layer of quotes around strings present on its standard input. It is useful in function calls, to prevent premature evaluation of text by the command interpreter.

For example, suppose the string

"# [a-d]"

were specified as an argument in the invocation of a command file which, in turn, passed the string as an argument to another program or command file. The first command file might access the string using the 'arg' command in a function call:

[arg 1]

However, to prevent the meta-characters "#", "[" and "]" from being interpreted by the shell after the evaluation of the function, the following function call should be used instead:

[arg 1 | quote]

The string will then be quoted before being substituted back into the command line containing the function call, and the meta-characters will not be evaluated.

The result of a function call is quoted automatically by the shell if the variable '_quote_opt' contains the string "YES". This, however, is not the default setting.

Examples

```
to ics002 [args | quote]
echo [arg 1 | quote] >request_file
```

Bugs

Depends on having both ' and " available as quoting characters.

Is probably too smart for general application, but understands the shell's quoting requirements quite well.

See Also

sh (1), arg (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

radix [-i <input radix>] [-o <output radix>] { <number> }

Description

'Radix' is a simple tool that converts numbers from one radix representation to another. The "-i" option specifies the default input radix. (This radix can be overridden with the "<radix>r<number>" notation accepted by 'gctol'). The "-o" option specifies the output radix. If either is omitted, 10 is assumed.

The numbers specified as arguments are converted to the output radix and printed on standard output, one number per line. If no <number> arguments are specified, 'radix' reads numbers from standard input (one per line), converts them, and writes them on standard output (one per line).

If an illegal character is encountered in a number, it and all following characters in the number are ignored.

Examples

```
radix 8r177
radix -i10 -o2 39 12 5
radix -i 16
```

Messages

"Usage: radix ..." for invalid argument syntax.

See Also

gctol (2)

Usage

```
rdatt {<option>}
<option> ::= -t | -l | -n
```

Description

'Rdatt' is part of the toy relational data base system, 'rdb'. It lists the attributes of a relation, specified as standard input, on standard output. The input relation must be a file containing a relation that was created by 'rdmake' or another 'rdb' program; a relation cannot be read from the terminal.

If no options are specified then the type, length, and name of each attribute are listed on one line for each attribute. If any of the options 't', 'l', or 'n' (type, length, name) are specified, then only the characteristics corresponding to the requested option will be listed.

Examples

```
p1.rel> rdatt
p2.rel> rdatt -tn >attrlist
```

Messages

"Sorry, a relation can't be read from the terminal"
"Can't access input relation"
"relation is corrupted!!"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

```
rdavg [<selection expr>] <attr>
```

Description

'Rdavg' is part of the toy relational data base management system, 'rdb'. It computes the average value of a specified attribute over all rows of the relation that satisfy the optional select expression. If no select expression is given then it computes the average of an attribute over all rows of the relation. Standard input 1 must be directed to a file containing an 'rdb' relation. The result is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The select expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal.

Examples

```
p.rel> rdavg weight
p.rel> rdavg "height>65&height<80" weight
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Cannot load input relation"
"Usage: rdavg [<selection expr>] <attr>"
"Domain not found"
"Strings can't be averaged"
"Average is undefined for empty relation"
"Invalid expression"
"expected domain name or literal"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdcount (1), rddiff (1), rddiv (1), rdint (1), rdmax (1), rdmin (1), rdnat (1), rdsum (1)

Usage

rdcat

Description

'Rdcat' is part of the toy relational data base system, 'rdb'. It creates a new relation by concatenating the two relations specified as standard inputs 1 and 2 and writes the new relation on standard output 1. Both relations must have identical descriptions -- the domains must be identical and in the same order.

The input relations must be files containing relations that were created by 'rdmake'; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

Identical tuples are not removed from the resulting relations. These can be removed using 'rdsort' and 'rduniq'.

Examples

```
p1.rel> p2.rel> rdcat >p.rel
p.des> newp.data> rdmake | p.rel> rdcat >newp.rel
```

Messages

"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"
"Can't access input relation 1"
"Can't access input relation 2"
"Relations must have identical descriptions"

Bugs

It would be nice if the relations only had to have the same structure to be concatenated.

If standard output is directed to "/dev/lps", the relation is written in binary.

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

rdcount [<selection expr>]

Description

'Rdcount' is part of the toy relational data base management system, 'rdb'. It lists the number of rows in a relation satisfying the optional select expression. If no select expression is given then it lists the total number of rows in the relation. Standard input 1 must be directed to a file containing an 'rdb' relation. The result is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The select expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal.

Examples

```
p.rel> rdcount
p.rel> rdcount "color='red'"
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Cannot load input relation"
"Invalid expression"
"expected domain name or literal"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rddiff (1), rddiv (1), rdint (1), rdmax (1), rdmin (1), rdnat (1), rdsum (1)

Usage

rddiff

Description

'Rddiff' is part of the toy relational data base system, 'rdb'. It creates a new relation by performing the set difference of the two relations specified as standard inputs 1 and 2 and writes the new relation on standard output 1. Both relations must have identical descriptions -- the domains must be identical and in the same order.

The new relation is formed by examining both input relations and retaining those rows that are in the first relation (standard input 1) but not in the second relation (standard input 2). The remaining rows of both relations are discarded.

For example:

p1.rel

code	name
100	pens
101	ink

p2.rel

code	name
100	pens
105	ruler

```
p1.rel> p2.rel> rddiff >p.rel
```

p.rel

code	name
101	ink

The input relations must be files containing relations that were created by 'rdmake' or other 'rdb' programs; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal; otherwise, the output relation is written in binary internal format for processing by other 'rdb' programs.

Examples

```
p1.rel> p2.rel> rddiff >p.rel
p.des> newp.data> rdmake | p.rel> rddiff >newp.rel
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Can't access input relation 1"
"Can't access input relation 2"
"Relations must have identical descriptions"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiv (1), rdint (1), rdmax (1), rdmin (1), rdnat (1), rdsum (1)

Usage

rddiv

Description

'Rddiv' is part of the toy relational data base system, 'rdb'. It creates a new relation by performing the set division of the two relations specified as standard inputs 1 and 2 and writes the resulting new relation on standard output.

Standard input 1 is the dividend, standard input 2 is the divisor, and standard output (the resulting relation) is the quotient. The quotient consists of those rows of the dividend, projected onto the attributes not in the division, whose corresponding attributes include every row of the divisor. In other words, a row X will appear in the quotient if and only if the pair <X,Y> appears in the dividend for all rows Y appearing in the divisor.

For example:

p1.rel

code	name
100	pens
100	ink
100	ruler
101	pens
101	paper
102	ink
103	ink
103	pens

p2.rel

name
pens
ink

```
p1.rel> p2.rel> rddiv >p.rel
```

p.rel

code
100
103

The input relations must be files containing relations that were created by 'rdmake' or other 'rdb' programs; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to the terminal; otherwise, the output relation is written in binary internal format for processing by other 'rdb' programs.

Identical tuples are not removed from the resulting relation. These can be removed using 'rdsort' and 'rduniq'.

Examples

```
p1.rel> p2.rel> rddiv >p.rel
p1.rel> p2.rel> rddiv | rdsort | rduniq | rdprint
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Cannot load input relation 1"
"Cannot load input relation 2"
"Relation 2 has domain not defined in relation 1"
"Couldn't rewind sort file 1"
"Couldn't rewind sort file 2"
"Error on sort file 2"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rdint (1), rdmax (1), rdmin (1), rdnat (1), rdsum (1)

Usage

rdextr

Description

'Rdextr' is part of toy relational data base management system 'rdb'. It converts a relation to a standard text file using a format file. Standard input 1 must be directed to a file containing an 'rdb' relation and standard input 2 must be directed to a file containing a description of the desired output format (see below). The relation data is output on standard output as a text file.

The output format file is very similar in structure to the input format file used by 'rdmake'. The only difference is that the data type of the relation domain is not included in the output format. Each line of the output format file controls the the conversion of one domain of the relation. The domains are output in the order listed in the format file; domains may be omitted or duplicated by omitting or duplicating lines in the output format file.

Each line of the file has the following format:

<domain name> [d[<delimiter>]] [l<length>]

<Domain name> must be the name of a domain in the relation; <delimiter> is a single character delimiter; <length> is non-negative integer. When a field is output, it is converted to character form and blank padded so that it takes no less than <length> characters (if "l<length>" is not specified, <length> is assumed to be zero). The characters are placed in the output followed by the delimiter character (if "d<delimiter>" is not specified, <delimiter> is assumed to be a blank; if "d" with no delimiter is specified, no delimiter is output). For the last domain in the format file, <delimiter> is always assumed to be a NEWLINE character. For example,

```
pno d,
pname d l20
city d$ l10
```

In the first line, "pno" is output with no blank padding, followed by a comma. In the second line, "pname" is output with blank padding to 20 characters with no delimiter (assuming "pname" was described as "s20" in the relation, 20 characters would always be output for "pname"). In the last line, "city" is output with blank padding to 10 characters and then followed by a newline character.

Examples

```
y.rel> y.fmt> rdextr >y.data
p.rel> p.fmt> rdextr | field ...
... rdsort | rduniq | x.fmt> rdextr >x.data
```

Messages

"Cannot access input relation"
"<domain>: domain not found"
"Illegal output length"
"Unrecognized word"
"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"

See Also

dtoc (2), ltoc (2), rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

rdint

Description

'Rdint' is part of the toy relational data base system, 'rdb'. It creates a new relation by performing the intersection of the two relations specified as standard inputs 1 and 2 and writes the new relation on standard output 1. Both relations must have identical descriptions -- the domains must be identical and in the same order.

The intersection creates a new relation containing all the rows which appear in both sets -- all other rows are discarded. Identical rows are not removed from the resulting relation. These can be removed by using 'rdsort' and 'rduniq'.

For example:

p1.rel

code	name
100	pens
101	ink

p2.rel

code	name
100	pens
105	ruler

```
p1.rel> p2.rel> rdint >p.rel
```

p.rel

code	name
100	pens

The input relations must be files containing relations that were created by 'rdmake' or other 'rdb' programs; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal; otherwise, the output relation is written in binary internal format for processing by other 'rdb' programs.

Examples

```
p1.rel> p2.rel> rdint >p.rel
p1.rel> p2.rel> rdint | rdsort | rduniq | rdprint
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Can't access input relation 1"
"Can't access input relation 2"
"Relations must have identical descriptions"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rddiv (1), rdmax (1), rdmin (1), rdnat (1), rdsum (1)

Usage

```
rdjoin <sel expr> { <new domain> }
   <sel expr> ::= <term> { '|' <term> }
       <term> ::= <factor> { & <factor> }
     <factor> ::= ~ <factor> | <primary>
    <primary> ::= <object> <rel op> <object>
     <object> ::= <domain> | <domain>.1 | <domain>.2
                | <integer> | <real> | <string>
     <rel op> ::= < | > | = | <= | >= | ~= | == | <>
 <new domain> ::= <old domain> [ = <domain> ]
 <old domain> ::= <domain> | <domain>.1 | <domain>.2
```

Description

'Rdjoin' is part of the toy relational data base management system 'rdb'. It joins two relations, selects relevant tuples, and projects the new relation over specified domains. Standard input 1 and standard input 2 must be directed to files containing 'rdb' relations. The result relation is written to standard output. Identical tuples are not removed from the resulting relation. These can be removed using 'rdsort' and 'rduniq'.

The input relations must be files containing relations that were created by 'rdmake' or other 'rdb' programs; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

The new relation is formed (effectively) by concatenating every tuple of relation 1 to every tuple of relation 2. The selection expression is then evaluated for every new tuple; tuples for which the selection expression is false are discarded. Then a new relation is formed from the selected tuples by projecting over the domains specified on the command line.

The selection expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal. The usual operator hierarchy applies: relational conditions first, followed by "~", "&" and then "|". Literals must be the same type as the domain to which they are compared: string literals must be quoted (either single or double quotes) and integer and real literals must follow the syntax allowed by 'gctol' and 'ctod'.

Since domains may have the same names in the input relations, domain names may be qualified by ".1" or ".2 suffixes corresponding to domain in the first or second relation, respectively. If a domain name appears in only one input relation, it need not be qualified; if it appears

in both relations, it must be qualified.

If no list of domains is specified for projecting the output relation, the output relation is projected over all of the domains of both input relations. Duplicate domain names are not allowed in the output relation. If there are duplicate domain names in output relation, the domains must be renamed using the "<old domain>=<domain>" form. Unique domain names may also be changed using this notation.

Examples

```
p.rel> sp.rel> rdjoin _
    "pno.1=pno.2" pno.1=no pname=name qty
p.rel> p.rel> rdjoin _
    "city.1=city.2" pno.1=pno1 pno.2=pno2
p.rel> p.rel> rdjoin _
    "pname.1=pname.2&color.1~=color.2" pno.1=pno1 pno.2=pno2
```

Messages

```
"Usage:  rdjoin <selection expr> { <domain> }"
"Cannot load input relation 1"
"Cannot load input relation 2"
"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"
"Resulting relation has too many domains"
"Too many fields in new relation"
"<domain>:  invalid name"
"<domain>:  domain not found"
"<domain>:  duplicate output domain"
"<domain>:  cannot add new domain"
"<domain>:  duplicate output domain"
"<domain>:  domain not found or ambiguous"
"Invalid expression"
"Unbalanced parentheses"
"Missing relational operator"
"Comparing two literals is bogus!"
"Types to be compared are not compatible"
"Expected domain name or literal"
"Too many literals"
"Invalid integer constant"
"Invalid real constant"
"Missing quote"
"Illegal character"
"Selection expression too complicated"
```

Bugs

Uses a slow and stupid algorithm.

If domain names are duplicated in the input relations, domains must be renamed on output; hence all desired output

domains must be listed.

If standard output is directed to "/dev/lps", the relation is written in binary.

See Also

ctod (2), gctol (2), rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

rdmake

Description

'Rdmake' is part of the toy relational data base management system 'rdb'. It creates an 'rdb' relation from a data file and a description file. The relation data is read from standard input 1 and the description file is read from standard input 2 (see below). The new relation is written to standard output. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

Identical tuples are not removed from the resulting relations. These can be removed using 'rdsort' and 'rduniq'.

The description file is very similar in structure to the output format file used by 'rdextr'. Each line of the description file causes the creation of a new domain in the relation and describes how the data for that domain is to be obtained. Each line has one of the following formats:

```
i       <domain name> [ d[<delim>] ] [ l<flen> ]
r       <domain name> [ d[<delim>] ] [ l<flen> ]
s<slen> <domain name> [ d[<delim>] ] [ l<flen> ]
```

The first two entries in each line describe the format of the relation. The first format describes an integer domain (containing 32 bit integers), the second describes a real domain (containing 64 bit reals), and the last describes a string domain containing <slen> character strings (<slen> must be a positive integer). The <domain name> must begin with a letter and contain only letters, digits, and underscores. Case is significant in identifiers.

The last two (optional) entries in each format describe how each domain is to be obtained from the data file. Data for each tuple is taken from a single line in the data file. Fields are extracted in the order of lines in the description file. Each field is extracted by first skipping over any leading delimiter characters specified by <delim> in the "d<delim>" entry (if the entry is omitted, <delim> is assumed to be a blank; if "d" is specified without a delimiter, no delimiter is allowed). Then characters are collected up to the next occurrence of <delim>. In any case, no more than <flen> characters are collected (if "l<flen>" is omitted, <flen> is assumed to be a very large number). The extracted field is then converted to the proper internal format and placed in the tuple. Integers and reals are converted into binary representations with

'gctol' and 'ctod'; strings are blank padded to full length. For example,

```
s6 pno
s15 pname l15 d
i qty d,
r price d, l10
```

"pno" is a string containing 6 characters; it is obtained by skipping blanks and then collecting characters up to the next blank. "Pname" is a string containing 15 characters; it is obtained by taking exactly 15 characters from the input line (if a NEWLINE is encountered, spaces are supplied). "Qty" is an integer domain that is extracted by skipping leading commas, collecting characters up to the next comma, and then converting the resulting string into an integer with 'gctol'. "Price" is a real domain; it is extracted by skipping leading commas, collecting characters up to the next comma (but not more than 10), and then converting the resulting string into a real using 'ctod'.

Examples

```
p.data> p.des> rdmake | rdsort | rdjoin >p.rel
sp.des>2 rdmake >sp.rel
```

Messages

"Illegal length"
"Illegal data type"
"Illegal domain name"
"Duplicate name"
"Can't add domain"
"Illegal input length"
"Unrecognized word"
"<integer>: bad integer"
"<real>: bad real"

Bugs

If standard output is directed to "/dev/lps", the relation is written in binary.

An empty field cannot be specified by two occurrences of a delimiter.

See Also

ctod (2), gctol (2), rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

```
rdmax [<selection expr>] <attr>
```

Description

'Rdmax' is part of the toy relational data base management system, 'rdb'. It finds the maximum value of a specified attribute over all rows of the relation that satisfy the optional select expression. If no select expression is given then it finds the maximum value of an attribute over all rows of the relation. Standard input 1 must be directed to a file containing an 'rdb' relation. The result is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The select expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal.

Examples

```
p.rel> rdmax size
p.rel> rdmax "color='red'" cost
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Cannot load input relation"
"Usage: rdmax [<selection expr>] <attr>"
"Domain not found"
"No rows satisfy selection expression"
"Invalid expression"
"expected domain name or literal"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rddiv (1), rdint (1), rdmin (1), rdnat (1), rdsum (1)

Usage

rdmin [<selection expr>] <attr>

Description

'Rdmin' is part of the toy relational data base management system, 'rdb'. It finds the minimum value of a specified attribute over all rows of the relation that satisfy the optional select expression. If no select expression is given then it finds the minimum value of an attribute over all rows of the relation. Standard input 1 must be directed to a file containing an 'rdb' relation. The result is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The select expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal.

Examples

```
p.rel> rdmin size
p.rel> rdmin "color='red'" cost
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"Cannot load input relation"
"Usage: rdmin [<selection expr>] <attr>"
"Domain not found"
"No rows satisfy selection expression"
"Invalid expression"
"expected domain name or literal"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rddiv (1), rdint (1), rdmax (1), rdnat (1), rdsum (1)

Usage

rdnat

Description

'Rdnat' is part of the toy relational data base system, 'rdb'. It creates a new relation by performing the natural join of the two relations specified as standard inputs 1 and 2 and writes the resulting new relation on standard output 1.

The new relation is formed (effectively) by "pasting together" tuples of relation 1 and relation 2 having the same values on the same attributes. Identical tuples are not removed from the resulting relation. These can be removed by using 'rdsort' and 'rduniq'.

For example:

p1.rel

code	name
100	pens
101	ink

p2.rel

code	loc
100	rear
105	front

```
p1.rel> p2.rel> rdnat >p.rel
```

p.rel

code	name	loc
100	pens	rear

The input relations must be files containing relations that were created by 'rdmake' or other 'rdb' programs; relations cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal; otherwise, the output relation is written in binary internal format for processing by other 'rdb' programs.

Examples

```
p1.rel> p2.rel> rdnat >p.rel
p1.rel> p2.rel> rdnat | rdsort | rduniq | rdprint
```

Messages

"Sorry, a relation can't be read from the terminal"

```
     "relation is corrupted!!"
     "Cannot load input relation 1"
     "Cannot load input relation 2"
     "Resulting relation has too many domains"
     "in add_field_to_rd; bogus type passed"
     "field not found"
```

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rdint (1), rddiv (1), rdmax (1), rdmin (1), rdsum (1)

Usage

```
rdprint { -d | -r }
```

Description

'Rdprint' is part of the toy relational data base management system 'rdb'. It displays a relation in readable form. Standard input 1 must be directed to a file containing an 'rdb' relation. The input relation must be a file containing relation that was created by 'rdmake' or other 'rdb' programs; a relation cannot be read from the terminal.

Printable output is produced on standard output. The "-d" option indicates that only the relation description is to be displayed; the "-r" option indicates that only the relation data is to be displayed. If both or neither of these options are present, both the description and data are displayed.

Examples

```
p.rel> rdprint -d
p.rel> rdproj pname pno | rdsort | rduniq | rdprint -r
```

Messages

"Usage: rdprint (-d | -r)"
"Sorry, a relation can't be read from the terminal"
"Can't access input relation"
"relation is corrupted!!"

Bugs

Relations more than 80 columns wide display badly on the terminal.

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

```
rdproj { <new domain> }
   <new domain> ::= <domain> [ = <domain> ]
```

Description

'Rdproj' is part of the toy relational data base management system 'rdb'. It projects a relation over specified domains. Standard input 1 must be directed to a file containing an 'rdb' relation. A new relation is created and written to standard output. The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; a relation cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

Domains are projected in the order specified on the command line. A domain can be renamed by using the syntax "<old>=<new>". Identical tuples are not removed from the resulting relations. These can be removed using 'rdsort' and 'rduniq'.

Examples

```
p.rel> rdproj pname=name color city | rdsort | rduniq
sp.rel> rdproj pno=no | rdsort | rduniq
```

Messages

"Can't access input relation"
"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"
"Too many fields in new relation"
"<domain>: invalid name"
"<domain>: field not found"
"<domain>: duplicate field"
"<domain>: cannot add new field"

Bugs

If standard output is directed to "/dev/lps", the relation is written in binary.

If a single domain is to be renamed, all other domains must be named in the argument list.

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

```
rdsel <sel exp>
   <sel expr> ::= <term> { '|' <term> }
       <term> ::= <factor> { & <factor> }
     <factor> ::= ~ <factor> | <primary>
    <primary> ::= <object> <rel op> <object>
     <object> ::= <domain>
                | <integer> | <real> | <string>
     <rel op> ::= < | > | = | <= | >= | ~= | == | <>
```

Description

'Rdsel' is part of the toy relational data base management system 'rdb'. It selects tuples from a relation based on a selection expression given as an argument. Standard input 1 must be directed to a file containing an 'rdb' relation. The result relation is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

The new relation is formed (effectively) by evaluating the selection expression for each tuple in the input relation. Tuples for which the selection expression is false are then discarded and the remaining tuples are placed in the output relation.

The selection expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal. The usual operator hierarchy applies: relational conditions first, followed by "~", "&" and then "|". Literals must be the same type as the domain to which they are compared: string literals must be quoted (either single or double quotes) and integer and real literals must follow the syntax allowed by 'gctol' and 'ctod'.

Examples

```
p.rel> rdsel "pno>'p1'&(color='Red'|weight<15" | rdprint -r
sp.rel> rdsel "sno<='s3'"
p.rel rdsel "weight>5&weight<20"
```

Messages

"Usage: rdsel <selection expr>"
"Cannot load input relation"

"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"
"Invalid expression"
"Unbalanced parentheses"
"Missing relational operator"
"Comparing two literals is bogus!"
"Comparing two literals is bogus!"
"Types to be compared are not compatible"
"Expected domain name or literal"
"<domain>: domain not found or ambiguous"
"Too many literals"
"Invalid integer constant"
"Invalid real constant"
"Missing quote"
"Illegal character"
"Selection expression too complicated"

Bugs

If standard output is directed to "/dev/lps", the relation is written in binary.

See Also

ctod (2), gctol (2), rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

rdsort { <domain> }

Description

'Rdsort' is part of the toy relational data base system, 'rdb'. It sorts the tuples in a relation on the domains specified in the argument list. Standard input 1 must be directed to a file containing an 'rdb' relation; the sorted relation is written on standard output. The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' program; a relation cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

The relation is sorted on the domains specified in the argument list. Integer and real domains are sorted in numeric order; string domains are sorted in the ASCII collating sequence. If no arguments are specified, the relation is sorted on all domains in the order they appear in the relation.

Examples

```
p.rel> rdsort color >np.rel
sp.rel> rdproj sno | rdsort | rduniq | rdprint
```

Messages

"Can't access input relation"
"Sorry, a relation can't be read from the terminal"
"Relation is corrupted!!"
"Too many sort keys"
"<domain>: field not defined"

Bugs

If standard output is directed to "/dev/lps", the relation is written in binary.

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

rdsum [<selection expr>] <attr>

Description

'Rdsum' is part of the toy relational data base management system, 'rdb'. It computes the sum of the values of a specified attribute over all rows of the relation that satisfy the optional select expression. If no select expression is given then it computes the sum of the values of an attribute over all rows of the relation. Standard input 1 must be directed to a file containing an 'rdb' relation. The result is written to standard output.

The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' programs; the relation cannot be read from the terminal. The select expression is formed from the logical operators "&" (and), "|" (or), and "~" (not) connecting relational conditions involving two domains or a domain and a literal. The sum cannot be computed for domains containing strings.

Examples

```
p.rel> rdsum length
p.rel> rdsum "length>20" total_quantity
```

Messages

"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"
"cannot load input relation"
"Usage; rdsum [<selection expr>] <attr>"
"Domain not found"
"Strings can't be averaged"
"Invalid expression"
"expected domain name or literal"

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1), rdatt (1), rdavg (1), rdcount (1), rddiff (1), rddiv (1), rdint (1), rdmax (1), rdmin (1), rdnat (1)

Usage

rduniq

Description

'Rduniq' is part of the toy relational data base system, 'rdb'. It removes duplicates from a sorted relation. Standard input 1 must be directed to a file containing an 'rdb' relation; the new relation is written on standard output. The input relation must be a file containing a relation that was created by 'rdmake' or other 'rdb' program; a relation cannot be read from the terminal. The output relation is displayed in a readable format if standard output is directed to a terminal (display in binary would be quite a mess); otherwise, the output relation is written in binary, internal format for processing by other 'rdb' programs.

Examples

```
sp.rel> rdproj sno | rdsort | rduniq | rdprint
```

Messages

"Can't access input relation"
"Sorry, a relation can't be read from the terminal"
"relation is corrupted!!"

Bugs

If standard output is directed to "/dev/lps", the relation is written in binary.

See Also

rdcat (1), rdextr (1), rdjoin (1), rdmake (1), rdprint (1), rdproj (1), rdsel (1), rdsort (1), rduniq (1)

Usage

```
repeat
   { <command> }
until [ <value> ]
```

Description

'Repeat' implements a Pascal-like repeat loop in the Shell. The optional <value> after the 'until' command may be any string or function call; if it is zero, empty, or missing altogether, it is interpreted as false, otherwise it is interpreted as true. If <value> is false, control transfers back to the top of the loop and the list of commands are executed again, otherwise the loop terminates and any other commands after the loop are executed.

'Repeat' operates by saving a copy of any commands entered between the 'repeat' statement and the 'until' statement in a temporary file. The top of the file contains a (hopefully) unique label and when the 'until' statement is entered, an 'if' statement is generated using <value> as the condition for a 'goto' to the label. For example the repeat loop

```
repeat
   set i = [eval i + 1]
until [eval i ">" 7]
```

generates the following Shell file

```
:L01t
set i = [eval i + 1]
if [eval i ">" 7]
else
goto L01t
```

and then calls the shell to execute it. Since it is executing as another level of the shell, the 'exit' command will actually cause early termination of the loop, but a 'goto' statement to a label outside the scope of the loop will not work because the label is not accessible from within the shell file. Another incidental advantage obtained from preprocessing the structure and executing as another Shell level is that this loop can be issued from the terminal and it will behave reasonably, i.e. - it will execute the loop instead of ignoring any further commands the way a 'goto' statement does.

Examples

```
declare i = 0
repeat | change ?* "-- & --"     # they pipe, also
   echo [i]
```

```
      set i = [eval i + 1]
   until [eval i ">" 7]

   repeat
      long_command
      even_longer_command
      if [flag]
         exit              # terminate the loop early
      fi

      very_short_command
   until [done]

   repeat
      hd swt
      pause for 5
   until                   # infinite loop (defaults to false)
```

Messages

"Can't create temporary file for repeat loop" if there is a problem creating a file to hold the processed 'repeat' loop.

"Too many arguments" if there is an argument overflow while trying to copy the current arguments for the 'repeat' statement.

"Missing 'until'" if end-of-file is reached on command input before a matching 'until' was found.

Bugs

Since the 'repeat' command causes another level of the shell to be executed, the arguments need to be copied to the next level. If there are many arguments to other commands in the network in which the 'repeat' is contained, then there could be an argument overflow.

Typing 'repeat' on someone's terminal will cause the Shell to ignore any command they type until an EOF or a matching 'until' is typed.

See Also

if (1), then (1), else (1), fi (1), case (1), goto (1), until (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

retract [-q] { <article number> }

Description

'Retract' is the recommended means of retracting an article from the Software Tools Subsystem news service. The articles mentioned by number as arguments are removed from the news index, news archive, and news delivery files. If the article has been read by a subscriber, a notice of retraction is placed in his newsbox; otherwise, no notice of the retraction is published.

Under normal circumstances, one never need retract a news story. 'Retract' exists to remedy the all-too-frequent circumstance of an erroneous news article. By retracting an incorrect article and re-publishing a correct version, the news archive is less cluttered and those users who have not read their news never know of the retraction.

When called with the "-q" option, 'retract' does not tell subscribers who have read an article that it has been retracted. This "quiet" option is often useful for removing all traces of an outdated article without bothering users who have read it.

WARNING: When news has a large circulation, 'retract' will take a significant amount of time to do its job. DO NOT interrupt it, or you may leave an article in a half-retracted state. In the event 'retract' is interrupted, just retract the article again -- the only problem (in almost all cases) will be that some users are given two retraction notices.

Examples

retract -q 12
retract 299 233

Files

=news=/articles/art<number> for archived articles
=news=/index for article index
=news=/delivery/<login_name> for delivery to subscribers
=news=/subscribers for the subscription list

Messages

"<article>: not an article number" for a non-numeric article number.
"<article>: can't retract" for an unwritable delivery file.

"<article>: not found" for trying to retract a non-existant article.
"<article>: not your article" for trying to retract someone else's article.
"can't open index file" for not being able to open index file.
"can't open subscribers list" for not being able to open subscriber file.
"Usage: retract ..." for incorrect arguments.

See Also

news (1), subscribe (1), publish (1)

Usage

rfc <file.r> [[<rp_args>] [/ [<fc_args>]]]

Description

'Rfc' is a shell program that causes the specified Ratfor program to be preprocessed and compiled, but not loaded. It is useful for rebuilding single modules in a multi-module program. The source file is expected to be named <program>.r and the output object code is named <program>.b. A check is made to verify the existence of the source program; if it is not present, processing is discontinued.

The ".r" suffix on the source file name is not required, although 'rfc' requires that the source code reside in a file named with a ".r" suffix; thus one may write "rfc file" to compile the contents of "file.r".

Special options for 'rp' may be placed after the file name. Options for 'fc' may be placed after the 'rp' options, as long as the two groups are separated by a slash. Example: "rfc prog -a / -c".

Examples

```
rfc profile.r
rfc profile

rfc stuff.r / -do0q
```

Messages

"<source_file>: can't open" for missing ".r" file
"Usage: rfc ..." for no arguments

See Also

rp (1), fc (1), fcl (1), ld (1), rfl (1)

Usage

rfl [<file.r> [<ld_args>] [/ [<rp_args>] [/ [<fc_args>]]]]

Description

'Rfl' is a shell program that causes the specified Ratfor program to be preprocessed, compiled and loaded. The source file is expected to be named <program>.r and the output object code is named <program>. A check is made to verify the existence of the source program; if it is not present, processing is discontinued.

A few examples may clarify the (somewhat obscure) command syntax.

'Rfl' shares the shell variable 'f' with the shell program 'e'; thus one may compile the last program edited simply by typing "rfl" with no arguments. If the source file is to be named explicitly, it follows the "rfl" (e.g. "rfl file.r"). The ".r" suffix on the source file name is not required, although 'rfl' requires that the source code reside in a file named with a ".r" suffix; thus one may write "rfl file" to compile the contents of "file.r".

Options for 'ld' (names of libraries, for example) may follow the name of the source file, e.g. "rfl prog -l vthlib". Special options for 'rp' may be placed after the 'ld' options, as long as they are separated by an argument consisting only of a slash; for example, "rfl prog -l vthlib / -c". Finally, options for 'fc' may be placed after the 'rp' options, as long as the two groups are separated by a slash. Example: "rfl prog -l vthlib / -c / -t".

Examples

```
rfl      # ratfor, fortran, and load the last file edited

rfl profile
rfl profile.r

rfl sol -l vthlib

rfl rsa -l vswtml / -t / -t -l rsa.list
```

Messages

"<source_file>: can't open" for missing ".r" file
"no source file" for missing file name and no 'f' variable

See Also

rp (1), fc (1), fcl (1), ld (1)

Usage

rnd { -l <lower> | -u <upper> | -n <number> }

Description

'Rnd' may be used to generate one or more pseudo-random numbers, uniformly distributed over a given range of integers. The arguments specify the range and number of pseudo-random integers to be generated.

The "-l" and "-u" options set the lower and upper bounds, respectively, of the range. The default values are 1 for the lower bound and 100 for the upper bound. The "-n" option controls the number of integers generated; the default is 1.

Examples

```
rnd
rnd -n 10 | stats -tq
rnd -u 10
rnd -l -5 -u 5
```

Bugs

Round-off error may make this program not quite uniform in the long run.

See Also

'rnd' function in Fortran library

Usage

rot [[+ | −] <rotation>]

Description

'Rot' circularly rotates character strings found on standard input the number of positions specified by <rotation>, in a manner similar to the APL function "reversal/rotate". Specification of a positive <rotation> will rotate the string from left to right the number of characters specified. If <rotation> is negative, the string will be rotated from right−to−left. When a string is encountered that is not as long as the absolute value of <rotation>, the string is rotated circularly until the rotation count is exhausted.

If <rotation> is not specified, the strings on standard input are reversed, as with the APL monadic function.

Examples

```
palindromes> rot
ar −t archive | rot −40 | sort | rot 40
rot 5
```

See Also

take (1), drop (1), iota (1), stake (2), sdrop (2)

Usage

```
rp [-{a | b | c | d | f | g | h | l | m | p | s | t | v | y}]
   [-o <output_file>] {<input_file>} [-x <translation file>]
```

Description

'Rp' is the Georgia Tech extended Ratfor preprocessor. It replaces the original Kernighan/Plauger Ratfor preprocessor, locally supported under the name 'rf' at Version 7.

A full description of the Ratfor language is quite beyond the scope of this document. For complete information, please see the *User's Guide for the Ratfor Preprocessor*

The following options are available:

-a Abort all active shell programs if any errors were encountered during preprocessing. This option is useful in shell programs like 'rfl' that wish to inhibit compilation and loading if preprocessing failed. By default, this option is not selected; that is, errors in preprocessing do not terminate active shell programs.

-b Do not map long identifiers or identifiers containing upper case letters into unique six character Fortran identifiers. This option is useful if your Fortran compiler will accept names longer than six characters.

-c Include statement-count profiling code in the generated Fortran. When this option is selected, calls to the library routines 'c$init', 'c$incr', and 'c$end' will be placed (unobtrusively) in the output code. When the preprocessed program is run, it will generate a file named "_st_count" containing execution frequencies for each line of source code. The utility program 'st_profile' may then be used to combine source code and statement counts to form a readable report.

-d Inhibit generation of the long-name dictionary. Normally, a dictionary listing all long names used in the Ratfor program along with their equivalent short forms is placed at the end of the generated Fortran as a series of comment statements. This option prevents its generation.

-f Suppress automatic inclusion of standard definitions file. Macro definitions for the manifest constants used throughout the Subsystem reside in the file "=incl=/swt_def.r.i". 'Rp' will process these definitions automatically, unless the "-f" option is specified.

-g Make a second pass over the code and remove GOTOs to GOTOs generated in Ratfor control structures. Use of this option lengthens preprocessing time significantly, but can result (sometimes) in a 2-5% speedup of the object program.

-h Produce Hollerith-format string constants rather than quoted string constants. This option useful in producing character strings in the proper format needed by your Fortran compiler.

-l Include Ratfor line numbers in the sequence number field of the Fortran output. This may be useful in tracking down the Ratfor statement that caused a Fortran syntax error. By default, no sequence field is generated.

-m Map all identifiers to lower case. When this option is selected, 'rp' considers the upper case letters equivalent to the corresponding lower case letters, except inside quoted strings.

-p Emit subroutine profiling code. When this option is selected, 'rp' places calls to the library routines 't$entr', 't$exit', and 't$clup' in the Fortran output, and creates a text file named "timer_dictionary" containing the names of all subprograms seen by the preprocessor. When the profiled program is run, a file named "_profile" is created that contains timing measurements for each subprogram. The utility program 'profile' may then be used to print a report summarizing the number of times each subprogram was called and the total time spent in each.

-s Short-circuit all logical conditions. The order of evaluation of logical operands in Fortran is unspecified; that is, in the expression "a&b" there is no guarantee that "a" will be evaluated before "b". Occasionally this creates inconveniences; one would like to say something like "if(i>1&array(i)~=0)...". 'Rp' supplies the short-circuit logical operators "&&" and "||" (read "andif" and "orif") for these occasions. Both operators evaluate their left operands; if the value of the logical expression is predictable solely on the basis of the value of the left operand, then the right operand remains unevaluated and the correct expression value is yielded. Otherwise the right operand is evaluated and the proper expression value is determined. The "-s" option may be used to automatically convert all "logical and" operators in a program to "andifs," and all "logical or" operators to "orifs." In addition to improving program portability, this option may also reduce execution

time. By default, however, this option is not in effect.

-t Trace subprograms. When a program preprocessed with the "-t" option is run, an indented trace of the subprograms encountered will be printed on ERROUT. This trace output is generated by calls to the library routine 't$trac' that are inserted automatically by 'rp'.

-v Output "standard" Fortran. This option causes 'rp' to generate only standard Fortran constructs (as far as we know). This option does not detect non-standard Fortran usage in Ratfor source code; it only prevents 'rp' from generating non-standard constructs in implementing its data and control structures. Programs preprocessed with this option are slightly larger and slower; the intermediate Fortran and binary files are approximately 10% larger.

-x Translate character codes. 'Rp' uses the character correspondences in the <translation file> to convert characters into integers when it builds Fortran DATA statements containing EOS-terminated or PL/I strings. If the option is not specified, 'rp' converts the characters using the native Prime character set. The format of the translation file is documented below.

-y Do not output "call swt". This option keeps 'rp' from generating "call swt" in place of all "stop" statements.

The remainder of the command line is used to specify the names of the Ratfor input file(s) and the Fortran output file. If the "-o" option, followed by a filename, is selected, then the named file is used for Fortran output. Any remaining filenames are considered Ratfor source files. If no other file names are specified, standard input is read. If the "-o" option is not specified, then the output filename is constructed from the first input filename by changing a ".r" suffix (if present) to ".f". If the ".r" suffix is not present, the output filename is the input filename followed by the suffix ".f".

The format of the translation file used with the "-x" option is as follows. Each line contains descriptions of two characters: the Prime native character to be replaced, and the character value to replace it. These descriptions may be any one of the following: a single non-blank Prime ASCII character, a number in a format acceptable to 'gctoi' (must be more than one digit), or an ASCII mnemonic acceptable to 'mntoc'. In addition, the character to be replaced may also be the mnemonic "EOS" to indicate that the value of the end-of-string indicator is to be changed. For example, here is

a portion of the table for converting the EBCDIC character set:

```
A 16rc1
B 16rc2
...
Z 16re9
0 16rf0
...
9 16rf9
SP 16r40
```

Examples

```
rp file.r
rp -scp slow_prog.r
rp -o all.f part1.r part2.r part3.r
cat [files .r] | rp >junk.f
```

Files

=temp=/tm?* for various internal temporaries
=incl=/swt_def.r.i for standard Subsystem macro definitions

Messages

See the *User's Guide for the Ratfor Preprocessor*

See Also

profile (1), st_profile (1), c$init (6), c$incr (6), c$end (6), t$entr (6), t$exit (6), t$clup (6), t$time (6), t$trac (6), *Software Tools*, *Software Tools Subsystem Tutorial*, *User's Guide for the Ratfor Preprocessor*

Usage

```
sacl <pathname>
sacl <pathname> {<access pairs>} [-l <pathname>]
sacl <pathname> -a <access category>
```

Description

'Sacl' is a command to manipulate Primos ACL's (Access Control Lists). It may be used to change the access to an object, create an access category, add an item to an access category, or delete an access control list.

An access control list is a set of pairs of names and associated access rights. Each access pair has the following syntax:

```
<name1> := <name2>
      - or -
<name> <op> <rights>
```

Each <name> is either a user name (eg., "system" or "netman"), the name of an accounting group (eg., ".lab" or ".system_staff"), or the special identifier "$rest" indicating everyone not otherwise named. The first form of ACL shown above indicates that the rights for <name1> should be set to exactly the same rights as for <name2>. In the second form of ACL pair, <op> is either the symbol "+=", "-=", or "="; "+=" means to add the indicated rights, "-=" means to remove the indicated rights, and "=" means to set the rights to the indicated permissions.

The <rights> argument consists of either a keyword or symbol, or some combination of letters indicating an access right. Each of the letters and their corresponding rights is as follows:

a -- corresponds to "add" access, that is, the right to create a new file within a directory. Note that once the file is created, the user creating the file can have full read/write access until the first time the file is closed. At that point, the other protections determine access.

d -- corresponds to "delete" access. This access simply allows the user to delete files within a directory. "d" access has no meaning when applied to individual files.

l -- corresponds to "list" access, which is the ability to list the contents of a directory (as in the 'lf' command).

p -- corresponds to "protect" access, which is the ability to set ACL's for objects within the directory.

r -- corresponds to "read" access, the ability to open a file for reading or execute a file.

u -- corresponds to "use" access. Use access means that a user can 'cd' to a command or open a file inferior to the named directory. As example, to open the file /disk1/system/lab/foobar, the user must have (at least) "u" access to the directories "system" and "lab", as well as "r" and/or "w" access to the file "foobar".

w -- corresponds to "write" access. This means that the user has the ability to write to or truncate a file.

Thus, to add "read" and "write" access to a file for user "waldo", the ACL pair "waldo+=wr" could be used, as could "waldo+=rw".

Some special symbols and keywords are recognized by the 'sacl' command. These are:

$all -- corresponds to "adlpruw"

***** -- same as "$all"; "adlpruw" rights

$none -- confers no rights whatsoever

0 -- same as "$none"

$owner -- all rights except protect: "adlruw"

$read -- "lru"

$use -- "alru"

$default -- same rights as currently belong to "$rest"

$def -- same as "$default"

? -- same as "$default"

Also associated with the concept of ACL is the *type* of the ACL. There are basically 5 types of ACL. The first type is a specific ACL which confers protection on one specific file. The second type of ACL is the default specific ACL which is a specific ACL set on an ancestor of the object; if an object is not protected by a specific ACL or an acat, it is protected by a default ACL -- the same ACL which protects its parent.

The third type of ACL is the access category, or "acat". An acat is an ACL which may protect many objects with the same access rights. An acat appears to be a file that cannot be read or written, and its name must end in the 5 character sequence ".acat". An acat need not currently protect any

files or directories; its existence is independent of other objects, unlike a specific ACL.

The fourth type of ACL is the default acat, which is similar in nature to the default specific ACL.

The fifth type of ACL is a priority ACL. This is an ACL set on an entire disk partition by the system administrator. Rights confered by a priority override rights confered by an ACL of any of the other four types. Priority ACLs cannot be set with this command. To set a priority ACL, use the Primos 'spacl' command.

"Sacl <pathname>" reverts the object to default protection; if <pathname> is an acat, it is deleted. The "-a" option adds the object to the named acat. The "-l" option is a "like" option; access rights for the object are the combination of the rights on the object specified with the "-l" option along with the given access pairs. In the second and third form, if <pathname> ends in ".acat" then an access category is created with the indicated rights.

Examples

```
sacl text harold=$read maude:=harold .staff+=r $rest=$none

sacl comm.acat .staff+=* .hackers-=w -l =lbin=

sacl text
```

Messages

"Usage: sacl <pathname> [<acl list>] [-l <pathname> | -a <acat>]" for incorrect usage.

"Cannot set ACL as specified." if the object is unreachable or the user does not have "p" access.

"Object specified after the "-a" is not an acat." if the name of the object after the "-a" option is not a valid acat name.

Bugs

Access categories must end in ".acat". This is not consistent with standard Subsystem naming conventions, but is consistent with Primos standard naming conventions.

See Also

lf (1), lacl (1), sfdata (2), parsa$ (6)

Usage

save

Description

'Save' saves the lexic level 1 shell variables, causing the same action as exiting and reentering the Subsystem.

Its primary use is to save the current values of the shell variables, so that if the Subsystem crashes they will not be lost.

Examples

```
if [nargs]
   set f = [arg 1 | quote]
   save
fi
```

See Also

declare (1), forget (1), vars (1), set (1)

Usage

```
Usage: se [ -t <term> ] { <pathname> | -<option> }
   <term> ::= adm31   | adm3a   | adm42   | adm5    | anp     |
              b150    | b200    | bantam  | bee2    | cg      |
              consul  | forsys  | fox     | gt40    | h19     |
              hp2621  | hp2626  | hp2648  | hp9845  | hz1420  |
              hz1421  | hz1510  | ibm     | info    | isc     |
              microb  | nby     | netron  | pst100  | pt45    |
              regent  | sbee    | sol     | terak   | trs80   |
              ts1     | tvi     | tvt     | vc4404  | vi200   |
              viewpt  | view90  | vt100   | z19

   <opt>  ::= a | c | d[<dir>] | f | g | h[<speed>] |
              i[a | <indent>] | k | l[<lop>] | lm[<col>] |
              m[<opts>] | p[<s | u>] | s<lang> | t[<tabs>] |
              u[<chr>] | v[<col>]  | w[<col>] | -[<row>]
```

Description

In order to understand 'se', you should be familiar with the line editor 'ed'.

'Se' works much like 'ed', accepting the same commands with a few differences. Rather than displaying only a single line from the file being edited (as 'ed' does), 'se' always displays a "window" onto the file. In order to do this, 'se' must be run from a CRT terminal and must be told what sort of terminal it is. If the user entered a valid terminal type when requested to do so upon entry into the Subsystem and that terminal type is recognized by 'se', the "-t <term>" option may be omitted from the 'se' command. Otherwise, either the "-t <term>" terminal type option must be specified, or 'se' will prompt the user for the terminal type. Trying out 'se' will make the screen format evident, so details are not given here.

'Se' is capable of being used from a variety of different terminals. New terminal types are easily added by making small additions to the source code. In general, all that is required of a terminal is that it have the ability to home the cursor (position it to the upper left hand corner of the screen) without erasing the screen's contents, although backspacing, a screen clear function, and arbitrary cursor positioning are tremendously helpful.

The terminals currently supported are the following:

adm31	Lear-Siegler ADM-31.
adm3a	Lear-Siegler ADM-3A.
adm42	Lear-Siegler ADM-42.
adm5	Lear-Siegler ADM-5.

anp	Allen and Paul model 1A.
b150	Beehive International B150.
b200	Beehive International B200.
bantam	Perkin-Elmer 550.
bee2	Beehive International Model 2.
cg	Chromatics Color Graphics Terminal.
consul	ADDS Consul 980.
forsys	Fortunes Systems Terminal.
fox	Perkin-Elmer 1100.
gt40	DEC GT-40 Graphics Terminal with Waugh terminal software
h19	Heath H19 using Heath-mode cursor control (supposedly compatible with DEC VT52's, although this has not been verified)
hp2621	Hewlett-Packard model 2621.
hp2626	Hewlett-Packard model 2626.
hp2648	Hewlett-Packard model 2648.
hp9845	Hewlett-Packard model 9845C color computer with Ray Robinson's terminal software.
hz1420	Hazeltine 1420.
hz1421	Hazeltine 1421.
hz1510	Hazeltine 1510.
ibm	IBM 3101.
info	Infoton 100.
isc	Intelligent Systems Corporation 8001 Color Terminal.
microb	Beehive Microb/DM1A.
nby	Newbury 7009.
netron	Netronics series.
pst100	Prime VT100 look-alike.
pt45	Prime PT45.

regent	ADDS Regent 100 and Regent 40.
sbee	Beehive International Superbee.
sol	Processor Technology Sol computer with software to emulate a Beehive B200.
terak	Terak Microcomputer.
trs80	Radio Shack TRS-80 with Brad Isley's terminal program.
ts1	Falco TS-1.
tvi	Televideo 912/920.
tvt	Southwest Technical Products TV Typewriter II with computer cursor control board and the following cursor controls: right: control-I, left: control-H, up: control-K, home: control-L, erase screen: control-O, down-and-erase-line: control-J.
vc4404	Volker-Craig 4404.
vi200	Visual 200.
viewpt	ADDS Viewpoint 3+.
view90	ADDS Viewpoint 90.
vt100	DEC VT100.
z19	Zenith Z19 (same as Heathkit H19).

The values associated with screen editor options should immediately follow their respective key letters, without intervening blanks between the option letter and the option value. The options that may be specified on the command line correspond to options controlled by the "option" (o) command and are as follows:

Option	*Action*
a	causes absolute line numbers to be displayed in the line number area of the screen. The default behavior is to display upper-case letters with the letter "A" corresponding to the first line in the window.
c	inverts the case of all letters you type (i.e., converts upper-case to lower-case and vice versa). This option causes commands to be recognized only in upper-case and alphabetic line numbers to be displayed and recognized only in lower-case.

d[<dir>] selects the placement of the current line pointer following a "d" (delete) command. <dir> must be either ">" or "<". If ">" is specified, the default behavior is selected: the line following the deleted lines becomes the new current line. If "<" is specified, the line immediately preceding the deleted lines becomes the new current line. If neither is specified, the current value of <dir> is displayed in the status line.

f selects Fortran oriented options. This is equivalent to specifying both the "c" and "t7 +3" (see below) options.

g controls the behavior of the "s" (substitute) command when it is under the control of a "g" (global) command. By default, if a substitute inside a global command fails, 'se' will not continue with the rest of the lines which might succeed. If "og" is given, then the global substitute will continue, and lines which failed will not be affected. Successive "og" commands will toggle this behavior. An explanatory message is placed in the status line.

h[<baud>] lets the editor know at what baud rate you are receiving characters. Baud rates can range from 50 to 19200; the default is 9600. This option allows the editor to determine how many, if any, delay characters (nulls) will be output when the hardware line insert/delete functions of the terminal are being used (if available). Use of the built-in terminal capabilities to insert/delete lines speeds up editing over slow-speed lines (i.e., dialups). Entering 'oh' without an argument will cause your current baud rate to appear on the status line.

i[a | <indent>] selects indent value for lines inserted with "a", "c" and "i" commands (initially 1). "a" selects auto-indent which sets the indent to the value which equals the indent of the previous line. If <indent> is an integer, then the indent value will be set to that number. If neither "a" nor <indent> are specified, the current value of indent is displayed.

k Indicates whether the current contents of your edit buffer has been saved or not by printing either a "saved" or "not saved" message on your status line.

l[<lop>] sets the line number display option. Under control of this option, 'se' continuously displays the value of one of three symbolic line numbers in the status line. <lop> may be any of the fol-

lowing:

. display the current line number

display the number of the top line on the screen

$ display the number of the last line in the buffer

If <lop> is omitted, the line number display is disabled.

lm[<col>] sets the left margin to <col> which must be a positive integer. This option will shift your entire screen to the left, enabling you to see characters at the end of the line that were previously off the screen; the characters in columns 1 through <col> - 1 will not be visible. You may continue editing in the normal fashion. To reset your screen enter the command 'olm 1'. If <col> is omitted, the current left margin column is displayed in the status line.

m[d] [<user>] displays messages sent to you by other users (via the 'to' command) while you are editing. When a message arrives while you are editing, the word "message" appears on your status line. To send other users messages while inside of the editor, you can insert the text of your message into the edit buffer, and then issue the command "line1,line2om <user>", where "line1" and "line2" are the first and last lines, respectively, of where you appended your message in the edit buffer and "<user>" is the login name or process id of the person to whom you want to send a message. The given lines are sent and deleted from the edit buffer. To prevent the lines from being deleted after they are sent, use the command line "line1,line2omd <user>"

p[s | u] converts to or from UNIX (tm) compatibility mode. The "op" command, by itself, will toggle between normal (Software Tools mode) and UNIX mode. The command "opu" will force 'se' to use UNIX mode, while the command "ops" will force 'se' to use Software Tools mode.

When in UNIX mode, 'se' uses the following for its patterns and commands:

?pattern[?] searches backwards for a pattern.

^ matches the beginning of a line.

. matches any character.

^ is used to negate character classes.

% used by itself in the replacement part of a substitute command represents the replacement part of the previous substitute command.

\(<regular expression>\) tags pieces of a pattern.

\<digit> represents the text matched by the tagged sub-pattern specified by <digit>.

\ is the escape character, instead of @.

t copies lines.

y transliterates lines.

~ does the global exclude on markname (see the "!" command, in the help on 'ed').

![<Software Tools Command>] will create a new instance of the Software Tools shell, or execute <Software Tools Command> if it is present (see the "~" command, in the help on 'ed').

All other characters and commands are the same for both UNIX and normal (Software Tools) mode. The help command will always call up documentation appropriate to the current mode. UNIX mode is indicated by the message "UNIX" in the status line.

UNIX mode is available *only* in 'se'. This extension is not available in 'ed'.

s[pma | ftn | f77 | s | f] sets other options for case, tabs, etc., for one of the three programming languages listed. The option "oss" is the same as "ospma" and the option "osf" is the same thing as "osftn" (the corresponding command line options are "-ss" and "-sf"). If no argument is specified the options effected by this command revert to their default value.

t[<tabs>] sets tab stops according to <tabs>. <tabs> consists of a series of numbers indicating columns in which tab stops are to be set. If a number is preceded by a plus sign ("+"), it indicates that the number is an increment; stops are set at regular intervals separated by that many columns, beginning with the most recently specified

absolute column number. If no such number precedes the first increment specification, the stops are set relative to column 1. By default, tab stops are set in every third column starting with column 1, corresponding to a <tabs> specification of "+3". If <tabs> is omitted, the current tab spacing is displayed in the status line.

u[<chr>] selects the character that 'se' displays in place of unprintable characters. <chr> may be any printable character; it is initially set to blank. If <chr> is omitted, 'se' displays the current replacement character on the status line.

v[<col>] sets the default "overlay column". This is the column at which the cursor is initially positioned by the "v" command. <Col> must be a positive integer, or a dollar sign ($) to indicate the end of the line. If <col> is omitted, the current overlay column is displayed in the status line.

w[<col>] sets the "warning threshold" to <col> which must be a positive integer. Whenever the cursor is positioned at or beyond this column, the column number is displayed in the status line and the terminal's bell is sounded. If <col> is omitted, the current warning threshold is displayed in the status line. The default warning threshold is 74, corresponding to the first column beyond the right edge of the screen on an 80 column crt.

-[<lnr>] splits the screen at the line specified by <lnr> which must be a simple line number within the current window. All lines above <lnr> remain frozen on the screen, the line specified by <lnr> is replaced by a row of dashes, and the space below this row becomes the new window on the file. Further editing commands do not affect the lines displayed in the top part of the screen. If <lnr> is omitted, the screen is restored to its full size.

Since 'se' takes its commands directly from the terminal, it cannot be run from a script by using Subsystem I/O redirection, and Subsystem erase, kill, and escape conventions do not exactly apply. In fact, 'se' has its own set of control characters for editing and cursor motion; their meaning is as follows:

Character Action

control-a Toggle insert mode. The status of the insertion indicator is inverted. Insert mode, when enabled, causes characters typed to be inserted at the current cursor position in the line instead of overwriting the characters that were there previously. When insert mode is in effect, "INSERT" appears in the status line.

control-b Scan right and erase. The current line is scanned from the current cursor position to the right margin until an occurrence of the next character typed is found. When the character is found, all characters from the current cursor position up to (but not including) the scanned character are deleted and the remainder of the line is moved to the left to close the gap. The cursor is left in the same column which is now occupied by the scanned character. If the line to the right of the cursor does not contain the character being sought, the terminal's bell is sounded. 'Se' remembers the last character that was scanned using this or any of the other scanning keys; if control-b is hit twice in a row, this remembered character is used instead of a literal control-b.

control-c Insert blank. The characters at and to the right of the current cursor position are moved to the right one column and a blank is inserted to fill the gap.

control-d Cursor up. The effect of this key depends on 'se's current mode. When in command mode, the current line pointer is moved to the previous line without affecting the contents of the command line. If the current line pointer is at line 1, the last line in the file becomes the new current line. In overlay mode (viz. the "v" command), the cursor is moved up one line while remaining in the same column. In append mode, this key is ignored.

control-e Tab left. The cursor is moved to the nearest tab stop to the left of its current position.

control-f "Funny" return. The effect of this key depends on the editor's current mode. In command mode, the current command line is entered as-is, but is not erased upon completion of the command; in append mode, the current line is duplicated; in overlay mode (viz. the "v" command), the current line is restored to its original state and command mode is reentered (except if under control of a global prefix).

control-g Cursor right. The cursor is moved one column to the right.

control-h Cursor left. The cursor is moved one column to the left. Note that this *does not* erase any characters; it simply moves the cursor.

control-i Tab right. The cursor is moved to the next tab stop to the right of its current position.

control-k Cursor down. As with the control-d key, this key's effect depends on the current editing mode. In command mode, the current line pointer is moved to the next line without changing the contents of the command line. If the current line pointer is at the last line in the file, line 1 becomes the new current line. In overlay mode (viz. the "v" command), the cursor is moved down one line while remaining in the same column. In append mode, control-K has no effect.

control-l Scan left. The cursor is positioned according to the character typed immediately after the control-l. In effect, the current line is scanned, starting from the current cursor position and moving left, for the first occurrence of this character. If none is found before the beginning of the line is reached, the scan resumes with the last character in the line. If the line does not contain the character being looked for, the message "NOT FOUND" is printed in the status line. 'Se' remembers the last character that was scanned for using this key; if the control-l is hit twice in a row, this remembered character is searched for instead of a literal control-l. Apart from this, however, the character typed after control-l is taken literally, so 'se's case conversion feature does not apply.

control-m Newline. This key is identical to the NEWLINE key described below.

control-n Scan left and erase. The current line is scanned from the current cursor position to the left margin until an occurrence of the next character typed is found. Then that character and all characters to its right up to (but not including) the character under the cursor are erased. The remainder of the line, as well as the cursor are moved to the left to close the gap. If the line to the left of the cursor does not contain the character being sought, the terminal's bell is sounded. As with the control-b key, if control-n is hit twice in a row, the last character scanned for is used instead of a literal control-n.

control-o Skip right. The cursor is moved to the first position beyond the current end of line.

control-p Interrupt. If executing any command except "a", "c", "i" or "v", 'se' aborts the command and reenters command mode. The command line is not erased.

control-q Fix screen. The screen is reconstructed from 'se's internal representation of the screen.

control-r Erase right. The character at the current cursor position is erased and all characters to its right are moved left one position.

control-s Scan right. This key is identical to the control-l key described above, except that the scan proceeds to the right from the current cursor position.

control-t Kill right. The character at the current cursor position and all those to its right are erased.

control-u Erase left. The character to the left of the current cursor position is deleted and all characters to its right are moved to the left to fill the gap. The cursor is also moved left one column, leaving it over the same character.

control-v Skip right and terminate. The cursor is moved to the current end of line and the line is terminated.

control-w Skip left. The cursor is positioned at column 1.

control-x Insert tab. The character under the cursor is moved right to the next tab stop; the gap is filled with blanks. The cursor is not moved.

control-y Kill left. All characters to the left of the cursor are erased; those at and to the right of the cursor are moved to the left to fill the void. The cursor is left in column 1.

control-z Toggle case conversion mode. The status of the case conversion indicator is inverted; if case inversion was on, it is turned off, and vice versa. Case inversion, when in effect, causes all upper case letters to be converted to lower case, and all lower case letters to be converted to upper case. Note, however, that 'se' continues to recognize alphabetic line numbers in upper case only, in contrast to the "case inversion" option (see the description of options above). When case inversion is on, "CASE" appears in the status line.

control-_ Insert newline. A newline character is inserted before the current cursor position, and the cursor is moved one position to the right. The newline is displayed according to the current non-printing replacement character (see the "u" option).

control-\ Tab left and erase. Characters are erased starting with the character at the nearest tab stop to the left of the cursor up to but not including the character under the cursor. The rest of the line, including the cursor, is moved to the left to close the gap.

control-^ Tab right and erase. Characters are erased starting with the character under the cursor up to but not including the character at the nearest tab stop to the right of the cursor. The rest of the line is then shifted to the left to close the gap.

NEWLINE Kill right and terminate. The characters at and to the right of the current cursor position are deleted, and the line is terminated.

DEL Kill all. The entire line is erased, along with any error message that appears in the status line.

ESC Escape. The ESC key provides a means for entering 'se's control characters literally as text into the file. In fact, any character that can be generated from the keyboard is taken literally when it immediately follows the ESC key. If the character is non-printing (as are all of 'se's control characters), it appears on the screen as the current non-printing replacement character (normally a blank).

The set of control characters defined above can be used for correcting mistakes while typing regular editing commands, for correcting commands that have caused an error message to be displayed, for correcting lines typed in append mode, or for inline editing using the "v" command described below.

There are a few differences in command interpretation between the regular editor and 'se'. The only effect of the "p" command in 'se' is to position the window so that as many as possible of the "printed" lines are displayed while including the last line in the range. In fact, the window is always positioned so that the current line is displayed. Typing a "p" command with no line numbers positions the window so that the line previously at the top of the window is at the bottom. This can be used to "page" backwards through the file. The ":" command, (which in the regular editor prints about a screenfull of text starting with a specified line), positions the window so it begins at the specified line, and leaves the current line pointer at this line. Thus, a ":" can be used to page forward through the file.

The "overlay" (v) command in the regular editor 'ed' only allows the user to add onto the end of lines, and can be terminated before the stated range of lines has been processed by entering a period by itself, as in the "append" command. But in 'se', this command allows arbitrary changes to be made to the lines, and the period has no special meaning. To abort before all the lines in the range have been covered, use the "funny return" character (ctrl-F). Doing this restores the line containing the cursor to the state it was in before the "v" command was started.

'Se' has a "draw box" command that can be used as an aid for preparing block diagrams, flowcharts, or tables. The general form is

```
top-line,bottom-line zb left-col,right-col character
```

For example, "1,10 zb 15,25 *" would draw a box 10 lines high and 11 columns across, using asterisks. The upper left corner of the box would be on line 1, column 15, and the lower right corner on line 10, column 25. The bottom-line may be omitted to draw horizontal lines, and the right-col may be omitted to draw vertical lines. If the "character" at the end of the command is omitted, it is assumed to be a space, thus allowing erasure of a box or line.

When the "write" command ("w") is used with a file name that is different from the name 'se' is remembering, the message "file already exists" will be displayed if the output file is already present. If the command is entered again (by typing a NEWLINE), 'se' will perform the write, destroying the existing file. To circumvent the warning, enter the write command suffixed by "!" (just like "quit" or "enter") and 'se' will always write to the file.

When 'se' starts up, it tries to open the file "=home=/.serc". If that file exists, 'se' reads it, one line at a time, and executes each line as a command. If a line has "#" as the *first* character on the line, or if the line is empty, the entire line is treated as a comment, otherwise it is executed. Here is a sample ".serc" file:

```
# turn on unix mode, tabs every 8 columns, auto indent
opu
ot+8
oia
```

The ".serc" file is useful for setting up personalized options, without having to type them on the command line every time, and without using a special shell file in your bin. In particular, it is useful for automagically turning on UNIX mode for Software Tools users who are familiar with the UNIX system.

Command line options are processed *after* commands in the ".serc" file, so, in effect, command line options can be

used to over-ride the defaults in your ".serc" file.

NOTE: Commands in the ".serc" file do *not* go through that part of 'se' which processes the special control characters (see above), so *do not* use them in your ".serc" file.

For a list of commands accepted by both 'se' and 'ed', see either the Reference Manual entry for 'ed' ("help ed") or the *Introduction to the Software Tools Text Editor*.

'Se' has an extended line number syntax. In general, whatever appears in the left margin on the screen is an acceptable line number and refers to the line displayed in that row on the screen. In particular, upper case letters are often used. Also, the line number element "#" is interpreted as being the number of first line of the current screen.

Examples

```
se -t b200 -c -w70 -t+6
se -t consul textfile
```

Files

=home=/.serc 'se' start up command file

=temp=/se<line><sequence_number> for scratch file

=doc=/se_h/?* help scripts for the "h" command

=home=/se.logout for saving the buffer on a LOGOUT$ condition

Messages

Many. Most are self-explanatory.

Bugs

Cannot be run from a script.

Cannot specify tab stops as the first option if no terminal type is specified first on the command line.

The auto-indent feature does not recognize a line consisting of just blanks and then a "." to terminate input, when the "." is not in the same position as the first non-blank character of the previous line.

Should be changed to use the 'vth' terminal operations library, instead of having code hard-wired in for each

terminal type. Unfortunately, 'vth' isn't quite up to this.

See Also

ed (1), *Introduction to the Software Tools Text Editor*

Usage

```
sema -(w | n | t | d | c) {<semaphore>}
sema -o {<pathname>} [-i <integer>]
   <semaphore> ::= integer
```

Description

'Sema' gives access to Prime's user semaphores, which are available to all users. There is no mechanism to assure that semaphores are used properly (i.e. - preventing deadlock or race conditions).

'Sema' performs the function indicated by its argument, on the semaphore number or pathname given. When multiple semaphores or pathnames are supplied, the operation given is performed on each argument in the order listed on the command line. The functions available are:

-w (wait) increment the semaphore's counter. If the resulting value is positive, enqueue on the semaphore's waiting list and block execution (sleep) until awakened by some other process.

-n (notify) decrement the counter. If the result is positive or zero, dequeue a process from the waiting list and wake it up.

-t (test) print the value of the counter (in decimal) on standard output.

-d (drain) initialize a semaphore for use. Set the counter to zero, dequeue all waiting processes and wake them up.

-o (open) initialize a semaphore for use. This initializes named semaphores for use and returns the semaphore numbers on standard output. A semaphore is opened only if the user has read access to the pathname given and if the pathname is on the current system. One semaphore is opened for each pathname specified. All semaphores opened on the same pathname are the same. This allows the user to restrict access to the semaphores by restricting access access to the files used to open the semaphores. This may be achieved with access control lists or passworded directories. The numbered semaphores (1-64) are considered always open and any attempt to open one of them will result in an error.

-i (initialize) when used with the "-o" (open) option causes the semaphores that are opened to be initialized to the value specified. This initialization only takes place the first time the semaphore is opened. If multiple users have opened a semaphore, only first time

cause the initialization to take place. Since initializing a semaphore to a positive value does not make sense, only non-positive values are allowed. If this option is omitted, the default is 0.

-c (close) close a named semaphore. This removes the user's number from the list of current users of the semaphore and makes the semaphore unavailable for further use for that user. Since the numbered semaphores are always open, any attempt to close one of them will result in an error.

Examples

```
sema -w 1
sema -n 1 2 3
sema -t [iota 62]
sema -d -32
set sema_number = [sema -o //system/restricted_file -i -1]
sema -c [sema_number]
```

Messages

"Usage: sema ..." for nonsensical arguments.
"no available semaphores" when all named semaphores are allocated.
"<pathname>: invalid semaphore name" when the semaphore used has not been opened.
"<pathname>: semaphore overflow" when another notify would exceed the limit on the semaphore.
"<pathname>: on a remote disk" when the pathname is not on the current system.
"<pathname>: file not found" when the file used to open a semaphore cannot be read.
"<pathname>: can't open semaphore" when some unknown reason causes an error.
"<integer>: initial value greater than 0" when an initial value is specified that is greater than 0.

See Also

Prime's Subroutines Reference Guide (*DOC3621-190P*), section 21, for a complete description of semaphores as implemented in Primos and for a description of the system calls used by 'sema'.

Peter Freeman: *Software Systems Principles*, for a discussion of how semaphores of this type can be used.

Usage

```
sep <prog> ( <module> { <module> }
           | -all | -cat | -stacc | -xref | -names | -link
           | -mklink     | -print [ <spool options> ]
           )
```

Description

'Sep' is a shell program that assists in maintaining large Ratfor programs with modules stored in separate files. 'Sep' insists on a number of file naming conventions so that it can locate all the files for a given program. First, all files must be stored in the same directory, and that directory must be the current directory when 'sep' is invoked. The program name <prog> is part of each source file name and is the name of the executable file produced. A module name <module> is an arbitrarily selected name for a group of routines contained in the same file. A module is the smallest unit that can be compiled separately. There may be an arbitrary number of modules.

'Sep' requires a number of files to be able to successfully compile a program. All definitions global to the program must be placed in the file "<prog>_def.i"; they are included each time a module is preprocessed. Even if there are no definitions for a particular program, this file must be present. A linkage statement naming all subroutine, function, and common block names must be present in the file "<prog>_link.i". (If you choose, 'sep' will build the file containing the linkage statement for you; see the "-mklink" option.) Finally, the main program (and other routines, if desired) must be present in a file named "<prog>.r".

There are a number of other files that can be present and will be used by 'sep' in compiling a program. Files containing groups of subprograms should be named "<prog>_<module>.r". Files of this type define the separately compiled "modules". A Stacc parser may be present in the file "<prog>.stacc"; it is converted to the Ratfor module "<prog>_stacc.r" on command. The definitions generated from the Stacc code are placed in the file "<prog>_def.stacc.i"; there must be an include statement for this file in "<prog>_def.i". If the program has an include file for common blocks, it must be named "<prog>_com.i" or "<prog>_com.r.i", but 'sep' does not automatically include it during compilation.

Other files that may be present are "<prog>.ldproc" which should contain an 'ld' command for linking the binary files; if this file is not present, 'sep' links all the binary files in the directory with names beginning with "<prog>". The file "<prog>.rpopts" may also be present; it contains command line options such as "-t" to be added to all calls to 'rp'. The file "<prog>.fcopts" contains command line

options to be presented to 'fc'. If the file "<prog>.ldopts" is present and 'sep' generates the 'ld' statement itself, the contents of this file are added to the end of the 'ld' command line. Usually this file contains the string "-t -m" so that a load map is produced.

'Sep' performs a number of different operations, depending on the arguments given to it.

sep <prog> <module> { <module> }
Each named module are preprocessed and compiled. The main program can be named with an argument containing the null string (i.e. ""). All the program's binary modules are then linked together.

sep <prog> -all
If a Stacc parser is present, it is converted to Ratfor. All the program's Ratfor modules are preprocessed and compiled, and then all the program's binary modules are linked together.

sep <prog> -stacc
The Stacc parser is converted to Ratfor.

sep <prog> -link
All the program's binary modules are linked together. If a file named "<prog>.ldproc" exists, it is used to perform the linking. Otherwise, all binary files in the directory with names beginning with "<prog>" are linked together. The text in the file "<prog>.ldopts", if present, is placed at the end of the generated 'ld' command.

sep <prog> -mklink
Call 'link' to build a Ratfor linkage statement for the program in the file "<prog>_link.i"

sep <prog> -cat
All the source code files area printed on standard output. No file is printed more than once.

sep <prog> -print [<spool options>]
All the source code files for the program are printed on the line printer using 'pr'. No file is included more than once. <Spool options> are used to determine the disposition of the output; they are any options acceptable to 'pr'.

sep <prog> -names
The names of all source code files are printed on standard output. No file name is printed more than once.

sep <prog> -xref
All Ratfor source modules is run through 'xref'. The listing from 'xref' appears on standard output.

Examples

```
sep rp -stacc
sep rp bool init other stacc
sep xref -all
sep fmt "" fill
sep nfmt -print
```

Files

"<prog>.r" for file containing main program.
"<prog>_def.i" for file containing global definitions.
"<prog>_link.i" for file containing the "linkage" statement.
"<prog>_<module>.r" for file containing the Ratfor source code for the module named <module>.
"<prog>.ldopts" for file containing program's link options (optional).
"<prog>.rpopts" for file containing program's Ratfor options (optional).
"<prog>.fcopts" for file containing program's Fortran options (optional).

Messages

"Usage: sep <prog> <options>" for missing program or options.

Bugs

Currently undergoing development. The user interface will probably be changed in the future.

Cannot handle more than about 50 modules in a program.

When presented with errors, it displays the lack of robustness of a typical shell file.

See Also

fc (1), stacc (1), ld (1), link (1), pr (1), xref (1), bind (3)

Usage

```
set [ <variable> ] = [ <string> ]
```

Description

'Set' can be used to assign arbitrary values to shell variables. The first argument is the name of the variable to be set; if absent, the value is printed on standard output instead of being assigned. The third argument is the value to be assigned to the variable; if absent, one line is read from standard input, and the text thus entered becomes the string to be assigned. The string may contain unprintable characters in a mnemonic form. This consists of a '<' sign followed by an ascii mnemonic and terminated by a '>' symbol. To prevent a symbol from being interpreted, simply escape the '<' with and '@' sign. For example to set the variable lfcr to a linefeed and a carriage return, use:

```
set lfcr = "<lf><cr>".
```

If <variable> exists in the current scope or any surrounding scope, then its value is altered by 'set'; otherwise, it is created at the current lexical level and then the value is assigned.

Examples

```
set i = 0
set i = [eval i + 1]
set lfcr = "<lf><cr>"
set nolfcr = "@<lf>@<cr>"
set atsign = "@"
set response =
```

See Also

declare (1), forget (1), vars (1), save (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

sh

Description

'Sh' is an entry into the Subsystem command interpreter. When invoked, it reads commands from standard input one until it encounters end-of-file, thus making it useful in pipelines.

The functions of the command interpreter are far too complex to describe here; see the *User's Guide for the Software Tools Subsystem Command Interpreter* for a tutorial and more detailed information.

Examples

```
files .r$ | change % "ar -u arch " | sh
command_file> sh >command_output
```

Files

=temp=/t?* for pipe temporaries, function returns, etc.
=gossip=/<login_name> for 'to' messages
=temp=/cn<line><sequence_number> for compound nodes

Messages

* Many. See the User's Guide.

See Also

User's Guide for the Software Tools Subsystem Command Interpreter, *Software Tools Subsystem Tutorial*

Usage

```
shtrace  { on | debug | all | value | <octal_integer>
         | cl | command_line
         | cn | compound_node
         | ex | execution
         | fn | function
         | it | iteration
         | lo | location
         | ls | linked_strings
         | nd | node
         | ou | onunit
         | pd | port_descriptor
         | sr | sv_restore
         | ss | single_step
         | sv | sv_save }
```

Description

'Shtrace' is a debugging aid intended for those who maintain the Subsystem, particularly its command interpreter. Because of its value in debugging shell programs, it has been released for general use, with the warning that it may change without notice.

In essence, 'shtrace' prints the status of a command line or its environment as the command is processed by the shell. Since this involves many operations, there are many potential "checkpoints" within the command interpreter. The 'shtrace' options are intended to pick out the most vital points along a command's path from entry to execution.

Each option is specified as a single character-string argument, which, except for special cases, may be abbreviated with a two-character mnemonic. These options are as follows:

command_line (cl)
: The command line is printed as it is read, without processing. (The output from this and other options is preceded by a number in brackets, e.g. [2.1]; the integer part is the lexic level of the command (the number of command inputs currently active), while the fractional part is the node number of the node within its network.)

compound_node (cn)
: The command line is printed after compound nodes have been replaced by temporary command files.

function (fn)
: The command line is printed after all function calls have been evaluated.

iteration (it)

The command line is printed after all iteration groups have been expanded.

execution (ex)
The network about to be executed is printed. No compound nodes, functions, or iterations will appear in this version of the command.

node (nd)
The node about to be executed is printed, along with its arguments.

single_step (ss)
Just before a network is executed, the command interpreter will stop, prompt for input with the string "continue?", and wait for a reply from the user. Inputs beginning with "n" or "N" cause execution to be terminated; other inputs cause processing of commands to be continued. (This option is most useful when used in conjunction with the "command_line" option.)

port_descriptor (pd)
The port descriptor table used by the shell to assign files to the standard input and output ports is dumped in symbolic format. Along with a mnemonic for each standard port is printed the file unit associated with it and the source or destination of the data (file or pipe).

sv_save (sv)
This option causes the shell variables and their values to be printed whenever they are saved on disk (e.g., when a 'stop' or 'save' command is executed).

sv_restore (sr)
This option causes the shell variables and their values to be printed whenever they are loaded from disk (e.g., when the Subsystem is started by the 'swt' command).

linked_strings (ls)
Whenever garbage collection takes place in the linked string storage area, a summary of the memory structure is printed.

location (lo)
Following the execution of each node, the full pathname of the command just executed is printed.

onunit (ou)
Whenever the shell's default onunit is invoked, the condition that was raised is printed.

In addition to these options, there are three "shorthand"

options for specifying common combinations. The "on" option turns on the "node" and "execution" traces; "debug" turns on "node", "execution", and "single_step"; "all" turns on all traces available.

All traces may be turned off by executing 'shtrace' with no arguments.

'Shtrace' prints on standard output one an octal integer reflecting the last state of the trace control variable, suitable for saving in a shell variable or otherwise recording for later use. The special option "value" may be used to simply print the current value of the control variable without changing it. If an octal integer is given as an argument to 'shtrace', that bit pattern is assigned to the trace control variable. Thus, a user's trace options may be changed and then reset to their original state.

Examples

```
shtrace on
shtrace
shtrace cn ss pd
set old_shtrace = [shtrace nd]
shtrace [old_shtrace]
```

See Also

dump (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

slice (-i | -x) <start_pattern> [(-i | -x) <end_pattern>]

Description

'Slice' searches its standard input for a line matching the pattern <start_pattern> and copies through to standard output all the lines from that one to the first line matching the pattern <end_pattern>.

The "-i" and "-x" options control the inclusion and exclusion (respectively) of the line matching the associated pattern.

If the <end_pattern> and its associated inclusion flag are missing, the copy operation continues until end-of-file is encountered.

'Slice' is useful for pulling out chunks from well-structured files, like the documentation files for the Subsystem Reference Manual. For example, "slice -i %.bu -x %.sa" would copy the "Bugs" section out of a Reference Manual entry.

Examples

```
slice.d> slice -i .bu -x .sa | fmt
slice -i % -x %-EOF$
```

Messages

"Usage: slice ..." for invalid argument syntax.

Bugs

Doesn't handle lines longer than MAXLINE.

See Also

cto (1), find (1), match (2), makpat (2)

Usage

```
sort {-d | -r} { <pathname> }
```

Description

'Sort' is a rather straightforward program that sorts the contents of the files named in its argument list and writes the result on its first standard output port. By default, lines are sorted in ascending order on the basis of ASCII collating sequence, using the entire line as a key. If the "-d" option is specified, dictionary collating sequence (upper and lower case are equivalent, punctuation and special characters are ignored) is used. If the "-r" option is specified, lines are sorted in descending order.

If no pathname arguments are given, or if the pathname "-" appears as an argument, standard input one is used for input. Thus, 'sort' may be used as a filter.

'Sort' uses a combination of 'quicksort' and merge; it is taken directly from *Software Tools*.

Examples

```
lf -c | sort | cat -n
sort -d wordlist dictionary >new_dictionary
files .r$ | sort -r | print -n
```

Files

=temp=/st$?* for sort temporary files

Messages

"<file>: can't open" for unreadable files.
"<file>: can't create" if temporary file can't be created.

See Also

Software Tools

Usage

source { <command> | <subroutine> }

Description

The 'source' command writes a copy of the source code of the named commands or subroutines to standard output, in 'cat -h' format.

Examples

source help

Files

=src=/misc/srcloc for locations of all source code files

Messages

"Usage: source ..." if called with no arguments

Bugs

Not exactly blindingly fast.

See Also

cat (1), locate (1)

Usage

```
sp {<file_spec>} [/ {<option>}]

<file_spec> ::= <filename> | -[<stdin_number>] |
                -n(<stdin_number> | <filename>)
<option> ::=   -a <location>
             | -b <banner>
             | -c <copies>
             | -d <defer_time>
             | -f
             | -h
             | -j
             | -n
             | -p <paper_type>
             | -r
             | -s
```

Description

'Sp' allows users of the Subsystem to send output to the on-site line printer.

Data to be printed is selected by a number of <file_spec>s. See 'cat' for detailed information on the semantics of this construct.

A number of spooler control operations may be specified on the command line after the files to be printed, provided the files and options are separated by a single argument consisting only of a slash. The options presently available are:

-a	Print the file on system <location>
-b	Change the output heading to <banner>
-c	Produce <copies> duplicates
-d	Do not print until <defer_time>
-f	Use FORTRAN forms control
-h	Suppress header page
-j	Suppress final page eject
-n	Generate line numbers in left hand margin
-p	Do not print until operator mounts <paper>
-r	Use raw forms control
-s	User standard PRIMOS forms control

Examples

```
sp file.l
fmt report | os | sp / -f
sp stuff / -d 23:59 -f -c 30
```

Files

//spoolq/prt??? for spooler queue file
//spoolq/q.ctrl for spool queue

See Also

pr (1), open (2), lopen$ (6)

Usage

```
speling { <filename> }
```

Description

'Speling' places on its first standard output a list of all the words in the named files (or standard input, if no files are named) that are not in the dictionary "=dictionary=". A "word" is a contiguous string of letters. 'Speling' is a shell program; the user is referred to its text to see how it works. To see how the word file is constructed, see the files in the directory "=aux=/spelling".

Examples

```
speling report >sp.errs
speling part1 part2 part3 >bogus_words
```

Files

=dictionary= for dictionary of correct spellings

Bugs

This command is superseded by the faster and more functional 'spell' command.

See Also

common (1), sort (1), spell (1), tlit (1), uniq (1)

Usage

```
spell  [ -(f | v) ]  { <pathname> }
```

Description

'Spell' can be used to check all the words in a document for presence in a dictionary. Thus, it provides an indication of words that may be misspelled.

'Spell' has two modes of operation, controlled by the absence or presence of the "v" option. If the "v" option is not specified, 'spell' simply produces a list of words that it thinks are misspelled. If "v" is specified, 'spell' will also print the original input text, following each line with a line containing possibly misspelled words. (This is intended to make the erroneous words easier to locate.) Each text line is preceded by a blank, while each word list line is preceded by a plus sign ('+'); if the output is redirected to /dev/lps/f, this causes all misspelled words to be boldfaced.

Normally, 'spell' ignores input lines that begin with a period, since those are normally text formatter control directives. However, the "-f" option can be used to force 'spell' to process those lines.

If the template =new_words= is defined, 'spell' will treat it as the pathname of a file into which it will append all words that it could not find. This file should be periodically sorted, uniq'ed, and then checked by hand against a dictionary. Any real words found in this manner should be added to =dictionary=.

'Spell' supersedes the slower and less functional 'speling' command.

Examples

```
spell report
spell -v report >/dev/lps/f
spell -f arbitrary_text | pg
spell part1 part2 part3 >new_words
files .fmt$ | spell -n
```

Messages

"Usage: spell ..." for improper arguments.
"in dsget: out of storage space" if there are too many misspelled words to handle.

Bugs

Could stand to be made smarter about suffixes and prefixes. At least it does now handle words with a trailing "'s".

See Also

speling (1)

Usage

```
splc {-<option>[<level>]} <input file>
         [-b [<binary file>]]
         [-l [<listing file>]]
         [-z <SPL option>]
   <option> ::= c | d | e | f | h | k | m | n |
                o | p | q | r | s | v | w | x
```

Description

'Splc' serves as the Subsystem interface to the Primos SPL compiler (SPL). It examines its option specifications and checks them for consistency, provides Subsystem-compatible default file names for the listing and binary files as needed, and then produces a Primos SPL command and causes it to be executed.

Options

The general structure of an 'splc' option is a single letter, possibly followed by a "level number" indicating the extent to which an option should be employed. The following list outlines the options and the meanings of their various levels. The first line of each description contains the option letter followed by its default level enclosed in parentheses, the range of available levels enclosed in square brackets, and a brief description of the option's purpose. In all cases, when an option is specified without a level number, the maximum allowable value is assumed.

-c(0) [0..1] - Case mapping.

Level 0 forces case to be insignificant in identifiers. Upper case identifiers are considered the same as lower case identifiers.

Level 1 cause case to significant in identifiers. Upper case identifiers are considered different from lower case identifiers.

-d(0) [0..2] - Debugging control.

Level 0 prevents all debugging information from being included in the generated code. A program so compiled may not be used with the source level debugger.

Level 1 allows limited debugging information to be included in the generated code, but does not interfere with optimization.

Level 2 causes complete debugging information to be included in the generated code and inhibits optimization. (Cannot be used when the "-o" option is

specified with a level greater than zero.)

-e(1) [0..1] - Error listing on terminal.

Level 0 inhibits the printing of compilation errors on the user's terminal.

Level 1 causes compilation errors to be printed on the terminal.

-f(2) [0..3] - Symbol table map and offset map control.

Level 0 inhibits the generation of either a symbol table map or a storage offset map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 1 causes the generation of a map listing the storage offset of each program variable, but still inhibits the generation of a a symbol table map. (Cannot be used when the "-x" option is specified with a level greater than zero.)

Level 2 causes the generation of a map listing the symbol names appearing in the program, but inhibits the generation of a storage offset map.

Level 3 causes the generation of both the symbol table and storage offset maps.

-h(0) [0..1] - Huge (multi-segment) arrays.

Level 0 insures that dummy arrays and array parameters will not be treated as multi-segment arrays.

Level 1 causes references to dummy arrays and array parameters to generate code that will work even if the arrays are larger than one segment (64K words) in length.

-k(0) [0..1] - Compilation statistics.

Level 0 inhibits the display of compilation statistics on the terminal.

Level 1 causes the display of compilation statistics on the terminal.

-m(2) [2..2] - Addressing mode.

Level 2 implies 64V addressing mode. At present this is the only addressing mode fully supported under the Subsystem.

-n(1) [0..1] - Nesting level indicator.

Level 0 inhibits the printing of the nesting level of each statement on the listing.

Level 1 causes the printing of the nesting level of each statement.

-o(1) [0..1] - Optimization control.

Level 0 turns off all optimizations.

Level 1 turns on optimizations. This option cannot be used with full debugging (-d2).

-p(0) [0..1] - Quick call of internal subroutines.

Level 0 causes all internal subroutines to be called with the normal procedure call (PCL) mechanism.

Level 1 causes internal subroutines to be "quick called" (shortcalled) whenever possible. This option cannot be used with full debugging (-d2).

-q(1) [0..1] - Suppress warning messages.

Level 0 inhibits the display of compiler warning messages.

Level 1 allows the display of compiler warning messages.

-r(0) [0..1] - Range checking.

Level 0 inhibits run-time checking of subscripts and substrings.

Level 1 causes the compiler to insert code for the run-time checking of subscripts and substrings.

-s(1) [0..1] - Constant copying for subroutine calls.

Level 0 inhibits the copying of constants into temporary variables for passing as subroutine parameters.

Level 1 causes the compiler to copy constants into temporary variables before calling subroutines.

-v(1) [0..2] - Listing verbosity.

Level 0 prevents the listing of source code, but allows the listing of error messages and statements that caused them.

Level 1 generates a full source code listing.

Level 2 generates a full source code listing plus a representation of the machine code generated for each statement.

-w(0) [0..1] - Generate floating round instructions.

Level 0 does not generate floating round (FRN) instructions.

Level 1 cause a floating round (FRN) instruction to be generated before every floating store (FST) instruction in the code produced by the SPL compiler. This option improves the accuracy of single precision floating point calculations at some slight run-time performance expense.

-x(1) [0..1] - Cross-reference listing control.

Level 0 inhibits the generation of a cross-reference.

Level 1 causes the compiler to generate a cross-reference listing. (Cannot be used when the "-f" option is specified with a level less than two.)

In addition to the options above, the "-z" option allows the explicit passing of a string verbatim into the command line.

File Control

The "-b" option is used to select the name of the file to receive the binary object code output of the compiler. If a file name follows the option, then that file receives the object code. (Note that if "/dev/null" is specified as the file name, no object code will be produced.) If the option is not specified, or no file name follows it, a default filename is constructed from the input filename by changing its suffix to ".b". For example, if the input filename is "prog.spl", the binary file will be "prog.b"; if the input filename is "foo", the binary file will be "foo.b".

The "-l" option is used to select the name of the file to receive the listing generated by the compiler. If a file name follows the option, then that file receives the listing. The file name "/dev/null" may be used to inhibit the listing; "/dev/tty" to cause it to appear on the user's terminal; "/dev/lps" to cause it to be spooled to the line printer. If the "-l" option is specified without a file name following it, a default filename is constructed from the input filename by changing its suffix to ".l". For example, if the input filename is "gonzo.spl", the listing file will be "gonzo.l"; if the input filename is "bar", the listing file will be "bar.l". If the "-l" option is not used, no listing is produced.

The input filename may be either a disk file name (conventionally ending in ".spl"). or the device "/dev/tty", in

which case input to the compiler is read from the user's terminal.

In summary, then, the default command line for compiling a file named "file.spl" is

```
splc -c0d0e1f2h0k0m2n1o1p0q1r0s1v1w0x1 _
     file.spl  -b file.b  -l /dev/null
```

which corresponds to the SPL command

```
spl -i *>file.spl -b *>file.b -l no
```

Examples

```
splc file.spl
splc -kf dmach.spl
splc -x dmach.spl -b b_dmach -l l_dmach
splc -m2 v_mode_prog.spl -z"-newopt"
```

Messages

"Usage: splc ..." for invalid option syntax.
"level numbers for -<option> are <lower bound> to <upper bound>" if an out-of-range level number is specified.
"missing input file name" if no input filename could be found.
"<name>: unreasonable input file name" if an attempt was made to read from the null device or the line printer spooler.
"<name>: unreasonable binary file name" if an attempt was made to produce object code on the terminal or line printer spooler.
"inconsistency in internal tables" if the tables used to process the options are incorrectly constructed. This message indicates a serious error in the operation of 'splc' that should be reported to your system administrator.

Numerous other self-explanatory messages may be generated to diagnose conflicts between selected options.

Bugs

'Splc' pays no attention to standard ports.

See Also

ld (1), splcl (1), bind (3)

Usage

splcl <program name> [<'ld' options>] [/ <'splc' options>]

Description

'Splcl' is a shell file that invokes the Primos SPL compiler and the Primos segmented loader. If 'splcl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".spl", although in <program name> it may be specified with or without the ending ".spl". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'splc' will be called with the <'splc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
splcl myprog.spl
splcl myprog subs.b subs2.b -l mylib
splcl myprog / -ok -l mylist
```

Messages

"<program name>.spl: cannot open"

Bugs

An alternate binary file name cannot be specified.

See Also

splc (1), ld (1), bind (3)

Usage

```
ssr [ <new search rule> ]
```

Description

The 'ssr' command allows users of the Subsystem shell to specify which directories and command libraries should be searched in the process of invoking a command. If an argument is specified, it becomes the new search rule; if not, the search rule remains unchanged. In either case, the resulting search rule is printed.

The search rule consists of a string of 'elements' separated by commas. Each element is a template that specifies either a special command library or a directory to be searched. In the process of invoking a command, the shell examines each element in the search rule from left to right. In each element, it replaces all ampersands ("&") with the command name specified by the user. It then searches for a command by that name. The shell keeps examining elements of the search rule until a command is located or the end of the search rule is reached.

For example, the default search rule,

```
'^int,^var,&,=lbin=/&,=bin=/&'
```

specifies the following directories and libraries:

^int	Internal commands - those commands recognized and executed by the shell itself.
^var	Shell variables - the effect of 'executing' a variable is to print the value of the variable on standard output 1.
&	A single ampersand specifies the current working directory.
=lbin=/&	The directory '//lbin', where locally-supported commands are stored.
=bin=/&	The directory '//bin', where standard Subsystem commands reside. Note that the trailing slash and ampersand MUST be included in the search rule.

Examples

```
ssr
ssr "^var,^int,&,//newshbin/&,//newbin/&"
ssr "^var,//project_lib/&"
```

See Also

set (1), declare (1), forget (1), vars (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

```
stacc  [ Ratfor | C | Pascal | Pl/1 | Plp ]
```

Description

'Stacc' (STill Another Compiler-Compiler) is a simple parser generator designed to reduce the effort involved in building recursive-descent parsers. Its development was motivated by two things: (1) the large number of ad-hoc recursive descent parsers constantly being written and re-written at the Georgia Tech installation; (2) the desire to quickly generate an SSPL compiler in SSPL for microprocessor software development work. Its design was inspired both by 'yacc' (Yet Another Compiler-Compiler) on Unix and the GTL Syntax Parser on the now defunct Georgia Tech Burroughs B5500.

Basic Theory

Given an LL(1) grammar written in an extended BNF, 'stacc' generates a very simple top-down, recursive-descent parser. Many excellent references are available on the subject of such parsers; the following will serve as starting points:

Gries, David, *Compiler Construction for Digital Computers*, John Wiley & Sons, Inc., New York, 1971
(See chapters 3, 4, 12, and 15)

Aho, Alfred V., and Jeffrey D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972
(See chapters 1, 3, and 5)

Georgia Institute of Technology School of Information and Computer Science, *GTL Programmer's Reference Manual for the Burroughs B5500*, 1974
(See chapter 8)

Principles of Operation

'Stacc' generates a "parser," a program which converts a stream of "tokens" into a representation of a derivation tree which describes the production of the input stream from a given grammar. In practice, the derivation tree exists solely in the call structure of the subprograms called to parse the input, so the user must supply "action" routines to produce the output he desires.

A parser written with 'stacc' also requires a "lexical analyzer," a routine that converts the input stream of ASCII characters into the tokens handled by the parser.

The operation of a 'stacc' parser is roughly as follows. In the initialization phase, the lexical analyzer is called to pick up the first token from the stream of input characters. An integer assigned to the class of token found is then placed in a "current symbol" variable by the lexical analyzer. The parser is then called. The parser attempts to "match" the current symbol against all possibilities for the first input symbol; if a match is found, any actions supplied by the user are performed and the lexical analyzer is called to fetch the next input token. This procedure is repeated until either the entire input stream is recognized as a valid sentence in the input language or an error (a missing or illegal "current symbol") is detected. In the event of an error, the user must supply recovery actions so that the parse can proceed.

The function of 'stacc' is to convert an extended BNF grammar into code that checks the current input symbol, calls the lexical analyzer when appropriate, and performs actions specified by the user after certain constructs in the input stream are recognized, thus freeing the user from the bookkeeping details needed to build a parser.

Usage Information

'Stacc' takes input (described in detail below) on its first standard input port and produces output on its first and second standard outputs. The first output is the code that implements the parser. This code is expressed in the language whose name is given as the first argument on the command line that invoked 'stacc'. The second output is a set of macro definitions that establishes mnemonics for the integer "current symbol" values supplied by the lexical analyzer.

Conventionally, 'stacc' input files have the extension ".stacc", e.g. "hp.stacc," "sspl.stacc," "stacc.stacc". The first output of 'stacc' (the parser) is normally placed in a file with extension ".stacc.<language>", e.g. "hp.stacc.r", "sspl.stacc.s", "stacc.stacc.r". The second output of 'stacc' (the macro definitions) is normally placed in a file with extension ".stacc.defs", e.g. "hp.stacc.defs", "sspl.stacc.defs", "stacc.stacc.defs". These files may then be "included" in a source file during compilation. (Note that slightly different naming conventions are used by the separate compilation handler 'sep').

Input Specifications

Input to 'stacc' consists of a series of "declarations" and "productions", separated by semicolons. There may be any number of either, and they may be mixed in any order. Input is free-form; whitespace may be inserted where desired to

improve readability. Ratfor-style comments (beginning with a sharp (#), ending with a NEWLINE) may be used for documentation.

Declarations consist of a period (.) followed by a keyword and an argument list whose format depends on the keyword. There are seven types of declarations.

Four declarations are used to select the names of critical objects used by the parser: ".state <variable>" declares the parser state variable (named "state" by default), ".scanner <routine>" declares the name of the lexical analyzer subprogram ("getsym" by default), ".symbol <variable>" declares the "current symbol" variable (named "symbol" by default), and ".epsilon <symbol>" declares the symbol to be used to match the null token (empty string of input symbols).

One declaration is used only by parsers written in Ratfor: ".common '<filename>'" specifies the name of an include file containing the declarations of the current symbol variable and any other variables used for communication between the parser and the lexical analyzer.

The final two types of declarations are used to list mnemonics for terminal symbols recognized by the lexical analyzer. The first consists of ".ext_term" followed by a list of identifiers used by the lexical analyzer to identify terminal symbols. This declaration merely informs 'stacc' that the given names represent terminal symbols; no macro definitions are generated. The second type consists of ".terminal" followed by a list of mnemonics. Each mnemonic is assigned an integer value, and output in a macro definition to make that value available to the lexical analyzer. A specific value may be assigned to a terminal symbol by preceding it with an integer; two terminal symbols may be equated by placing an equals sign (=) between them. Otherwise, increasing values (starting from zero) are assigned. For example, the following declaration

```
.terminal
        ALTSYM
        DECLSYM = NOADVANCESYM
        100 LETTER_D
        LETTER_E
        ;
```

produces the following Ratfor macro definitions:

```
define(ALTSYM,0)
define(DECLSYM,1)
define(NOADVANCESYM,1)
define(LETTER_D,100)
define(LETTER_E,101)
```

Productions are written in a language similar to the exten-

ded BNF used throughout the Subsystem. A production consists of a nonterminal symbol, followed by the "rewrites as" symbol (->), followed by a right-hand-side with imbedded semantic actions.

The right-hand-side allows the usual BNF operators: vertical bar (|) to indicate a choice, parentheses to nest right-hand-sides, square brackets ([]) to enclose optional constructs, and curly braces ({}) to enclose repeated constructs.

Items in the right-hand-side are nonterminal symbols (those that appear in the left-hand-side of some production), terminal symbols (those declared by the ".terminal" and ".ext_term" declarations), quoted single characters, or two terminal symbols or quoted characters separated by a colon (:) (which matches any terminal symbol or character within the given limits). In addition, a right-hand-side may be preceded by a dollar sign ($), indicating that it represents a particularly common form of production: a number of alternatives, each of which is distinguished by a single leading terminal symbol. (This happens, for example, in parsing statements in most algorithmic languages; each different type of statement is preceded by a unique key word.) Recognition of this common case allows much faster special-purpose code to be generated.

After any terminal, nonterminal, or special construct in the right-hand-side there may appear semantic actions. Actions to be performed after a symbol is successfully matched are preceded by an exclamation point (!); actions to be performed after a symbol fails to be matched are preceded by a question mark (?). Actions extend from their initial character to the end of the line on which they appear. Actions appearing after terminal symbols are executed *after* a symbol is matched and *before* the lexical analyzer is called; thus, they may perform some operation based on characteristics of the symbol matched. If the terminal symbol or range of terminal symbols being matched is followed by a period (.), the automatic call of the lexical analyzer is disabled, allowing the user to substitute his own scanning actions.

Actions may appear immediately after the "rewrites as" symbol (->), in which case they are executed before any code generated by 'stacc', or immediately after the production-terminating semicolon (;), in which case they are executed unconditionally before control leaves the production.

A sample production:

```
parser ->
   {
      (     declaration
         |  production
      )
```

```
                ';'     ? call error ("missing semicolon.")
                        ! numprods = numprods + 1
              }
            EOF.        ? call error ("EOF expected.")
                        ! call analyze
            ;
```

Using 'Stacc' With Ratfor

Ratfor users of 'stacc' should note that they must declare a common block "include" file with the ".common" declaration, so that the lexical analyzer can communicate with the parser.

The form of a 'stacc' output routine in Ratfor is a subroutine with one integer argument, which exports the parser state upon return. The parser state will either be NOMATCH (1) if the first symbol failed to match any legal alternative, FAILURE (2) if some symbols matched but some did not (and no error recovery succeeded), or ACCEPT (3) if the input was a legal sentence in the language being processed. The name of the status argument is presently fixed at "gpst"; this variable is reserved for 'stacc' and should not be used for any other purpose.

To use a 'stacc'-generated parser, the Ratfor programmer should simply call the subroutine whose name corresponds to the start symbol of his grammar, passing one integer variable as an argument. That variable will contain the parse state upon completion of the parse.

If the user specifies a grammar that is recursive, 'stacc' will produce recursive output; it will then be necessary to use Virtual-mode Fortran with the local-variables-allocated-in-stack-frame option. (This is the default under the Subsystem.)

Examples

The following sample 'stacc' input will generate a program to convert infix arithmetic expressions to reverse-Polish. An expression consists of letters, digits, and operators arranged in the usual manner. Multiplication and division have priority over addition and subtraction.

```
.scanner "getchar";
.symbol "char";
.common "rpn.com";

expression ->
                    ! integer op
   term
   {
      ($    '+'    ! op = '+'c
```

```
                  |  '-'    ! op = '-'c
                  )
               term         ! call putch (op, STDOUT)
               }
         ;

      term ->
                            ! integer op
         factor
         {
            ($    '*'      ! op = '*'c
               |  '/'      ! op = '/'c
               )
            factor          ! call putch (op, STDOUT)
            }
         ;

      factor ->
            'a':'z'         ! call putch (char, STDOUT)
         |  '0':'9'         ! call putch (char, STDOUT)
         |  '('
               expression
               ')'
         ;
```

'Stacc' produced the following Ratfor output:

```
      define(NOMATCH,1)
      define(FAILURE,2)
      define(ACCEPT,3)



      subroutine expression (gpst)
      integer gpst
      include 'rpn.com'
      integer state
      integer op
      call term (state)
      select (state)
         when (FAILURE) {
            gpst = FAILURE
            return
            }
      if (state == ACCEPT) {
         repeat {
            state = NOMATCH
            select (char)
            when (171) {
               state = ACCEPT
               op = '+'c
               call getchar
               }
            when (173) {
```

```
                    state = ACCEPT
                    op = '-'c
                    call getchar
                    }
                 if (state == ACCEPT) {
                    call term (state)
                    select (state)
                       when (FAILURE) {
                          gpst = FAILURE
                          return
                          }
                       when (ACCEPT) {
                          call putch (op, STDOUT)
                          }
                    if (state ~= ACCEPT) {
                       gpst = FAILURE
                       return
                       }
                    }
                 } until (state ~= ACCEPT)
              select (state)
                 when (NOMATCH)
                    state = ACCEPT
              if (state ~= ACCEPT) {
                 gpst = FAILURE
                 return
                 }
              }
           gpst = state
           return
           end



           subroutine term (gpst)
           integer gpst
           include 'rpn.com'
           integer state
           integer op
           call factor (state)
           select (state)
              when (FAILURE) {
                 gpst = FAILURE
                 return
                 }
           if (state == ACCEPT) {
              repeat {
                 state = NOMATCH
                 select (char)
                 when (170) {
                    state = ACCEPT
                    op = '*'c
                    call getchar
                    }
                 when (175) {
                    state = ACCEPT
```

```
                op = '/'c
                call getchar
                }
             if (state == ACCEPT) {
                call factor (state)
                select (state)
                   when (FAILURE) {
                      gpst = FAILURE
                      return
                      }
                   when (ACCEPT) {
                      call putch (op, STDOUT)
                      }
                if (state ~= ACCEPT) {
                   gpst = FAILURE
                   return
                   }
                }
             } until (state ~= ACCEPT)
          select (state)
             when (NOMATCH)
                state = ACCEPT
          if (state ~= ACCEPT) {
             gpst = FAILURE
             return
             }
          }
       gpst = state
       return
       end



       subroutine factor (gpst)
       integer gpst
       include 'rpn.com'
       integer state
       state = NOMATCH
       if (225 <= char && char <= 250) {
          state = ACCEPT
          call putch (char, STDOUT)
          call getchar
          }
       if (state == NOMATCH) {
          if (176 <= char && char <= 185) {
             state = ACCEPT
             call putch (char, STDOUT)
             call getchar
             }
          if (state == NOMATCH) {
             if (char == 168) {
                state = ACCEPT
                call getchar
                }
             if (state == ACCEPT) {
                call expression (state)
```

```
                    select (state)
                       when (FAILURE) {
                          gpst = FAILURE
                          return
                          }
                    if (state ~= ACCEPT) {
                       gpst = FAILURE
                       return
                       }
                    state = NOMATCH
                    if (char == 169) {
                       state = ACCEPT
                       call getchar
                       }
                    if (state ~= ACCEPT) {
                       gpst = FAILURE
                       return
                       }
                    }
                 }
              }
           gpst = state
           return
           end
```

The following main program and common block include file were necessary to finish the implementation:

```
     # rpn --- convert to Reverse Polish

        include "rpn.stacc.defs"

        integer state

        include "rpn.com"

        call getchar
        call expression (state)
        call putch (NEWLINE, STDOUT)
        if (state ~= ACCEPT || char ~= NEWLINE)
           call error ("syntax error.")

        stop
        end

     # getchar --- get next character from standard input

        subroutine getchar

        include "rpn.com"

        character getch

        char = getch (char, STDIN)
```

```
            return
            end

         include "rpn.stacc.r"



         # common blocks for 'rpn'

            character char
            common /chcom/ char
```

Messages

(Note that all error messages are preceded by the number of the line in the input stream being processed at the time of the detection of the error.)

```
-> symbol is ill-formed
   '-' was seen, but '>' was missing
EOF expected
   there is data after the last legal production
actions are illegal here
   actions are not allowed immediately after '$'
bad symbol
   input string could not be lexically analyzed
error actions illegal here
   error actions not allowed after quick-select terminal
identifier or string expected
   missing declaration parameter
illegal declarator
   keyword after '.' was not recognizable
illegal term/nonterm
   expected a terminal or nonterminal; didn't find one
missing '->'
   should be a '->' after the left-hand-side of a production
missing alternative
   missing or illegal alternative after '|'
missing choice
   missing or illegal quick-select alternative after '|'
missing declarator
   missing keyword after '.'
missing optional rhs
   there should be a right-hand-side within square brackets
missing quote
   obvious, hopefully
missing quote or string too long
   strings have a maximum length of about 100 characters
missing repeated rhs
   there should be a right-hand-side within curly braces
missing rhs in parentheses
   there should be a right-hand-side within parentheses
missing right brace
```

missing right bracket
missing right parenthesis
missing right-hand-side
   missing entire right-hand-side of production
missing semicolon
missing upper bound
   missing second terminal in range of form 'lower:upper'
not yet available
   the language specified cannot be used with 'stacc' yet
production expected
   input stream contained neither declaration nor production
too many action/erroraction lines
   there are too many lines of code to store internally
too many characters pushed back
   internal error --- see your Subsystem manager
too much action/erraction text
   there is too much code to store internally
unsupported language
   'stacc' doesn't recognize the language name specified

(Other messages may occasionally arise from the dynamic storage routines. If these occur, see your Subsystem manager for assistance.)

Bugs

'Stacc' does not optimize its Ratfor output as well as it could; some redundant code shows up occasionally.

Error recovery in 'stacc' is still somewhat primitive.

No check is made to see if the input grammar is LL(1).

See Also

rp (1)

Usage

```
stats  [ -{option} ]

option ::= t | a | m | s | v | h | l | r | q | n | %<rank>
```

Description

'Stats' is a filter that can be used to generate various statistical measures of a set of floating point data. Input to 'stats' is a list of numbers, appearing one per line but free-form within each line, on its first standard input. Output from 'stats' is a list of statistics, preceded by labels (unless the "-q" option has been specified) on the first standard output.

The options control the statistics to be printed. The available options are:

-t	Print the sum (total) of all data values.
-a	Print the arithmetic mean (average) of the data values.
-m	Print the mode (most frequently occurring value).
-s	Print the standard deviation of the population sampled.
-v	Print the variance of the population sampled.
-h	Print the highest value in the sample.
-l	Print the lowest value in the sample.
-r	Print the range of values in the sample (highest - lowest).
-q	Quiet; turn off the printing of labels on the output.
-n	Print the number of data values in the sample.
%	Print percentile ranks for the data. The percent sign (%) must be followed by the percentile increment to be used for the ranking. Note that "-%50" yields the median value for the sample.

The default options are currently "-as%50".

Examples

```
grades> stats
grades> stats -ahl%25
{ ([files .r])> tc -l } | stats -tahl
lf -cw | field 1-8 | stats -tq
```

Messages

"Usage: stats ..." for improper options.

Bugs

The mode and percentile rank statistics are limited to relatively small data sets because of an internal sort.

Usage

stop [- | <pathname>]

Description

'Stop' is used to exit from the Software Tools Subsystem. If no option is specified, the user's profile is saved and a normal exit to Primos occurs. If the "-" option is specified, the user's profile is saved and logout from the Prime is forced. If a pathname is given, then the given file is deleted before logout is forced and the user's profile is not saved. This is used in conjunction with the 'ph' command to log out phantoms.

Examples

```
stop
stop -
stop =varsdir=/ph00301
```

See Also

bye (1), ph (1), Primos logo$$

Usage

st_profile [<count_file>] <source_code_file>

Description

'St_profile' is used to convert the profiling information generated by a Ratfor program processed with the "-c" option into a readable report. The optional <count_file> argument is the name of a statement_level profile data file generated by a profiled program; if omitted, the default name of "_st_profile" is assumed. The <source_code_file> argument is the name of the file containing the Ratfor source code for the program being profiled.

Examples

```
st_profile rp.r
st_profile guide_profile fmt.r
```

Files

_st_profile is the default <count_file>.

Messages

"Usage: st_profile ..." if no arguments given.
"can't open" if files are inaccessible.

Bugs

See Reference Manual entry for 'rp'.

Seems to leave out the last line of source code.

See Also

rp (1), profile (1), c$init (6), c$incr (6), c$end (6)

Usage

subscribe

Description

'Subscribe' allows a user to become a subscriber to the Subsystem news service.

There is no difference in the news received by subscribers and non-subscribers; the only difference is that subscribers are automatically informed whenever a news item is published.

Examples

subscribe

Files

=news=/subscribers for news service subscribers

See Also

news (1), publish (1), retract (1)

Usage

substr <start> <length> <string>

Description

'Substr' is similar in function to the PL/I substr function; it prints on standard output a specified substring of its third argument. The substring printed is taken from <string> starting at position <start> and continuing for <length> characters, or until the end of <string> is reached. If <start> is negative, the starting position is -<start> characters from the end of <string>. If <length> is negative, characters are extracted from right to left.

'Substr' is perhaps excessively general; for common problems, the 'take' and 'drop' commands will usually suffice.

Examples

```
substr 1 2 11/27/84
substr [start] [len] [full_name]
set last_five = [substr -5 5 [variable]]
```

See Also

take (1), drop (1), rot (1), substr (2), stake (2), sdrop (2)

Usage

systat

Description

'Systat' is a shell program which contains commands to list the contents of the gossip directory (names of users with pending messages from the 'to' command), and the mail directory (names of users with undelivered mail).

Examples

systat

Files

=gossip= for messages from 'to'
=mail= for messages from 'mail'

See Also

mail (1), to (1), lf (1)

Usage

tail [<number of lines>] [<file>]

Description

'Tail' is a filter that prints on standard output the last few lines that it reads from standard input. The number of lines printed may be specified as an integer argument; the default value is twenty if none is given. Currently, the maximum number of lines that can be printed is 300. If a number larger than this is specified, a value of 300 is used and no error message is issued.

If <number of lines> is preceded by a minus sign, 'tail' discards the first <number of lines> lines from its input file and copies the remainder to standard output.

If a file name is given as the second argument, 'tail' takes its input from the named file instead of standard input.

Examples

```
log_file> tail 10
tail 10 log_file
lf -cw | sort | tail 5
listfile> tail -1
tail -1 listfile
```

Bugs

For the single argument case, if argument is the string "0", the program will read in the default number of lines from file "0". If the single argument is a file name that starts with digits, those digits will be interpreted as the number of lines to be read from standard input.

For the two argument case, if the first argument is the string "0", the second argument is ignored and a file name of "0" is assumed.

See Also

slice (1)

Usage

take <length> <string>

Description

'Take' can be used to extract substrings from the beginning or end of a string. It is essentially identical in function to APL's dyadic take operator as applied to character vectors.

The absolute value of the first argument specifies the number of characters to be taken (with blank padding, if the source string is not long enough.) If it is positive, characters are taken from the beginning of <string>; otherwise, characters are taken from the end of <string>.

Other useful string-handling commands are 'drop', 'index', and 'substr'.

Examples

```
take 6 [filename]
take -2 [source_file]
take 2 [date]
take 3 [day]
```

See Also

drop (1), index (1), substr (1), stake (2), sdrop (2)

Usage

```
tc [-{c | l | p | w | v}] [<pathname>]
```

Description

'Tc' is a filter used to count various parameters of text, specifically characters, words, lines, and pages.

If a pathname is specified, the text to be counted is read from that file; otherwise, text is read from standard input 1. All output from 'tc' is written to its first standard output. If no options are specified, or the "v" option is specified, the counts are labeled; otherwise only the counts themselves appear.

The options control the items counted: "c" for characters, "w" for words, "l" for lines, "p" for pages. The "v" option causes unconditional printing of labels on the output.

Examples

```
report> tc
eval [tc -w part1] + [tc -w part 2]
lf -c | tc -l
```

Messages

"Usage: tc ..." for argument specified without "-" flag
"illegal option" for unrecognized option letter

Bugs

The page length is fixed at 66 lines, which is incorrect for output generated by 'print'. In addition, the definition of word used for word counting ("a sequence of non-blank, non-tab, non-newline characters") may be too simplistic.

Usage

```
tee { <pathname> | -[1 | 2 | 3] }
```

Description

'Tee' creates multiple copies of data flowing into its first standard input. By default, it copies this stream of data to its first standard output. In addition, a copy is made on each of the files named in its argument list. If a named file did not previously exist, it is created.

If an argument consists only of a dash ("-"), optionally followed by a single digit in the range 1-3, a copy is sent to the standard output port corresponding to the digit. If the digit is missing, standard output one is assumed.

'Tee' is suitable for checkpointing data flowing past a given point in a pipeline, or for fanning out a data stream to feed multiple, parallel pipelines.

Examples

```
lf -c | tee file_names | print -p -n >/dev/lps
memo> tee [cat distribution_list]

file_names>  tee -2  |P1  |P2 _
   :P1  change % //dir1/  |  cat -n  |$ _
   :P2  change % //dir2/  |  cat -n  |$ _
        lam
```

Messages

"<pathname>: can't create" if a file cannot be created.

Bugs

This function could be performed by the i/o primitives.

See Also

cat (1)

Usage

template [-a | -r] [-l{usv}] { <string> }

Description

'Template' allows the user to expand templates, list the contents of the system template file or his own private template file, or edit the contents of his private template file. The operation of 'template' is controlled by command line options as follows:

-a add or change templates. Any <string>s that appear on the command line are taken in pairs of the form

<name> <definition>

If any template in the user's template file has a name corresponding to <name>, its definition is replaced by <definition>; otherwise, the new template is added to the file.

-r remove templates. Each <string> on the command line is taken as the name of a template to be removed from the user's template file.

-l list templates. The contents of either the system template file or the user's private template file, or both, are listed on standard output. The "-l" may be followed by one or more of the following options to control the listing:

u list the contents of the user's private template file.

s list the contents of the system template file.

v print a label before listing each set of templates.

If neither the "s" nor the "u" option is specified, "u" is assumed by default.

Note that the "-l" option may be used with either the "-a" or the "-r" option to list the modified templates.

In the absence of any of the above options, 'template' expands each <string> argument and prints the result on standard output. In order to allow arbitrary strings containing templates to be expanded, it is necessary to enclose the template name in "equals" symbols (=) just as it would appear in a pathname. This is the *only* context in which 'template' requires or allows the name of a template to be so enclosed.

Examples

```
template =date= =time= =doc=/man
template -al mybin //mydir/bin.ufd
template -r oldtemp
template -lusv
```

Files

=utemplate= for storage of personal templates

Messages

"Usage: template ..." for improper options
"<string>: duplicate name" if two or more templates with the same name are specified with "-a"
"<string>: missing definition" if a template name is not followed by a definition string with "-a"
"<string>: may not contain '=' " if an attempt is made to add a template name containing an equals sign
"file not altered" if either of the previous two messages is issued
"<string>: not in template file" if an attempt is made to delete a non-existent template
"can't open user template file" if "=utemplate=" can't be opened for reading and writing
"can't open temporary file" if a temporary file can't be created for the "-a" and "-r" options

See Also

expand (2), lutemp (6)
For more information on templates, see *User's Guide to the Primos File System* in the *Software Tools Subsystem User's Guide*.

Usage

```
term { ? | <type> | <option> }
   <option> ::=  -erase <echar>      |  -kill <kchar>
                 |  -retype <rchar>    |  -escape <escchar>
                 |  -newline <nlchar>  |  -eof <eofchar>
                 |  -[no]break         |  -[no]echo
                 |  -[no]lcase         |  -[no]lf
                 |  -[no]xon           |  -[no]xoff
                 |  -[no]vth           |  -[no]se
                 |  -[no]inh
```

Description

'Term' sets or displays the parameters that control terminal input and output. At present, these parameters are:

- erase character
- kill character
- retype character
- escape character
- end-of-file character
- newline character
- recognition of terminal interrupts
- full/half duplex selection
- inhibition of terminal output
- automatic mapping of lower case input
- line feed suppression
- interpretation of DC1 and DC3 control characters
- support by the screen editor 'se'
- support by the virtual terminal handler

Arguments to 'term' consist of either a terminal type, in which case parameters applicable to that particular terminal are set, or of keywords that set specific parameters to specific values. A list of available terminal types will be displayed if the "?" option is requested.

Keywords that may be specified, and their respective meanings, are as follows:

-erase set erase character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new erase (character delete) character.

-kill set kill character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new kill (line delete) character.

-retype set retype character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new retype (repeat line) character.

-escape set escape character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new escape character. (The escape character is used to enter special character codes that could not otherwise be entered from a standard keyboard.)

-newline set newline character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new newline character. The new character must then be used to terminate all subsequent input lines.

-eof set end-of-file character. The next argument must be a single character or an ASCII mnemonic for the character which is to become the new end-of-file character.

-break enable terminal interrupts. When terminal interrupts are enabled, hitting the BREAK key or control-p has the effect of halting the currently executing program.

-nobreak disable terminal interrupts. When terminal interrupts are disabled, the currently executing program may not be interrupted by the BREAK key or control-p. If, however, either of these keys is hit, a flag is set indicating a pending interrupt that will take effect as soon as terminal interrupts are re-enabled.

-echo full duplex, each character typed is echoed by the computer. When this mode is selected, the terminal should be set to **full duplex** mode to disable self echo.

-noecho half duplex, characters are not echoed by the computer; they must rather be echoed by the terminal if they are to be printed. When this mode is in effect, the terminal should be set to **half duplex** mode to enable self echo.

-inh inhibit output. The effect is the same as if a DC3 character had been received while "-xon" mode was enabled.

-noinh enable output. The effect is the same as if a DC1 character had been received while "-xon" mode was enabled. Any buffered output is sent to the terminal.

-lcase lower case. This option specifies that the user's terminal can send and receive lower case characters.

-nolcase upper case only. This option is intended for use

with upper-case-only terminals such as Teletypes. All input is forced to lower case and all output is forced to upper case. Upper case letters may be entered by preceding the letter with the escape character (nominally "@"). On output, upper case letters are printed as an escape character followed by the letter.

-lf echo line feed when carriage return is received. The computer echoes a line feed whenever it receives a carriage return. This is independent of whether or not echo is on. However, if echo is on, the line feed is echoed after the carriage return.

-nolf do not echo line feed when carriage return is received. The terminal should have an automatic line feed feature for this mode to produce desirable results.

-xon the computer recognizes the control characters DC1 and DC3 (Control-Q and Control-S) as X-ON and X-OFF signals, respectively. When X-OFF is received, output is inhibited until X-ON is received. Characters output by a program when output is inhibited are not lost, but are buffered until an X-ON signal is next received. The options "-xon" and "-xoff" are synonymous, as are "-noxon" and "-noxoff".

-noxon the computer does not recognize X-ON and X-OFF signals.

-se the terminal is supported by the screen editor. User modification of this option is allowed for completeness. Setting it does not necessarily mean that 'se' will operate correctly with the terminal.

-nose the terminal is not supported by the screen editor. User modification of this option is allowed for completeness.

-vth the terminal is supported by the virtual terminal handler. User modification of this option is allowed for completeness. Setting it does not necessarily mean that the 'vth' routines will operate correctly with the terminal.

-novth the terminal is not supported by the virtual terminal handler. User modification of this option is allowed for completeness.

If no arguments are specified, term prints the values of the various terminal parameters on standard output one.

Examples

```
term tty
term -lcase -noecho -nolf
term
```

Messages

"Usage: term ..." for incorrect arguments.

See Also

ek (1), Primos duplx$, gttype (2), gtattr (2) *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

term_type [-[no]se | -[no]vth | -[no]lcase]

Description

'Term_type' prints a mnemonic for the type of the current user's terminal on standard output when it is called with no arguments. The mnemonic is suitable for use with 'se', among other things.

If one of the other options is given, 'term_type' prints a "1" or "0" to indicate whether or not the option is selected for the terminal. For instance, "term_type -se" prints "1" if the terminal is supported by 'se'.

If no terminal type has been specified for the user's terminal, the call to 'gtattr' or 'gttype' in 'term_type' will request the terminal type from the user. Otherwise, 'term_type' will use the remembered terminal type.

Examples

```
echo "Your terminal type: "[term_type]
if [term_type -se]
   se [args]
else
   ed [args]
fi
```

Files

=termlist= for the terminal list.
=ttypes= for the legal terminal type list.

Messages

"Usage: term_type ..." for illegal arguments.

"No terminal type information is available". For some reason no terminal type is configured for the line and the user has refused to supply a terminal type.

See Also

line (1), term (1), se (1), gtattr (2), gttype (2)

Usage

```
if [ <value> ]
   then
      { <command> }
   else
      { <command> }
fi
```

Description

'Then' is a do-nothing command that may be used to introduce the "affirmative" or "asserted" part of a conditional statement. It is available solely for the purpose of improving the appearance of conditional statements in command files, and is always optional.

Examples

```
if [nargs]
   then
      set file = [arg 1 | quote]
fi
```

See Also

if (1), else (1), fi (1), case (1)

Usage

time

Description

'Time' prints the time of day in the format hh:mm:ss on standard output one.

Examples

```
echo Run at [time]
```

See Also

date (1), day (1), ctime (1), date (2)

Usage

```
tip
```

Description

'Tip' checks to see if there is any terminal input pending. If terminal input is waiting to be read, 'tip' outputs a "1". If no terminal input is waiting to be read, 'tip' outputs a "0".

'Tip' is most commonly used with the 'if' command.

Examples

```
if [tip]
   [set =]
fi
```

Bugs

Should probably be able to check an assigned terminal line for pending input.

See Also

if (1), chkinp (2)

Usage

tlit <from set> [<to set> { <string> }]

Description

'Tlit' is the character transliteration program from *Software Tools*. Character strings are read from the command line arguments, or from standard input, transliterated according to instructions provided in the command line arguments, and the results written to standard output. The <from set> and <to set> arguments are sets of characters, with some special shorthand notation. Each set may have any number of the following components:

<character>
The character specified becomes part of the set.

<letter>-<letter>
The letters specified, and all letters between them alphabetically, become part of the set. (Note that letters of a given case are contiguous; A-Z means all upper case letters.)

<digit>-<digit>
The digits specified, and all digits between them in numerical order, become part of the set.

@n,@t
A NEWLINE (if the first form is used) or a TAB (if the second form is used) becomes part of the set.

In addition, if the <from set> is preceded by a tilde (~), the complement of the set is used. For example, "~A-Z" means all characters except upper case letters.

For each character read that is a member of the <from set>, the corresponding member of the <to set> is substituted. If the <to set> is shorter than the <from set>, each string of contiguous characters that are in the <from set> but have no corresponding element in the <to set> is replaced by a single occurrence of the last member of the <to set>. If the <to set> is empty or only a single argument is supplied, such character strings are deleted.

When strings are read from the argument list, each separate argument is treated as a NEWLINE-terminated string. Thus, lacking specific transliteration of NEWLINE characters, each separate argument string will result in one line of output.

Examples

```
file> tlit a-z A-Z >uc_file
file> tlit A-Z a-z | tlit ~a-z @n >words
tlit a-z A-Z "output one line"
tlit a-z A-Z output three lines
```

Messages

"Usage: tlit ..." if no arguments are supplied.
"<from> set too large" if <from set> cannot be contained in the internal buffer.
"<to> set too large" if <to set> cannot be contained in the internal buffer.

See Also

change (1), ed (1), *Software Tools*

Usage

to (<login-name> | <user-number>) [<message>]

Description

'To' is used to send messages to another logged-in user. The first argument is the login name or user number of the user to whom the message is to be sent. If any other arguments are present, they are assumed to be message text and are sent to the named user. If the login name, or user number, is the only argument specified, 'to' copies the text of the message from standard input.

The message, preceded by the sender's name and user number and the current day and time, will appear on the named user's terminal when he is next prompted for a command by the Subsystem command interpreter.

Examples

```
to jack There seems to be a problem with se on the tvt...
to 16 Why is the system so slow?
message> to perry
```

Files

=gossip=/<login-name> to hold the message
=gossip=/*<user-number> to hold the message

Messages

"Usage: to ..." if no arguments are specified.
"bad user number" if the user number is not in the range 1..128.
"bad user name" if the user name is not that of a valid user.
"User is busy. Try later." if message file is in use.

Bugs

Gossip messages are neither secure nor private.

See Also

mail (1)

Usage

touch [-d <date>] [-t <time>] {<pathname>}

Description

Every file system object has a field indicating the date and time (to within 4 seconds) it was last modified. This command will set the date/time modified field of file system objects.

The "-d" option may be used to specify a date as recognized by the 'parsdt' routine; if no date is specified then the current date is used. The "-t" option may be used to specify a time as recognized by the 'parstm' routine; if no time is specified then the current time is used.

The remaining command line arguments are taken as names of files for which to set the modification time. If "-n" appears in place of a pathname, pathnames are read from the standard input. For more information on this syntax, see the entry for 'cat' (1).

Examples

```
lf -c | touch -n
touch -t 1124 foo.r bar.r
```

Messages

"Usage: touch ..." for invalid arguments.
"invalid format in date argument" or "invalid format in time argument" for improper arguments.
"<pathname>: can't "touch"" for protected or non-existant files.

See Also

cat (1), lf (1), gfdata (2), parsdt (2), parstm (2), sfdata (2)

Usage

```
tsort [-v]
```

Description

'Tsort' reads pairs of names from its standard input, sorts them topologically, and writes the result to its standard output. Such a function is useful, for example, in ordering the subroutines in a library so that it may be loaded in a single pass.

Each input line consists of a pair of names, separated by blanks. The first name is taken as the predecessor, and the second as the successor. The names are printed, one per line, such that each name that appeared as a successor on input follows all the names that appeared as its predecessor. Normally, only names that appeared somewhere as a predecessor are printed. However, if the "-v" flag is specified, all names are printed.

Examples

```
brefs library.b | tsort >lib_order
precedences> tsort -v
```

Messages

"Usage: tsort ..." for specifying unknown flag arguments.
"input data error" if an input line contains only one name.
"cycle (in reverse order): name ..." when a set of mutually recursive references is detected.

See Also

bmerge (5), bnames (5), brefs (5)

Usage

ucc { <input_files> } [<cc_opts>] [<compile_opts>]

Description

'Ucc' is a "UNIX-style" C compiler and loader. It does NOT, however, behave like Unix's 'cc' or any other known Unix program!

'Ucc' compiles and loads the pathnames specified. It assumes that all programs require C runtime support and loads them accordingly.

'Ucc' recognizes the same set of options as 'cc'. These are passed as C-specific options to 'compile'. Any other options are passed on to 'compile' directly, which does the real work of recognizing suffixes and compiling and loading the files appropriately.

'Ucc' remains in existence mainly for compatibility with the first release of the C compiler.

Examples

```
ucc sort.c
ucc sort.c -ud  # the -ud is automatically passed on to 'ld'
ucc main.c lib.r lib.s -R-t
```

Bugs

Has basically become obsolete.

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

compile (1), cc (1), ccl (1), vcg (1), ld (1), bind (3), *User's Guide for the Georgia Tech C Compiler*

Usage

uniq [-n]

Description

'Uniq' is used to strip adjacent duplicate lines from its standard input. The resulting text is copied to standard output. 'Uniq' is usually used to eliminate redundant lines from a sorted file.

If the "-n" option is specified, 'uniq' counts the number of occurrences of each line. The count is placed right justified in the first five columns of the output, suitable for sorting or further manipulation with 'change', 'find', or 'field'.

Examples

words> sort | uniq -n

Messages

"Usage: uniq ..." for invalid argument syntax.

Bugs

Does not handle lines of length greater than MAXLINE.

See Also

sort (1), speling (1)

Usage

unrot [-w <width>]

Description

'Unrot' processes the "rotated" output of 'kwic' to generate a key-word-in-context index. It reads lines from standard input one and writes the index on standard output one.

The length of the output lines may be specified with the "-w <width>" argument sequence. The maximum width is currently 137 characters. If no width is specified, 65 is assumed.

Examples

definitions> kwic | sort | unrot >index

Messages

"Usage: unrot ..." for invalid argument syntax.

See Also

kwic (1), sort (1), uniq (1)

Usage

```
repeat
   { <command> }
until [ <value> ]
```

Description

'Until' marks the end of a 'repeat' loop. It actually does nothing and is just searched for by the 'repeat' statement when it is in the process of pre-processing a loop. Each 'repeat' command must be followed by a matching 'until' command.

Examples

```
repeat
   echo This terminal is taken
until                              # infinite loop

repeat
   hd swt
   lf
until [eval [template =date=] == 110284]
```

See Also

if (1), then (1), else (1), fi (1), case (1), repeat (1)

Usage

us

Description

'Us' is a shell file that invokes the Primos STATUS command to print a listing of the users currently logged in, along with their line number and any devices they have assigned. Output is always to the user's terminal.

Examples

us

See Also

who (3)

Usage

usage { <command> }

Description

'Usage' prints a summary of the acceptable command line syntax for each command and a summary of the acceptable calling sequence for each library subprogram named in its argument list. Command line syntax is expressed in a BNF-like meta-language that is described by "help -g bnf".

Examples

usage rf ed se

Files

=doc=/fman/s1/<command>.d for command documentation
=doc=/fman/s2/<subprogram>.d for subprogram documentation
=doc=/fman/s3/<command>.d for local command documentation
=doc=/fman/s4/<subprogram>.d for local subprogram doc.
=doc=/fman/s5/<command>.d for low-level command documentation
=doc=/fman/s6/<subprogram>.d for low-level subprogram doc.
=doc=/fman/index for command and subroutine index

Messages

"Sorry, no help is available for <command>" in case of missing or unreadable documentation file.

See Also

help (1), *Software Tools Subsystem Reference Manual*

Usage

vars [-{v | c | g | a | l}] [-r [<file>] | -s [<file>]]

Description

'Vars' can be used to print the names and values of all currently defined shell variables, save the variables in a file, or restore them from a file. The options have meanings as follows:

-v Values. Print the value of each variable as well as its name.

-c Columnar. Print information in a single column, instead of across the page (similar to the -c option of 'lf').

-g Global. Print names of all global variables, as well as those on the current nesting level.

-a All. Print names of all shell variables, on any nesting level, including those beginning with "_" (normally reserved for use by the shell).

-l Long. Select options a, g, and v.

-s Save. Save shell variables. If a file name is specified, variables and their values are saved in the given file; otherwise, the file "=varsfile=" is used. Only level 1 (global) variables are saved. Listing of variable names and values still occurs.

-r Restore. Restore shell variables. Variables and values from the named file (default "=varsfile=") are merged with the currently active set of variables, at the current nesting level. Listing of variable names and values still occurs.

If no options are specified, 'vars' lists the names of all variables active at the current nesting level in a multi-column format.

For more information on shell variables, see the 'set', 'declare' and 'forget' commands, or the *User's Guide for the Software Tools Subsystem Command Interpreter*.

Examples

```
vars
vars -vc
vars -l
vars -s
```

```
vars -s =varsdir=/environment
vars -r =varsdir=/environment
```

Bugs

Should print the mnemonic form of the variable names.

See Also

declare (1), forget (1), set (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

```
vcg [-m <module>] [-b [<path>]] [-s [<path>]]
```

Description

'Vcg' is a reusable, general purpose code generator which accepts a linearized intermediate form tree and generates either 64V-mode object text or PMA or both. For a complete description of 'vcg' see the *VCG User's Guide*.

'Vcg' requires three input files; the first contains entry points, the second contains external definitions and the third contains the intermediate form tree.

If given, the "-m" argument specifies the name of the <module> to which 'vcg' will append the suffixes ".ct1", ".ct2", and ".ct3" for the three input files. Otherwise, 'vcg' expects the three input files to be on its three standard input ports.

The following command line options are available:

-b Generate 'ld'-compatible object text directly. 'Vcg' generates object text by default and places it in the file "<path>". If <path> is not specified, but a <module> name is given with the "-m" argument, 'vcg' will place the object text in the file "<module>.b". Otherwise, 'vcg' will print an error message and exit.

-s Generate assembly code. 'Vcg' will produce Prime Macro Assembly Language (PMA) and place it in the file "<path>". Object text generation is suppressed unless the "-b" command line flag is also specified. If <path> is not specified, but a <module> name is given with the "-m" argument, 'vcg' will place the PMA in the file "<module>.s". Otherwise, 'vcg' will simply write the PMA on its first standard output port.

-m Specify input and output module names. This option was discussed above.

In general, the user should not invoke this command directly. Rather, 'vcg' should be called via one of the compiler interludes, like 'cc'.

Examples

```
vcg -m temp # use temp.ct(1 2 3)
p.ent> p.ext> p.tree> vcg -s  # write PMA to stdout
```

Messages

Numerous, but sometimes opaque.

Bugs

'Vcg' expects correctly formed input. When it is presented with something else, it usually manages to complain, but it may either die or blithely emit incorrect code. The major problem in dealing with 'vcg' is that it is often not easy to tell what part of the input is causing the difficulty.

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

cc (1), ccl (1), ucc (1), vcgdump (1), *A Re-Usable Code Generator for Prime 50-Series Computers User's Guide*, *User's Guide for the Georgia Tech C Compiler*

Usage

```
vcgdump [ <path prefix> ]
```

Description

'Vcgdump' displays an input file intended for 'vcg' in a semi-readable format. For a complete description of 'vcg' see the *A Re-Usable Code Generator for Prime 50-Series Computers User's Guide*.

'Vcgdump' requires three input files; the first contains entry points, the second contains external definitions and the third contains the intermediate form tree. If no argument is given, these files are read from standard inputs 1, 2 and 3 respectively. Otherwise, 'vcgdump' will append the suffixes ".ct1", ".ct2", and ".ct3" to <path prefix> and use these names for the input files.

The output of 'vcgdump' is placed on standard output.

Examples

```
vcgdump temp | pg    # use temp.ct(1 2 3)
p.ent> p.ext> p.tree> vcgdump | pr
```

Bugs

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

cc (1), ccl (1), ucc (1), vcg (1), *A Re-Usable Code Generator for Prime 50-Series Computers User's Guide*

Usage

vpsd <program> { <arguments> }

Description

'Vpsd' allows the user to invoke the Primos V-mode Symbolic Debugger (VPSD) on a Subsystem program. The program must have been linked by 'ld' with the "-d" option (i.e., it must be a segment directory). The standard ports are assigned and the arguments are set-up, and then a call to the Primos loader (SEG) with the "vpsd" option is made, via 'sys$$'.

Examples

```
vpsd bogus_program >bonzo
echo Test data | vpsd non_debugged_program -d -l -t 2>answers
```

Bugs

There is no protection in either the shell or 'vpsd' for user errors (such as changing incorrect memory locations) while in VPSD.

'Vpsd' is not much good for debugging programs written in anything other than assembly. For programs written in a higher level language, use 'dbg'.

The single character I/O of VPSD is not the duke's choice.

See Also

dbg (1), sys$$ (2), *Prime Assembly Language Programmer's Guide* (Chapters 18 and 21 on VPSD), *Prime Load and Seg Reference Guide*

Usage

```
case <value>
   when <alternative1>
      { <command> }
   when <alternative2>
      { <command> }
   ...
   out
      { <command> }
esac
```

Description

'When' is used by the 'case' command to flag alternatives in a multi-way comparison. The argument of 'when' is tested by 'case', and if it is found to be equivalent to <value>, then the group of statements following 'when' is executed. In this function, 'when' is similar to the case statement in the language C.

'When' itself is executed only when control falls out of the series of commands controlled by the previous 'when'. The action taken in this case is to skip until the next unmatched 'esac' is seen. In this respect, 'when' and 'out' are identical.

Like 'out' and 'else', if executed from a terminal without proper termination by an 'esac', 'when' will cause the shell to skip input until end-of-file is seen.

Examples

```
case [line]
   when 10
      se -t adds [arg 1]
   when 15
      se -t b200 [arg 1]
   out
      ed [arg 1]
esac
```

Messages

"Missing 'esac'" if end-of-file is seen before an 'esac' command.

Bugs

Redirectors before the 'esac' prevent 'when' from spotting it.

The string given as the argument to 'when' is not evaluated; therefore, function calls and iteration groups are not allowed.

See Also

case (1), out (1), esac (1), if (1), *User's Guide for the Software Tools Subsystem Command Interpreter*

Usage

whereis [- | <terminal number>]

Description

'Whereis' is a shell program that may be used to find the location of a specific terminal. The argument may be the line number of the terminal to be located or a single dash (meaning "all terminals").

Terminal numbers appear under the column heading "LINE" in the output produced by the Subsystem's 'us' command (Primos STATUS USERS command).

Examples

whereis 10
whereis -

Files

=termlist= for terminal list

Bugs

Dependence on the fixed terminal list means that inaccuracies will occur as terminals get changed or moved around.

See Also

whois (1), us (1), term_type (1)

Usage

```
which <command>
```

Description

'Which' steps along the path indicated in the shell variable '_search_rule' to locate the <command> named as its argument. It knows which commands are internal to the shell.

Examples

```
which cd
which (fmt se)
```

Messages

"Usage: which ..." for improper arguments.

Bugs

Ignores all arguments but the first.

See Also

sh (1), svget (2)

Usage

whois (- | { <login-name> })

Description

'Whois' is used to determine the name of the user associated with a particular login name. The most frequent usage is "whois <login_name>", which looks up the login name specified and prints the name of its owner. If the "-" option is specified, 'whois' prints the entire name list. If no argument is given, 'whois' accepts a list of login names from standard input and prints the name of the owner of each.

Examples

```
whois -
whois - | pg
whois allen perry dan
```

Files

=userlist= for name list

Bugs

Has a grubby output format due to the incredibly long user login names.

See Also

us (1), vfyusr (2), who (3)

Usage

x [-d <directory-name>] [<Primos command>]

Description

'X' allows users to execute Primos commands without leaving the Subsystem. The command and its arguments may be specified as arguments to 'x', or, if no command is so specified, 'x' reads commands from its standard input. If arguments are present, 'x' forms a Primos command by concatenating all arguments (other than the "-d <directory-name>" pair) into a single Primos command line and passes it to the Primos command interpreter with a call to the Primos routine CP$. If the Primos command returns with a positive return code, 'x' exits with a call to 'error'; otherwise, 'x' exits normally. No change is made to the Primos command input source.

If 'x' reads commands from standard input, it reads the first line, connects the Primos command source to the standard input file (either disk or terminal) and passes the line to the Primos command interpreter through CP$. When the command returns, 'x' resets the Primos command input source. Then, if the Primos command returned with a non-positive return code, 'x' continues to read commands from standard input until end-of-file; otherwise 'x' terminates with a call to 'error'.

All Primos file units that are left open by a Primos command will be automatically closed by the Subsystem when the 'x' command returns. Please note that this means the sequence

```
x "l mylist; ftn myprog"
```

will correctly deposit the listing file in "mylist", but

```
x l mylist
x ftn myprog
```

will deposit the listing in "l_myprog", since 'x' will close the listing file when it returns.

If the "-d" option is specified, 'x' will attach to the named directory without changing the home directory. This gives the user the ability to execute Primos commands on any point in the file system from any point in the file system.

Examples

```
x -d //system share se2031 2031 700
x pma prog.p 1/707
x "r system>sw4000 1/1"
```

Messages

"usage..." for missing directory name after "-d".
"<directory-name>: bad pathname" for bad directory.

See Also

fc (1), ld (1), primos (1), stop (1)

Usage

xcc <pathname> [-c] [-b[<b_pathname>]] [-l[<l_pathname>]]

Description

'Xcc' compiles the C program in <pathname> with Prime's C compiler. The ".c" suffix on the source file name is optional, although 'xcc' requires that the source code reside in a file named with a ".c" suffix. If the source file name specified in <pathname> does not have a ".c" suffix, 'xcc' will append a ".c" and attempt to process a file with that name. The object code is stored in "<pathname>.b". If the "-b" command-line argument specifies <b_pathname>, 'xcc' stores the object code in a file with that name.

A full description of the C language is beyond the scope of this document. For complete information, refer to *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie (Prentice-Hall, 1978).

The following options are available:

- -b Compile the source code into the object file named "<b_pathname>". 'Xcc' places the object code into the file "<pathname>.b" if this option or <b_pathname> is unspecified.

- -c Invoke the "-CHECKOUT" option. This option causes the compiler to parse the source code without producing object code. This option suppresses the "-b" and "-l" options.

- -l Produce a listing in the file "<l_pathname>.l". If <l_pathname> is unspecified, 'xcc' places the listing in the file named "<pathname>.l".

Examples

```
xcc file.c
xcc prog.c -l prog_list -b bonzo.b
xcc test.c -c -l
```

Messages

Numerous and self-explanatory.

Bugs

There is no way to tell Prime C programs about the Subsytem standard input/output ports.

Does not give full access to the all the options available with Prime's C compiler.

See Also

ld (1), xccl (1), bind (3)

Usage

xccl <program name> [<'ld' options>] [/ <'xcc' options>]

Description

'Xccl' is a shell file that invokes the Primos C compiler and the Primos segmented loader. If 'xccl' is invoked with no <program name> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. The name of the file containing the program to be compiled must end with ".c", although in <program name> it may be specified with or without the ending ".c". If no output file is specified in the <'ld' options>, the output object file name will be <program name> with no extension.

Concerning the options, 'xcc' will be called with the <'xcc' options> specified on the command line; then 'ld' will be called with the <'ld' options> specified.

Examples

```
xccl myprog.c
xccl myprog subs.b subs2.b -l mylib
xccl myprog / -l mylist
```

Messages

"<program name>.c: cannot open"

See Also

xcc (1), ld (1), bind (3)

Usage

```
xref { -{b | i | l | p | u} } { <pathname> }
```

Description

'Xref' produces a cross referenced listing of a Ratfor program. The listing consists of the program text, with consecutively numbered lines, and an alphabetical concordance of the identifiers that occur in the program text.

The program text is read from the concatenation of all the files specified as <pathname> arguments. If there are none, standard input is used.

Several command line options are available to control the operation of 'xref':

-b highlight keywords by boldfacing. Any Ratfor or Fortran keywords contained in the program text will be boldfaced (by overstriking) in the listing. If the listing is to be printed on a line printer, the output should be filtered through 'os' to convert the overstrikes into multiple lines with Fortran forms control.

-i process 'include' statements. Any 'include' statements encountered in the program text will be expanded inline. The included lines will be numbered consecutively as if they appeared in the primary input file.

-l include literals in the concordance. Any numeric literals that appear in the program text will be included with the identifiers in the concordance.

-p format listing for printing. A formfeed will be generated following the listing of each input file. Output under this option is suitable for piping into 'pr'.

-u highlight keywords by underlining. Any Ratfor or Fortran keywords contained in the program text will be underlined (by overstriking) in the listing. If the listing is to be printed on a line printer, the output should be filtered through 'os' to convert the overstrikes into multiple lines with Fortran forms control.

Examples

```
rfprog.r> xref
xref -pu prog1 prog2 prog3 | print | os >/dev/lps/f
```

See Also

os (1), pr (1), rp (1)

Usage

yesno [-yes | -no]

Description

This program can be used as a filter in a pipe to regulate input to other commands. It reads a line at a time from standard input (STDIN), echoes the line in quotes followed by a question mark, and then prompts the user for a yes or no answer. A response of "y", "ye", "yes", or "ok" all result in the line being passed to standard output (STDOUT). A response of "n" or "no" discards the line. Any other response causes an error message and the user is prompted with the line again. Case is not significant in responses.

Use of the optional "-yes" argument causes a carriage return (null response) to default to a "yes" response. Use of the optional "-no" argument causes a null response to default to "no."

If the user types an end-of-file character (normally a control C) as the response to any decision prompt then the current line and all subsequent lines in the input are discarded.

All display and prompting are done to and from device TTY and thus will not show up in any of the standard inputs or outputs should they be redirected or piped. As a result, this command cannot be used in a phantom job, nor may a set of pre-determined answers be constructed as input to the program.

Examples

```
lf -ca | yesno -no | del -n
lf -c /user/mfd | yesno | del -sd -n
=dictionary=> yesno -yes >my_dictionary
```

Messages

"Usage: yesno ..." for improper command line syntax.

"answer YES or NO." for incorrect or unknown response.

Bugs

Does not recognize "-y" or "-n" as command line arguments.

Will not work in batch or phantom jobs.

Section 2 - Library Subprograms

A complete set of library subroutines is necessary for effective program development under the Subsystem. The primitive operations suggested by Kernighan and Plauger in *Software Tools*, as well as many local functions, have been compiled and placed in the libraries =lib=/vswtlb, =lib=/vswtmath, and =lib=/vshlib (for V-mode programs).

This section is designed to give the user a working knowledge of these functions and subroutines. Each routine has its own entry organized under the following headings. Note that empty entries are omitted entirely.

Header Line

The subprogram's name, a synopsis of its purpose, and the date of last modification to its documentation.

Calling Information

The subprogram declaration and the declarations of its arguments, as well as the name of the library in which it can be found. This should be used as a reference when constructing calls to a given routine.

Function

A description of the purpose of the routine, along with the interpretations of its arguments and the returned value (if any).

Implementation

A short discussion of the strategy used to implement the routine, abstracted from the source code.

Arguments Modified

Names of those arguments modified by the routine.

Calls

Other subprograms called by this routine.

Bugs

Known problems with the use of the routine.

See Also

References to further information or related routines.

Calling Information

```
longreal function acos$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse cosine of an angle. The argument to the function is the cosine of the angle, and the function returns the measure of the angle, in radians. Arguments to the function must be in the closed interval [-1.0, 1.0]. In the case of an error, the default return value is zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is implemented as a rational minimax approximation on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dsqt$m, Primos signl$

See Also

cos$m (2), dacs$m (2), dsqt$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
integer function addset (c, set, j, maxsiz)
character c, set (maxsiz)
integer j, maxsiz
```

Library: vswtlb (standard Subsystem library)

Function

'Addset' puts the character 'c' in the array 'set' at position 'j' and increments 'j', provided that 'j' is not greater than 'maxsiz'. The function return is YES if 'c' was inserted, NO otherwise.

Implementation

Trivial.

Arguments Modified

set, j

Calling Information

```
integer function amatch (lin, from, pat, tagbeg, tagend)
character lin (ARB), pat (MAXPAT)
integer from, tagbeg (9), tagend (9)
```

Library: vswtlb (standard Subsystem library)

Function

'Amatch' checks the substring of the line 'lin' starting at position 'from' to see if it matches the pattern in 'pat'. 'Pat' must have been created by the utility routine 'makpat' beforehand. If a match occurs, then the arrays 'tagbeg' and 'tagend' are used to record the beginning and end of any tagged subpatterns that appeared in 'pat' (i.e., 'tagbeg(N+1)' contains the index in 'lin' of the start of the Nth tagged subpattern, 'tagend(N+1)' contains the index in 'lin' of the end of the Nth subpattern, while 'tagbeg(1)' and 'tagend(1)' bracket the entire matched string). The function return is the index of the next unexamined character in 'lin' if a match was found, zero otherwise.

Implementation

'Amatch' steps through successive entries in the pattern, attempting to match them against the line of text. Most of the complexity arises in handling closures; 'amatch' calls 'omatch' repeatedly to match the longest possible substring, then backs up as necessary to make the remainder of the pattern match. This may involve multiple backups, since there may be more than one closure in a pattern.

'Omatch' is called to match all single non-closure pattern elements. If 'omatch' fails, then the stack of pending closures is examined. If empty, 'amatch' returns zero; if non-empty, 'amatch' reduces the last closure and attempts to match again.

Whenever a pattern tag (open brace or close brace) is encountered in the pattern, 'amatch' records the current offset in the line in 'tagbeg' or 'tagend', whichever is appropriate.

Arguments Modified

tagbeg, tagend

Calls

omatch, patsiz

Bugs

Rather slow.

See Also

match (2), makpat (2), omatch (2), find (1), ed (1), se (1)

Calling Information

```
longreal function asin$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse sine of an angle. The argument to the function is the sine of the angle, and the function returns the measure of the angle, in radians. Arguments to the function must be in the closed interval [-1.0, 1.0]. If an error is signalled, the default function value is zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is implemented as a rational minimax approximation on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dsqt$m, Primos signl$

See Also

dasn$m (2), dsqt$m (2), err$m (2), sin$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function atan$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse tangent of an angle. The argument to the function is the tangent of the angle, and the function returns the measure of the angle, in radians. The function will not signal any errors based on input values.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is implemented as a rational approximation on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

See Also

datn$m (2), err$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function atoc (ptr, str, size)
integer ptr (3), size
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Atoc' converts the 2 or 3 word 64V mode indirect pointer in the address 'ptr' to a printable EOS-terminated string in 'str'. No more than 'size' elements of 'str' will be modified, including the trailing EOS.

The pointer is converted into the format

[f]<ring>..<word>[.<bit>]

<Ring>, , and <word> are positive octal integers. The character "f" is present only if the fault bit in the pointer is set, and <bit> is included only if the extension bit is set.

The function return is the number characters used to represent the address (the length of 'str').

Implementation

Bits are removed from the indirect pointer and converted to character representation with calls to 'gitoc' in a straightforward manner.

Arguments Modified

str

Calls

gitoc, ctoc

See Also

atoc (2), other conversion routines (?*toc (2), cto?* (2))

Calling Information

```
subroutine cant (file_name)
character file_name (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Cant' is a Kernighan/Plauger subroutine normally used to report errors after an attempt to open a file. The 'file_name' supplied (which must be an EOS-terminated string) is printed on ERROUT, followed by the message "can't open", and an immediate return to the shell is taken.

Implementation

'Cant' calls 'putlin' to print the filename supplied, and 'error' to print the "can't open" message and return to the Subsystem command interpreter.

Calls

putlin, error

See Also

open (2), create (2), remark (2)

Calling Information

```
subroutine catsub (lin, from, to, sub, new, k, maxnew)
character lin (MAXLINE), new (maxnew), sub (MAXPAT)
integer from(10), to(10), k, maxnew
```

Library: vswtlb (standard Subsystem library)

Function

'Catsub' adds replacement text onto a string after a pattern match and substitution operation. 'Lin' is the original text string matched by 'amatch'. 'From' and 'to' are ten-entry arrays specifying the beginning and end of all tagged subpatterns; the N'th element refers to the N-1th tagged pattern, and element 1 refers to the entire string matched. 'Sub' is the substitution pattern created by 'maksub'. 'New' is the string to receive the replacement text; its maximum length is 'maxnew' and the index at which the replacement text is to be inserted is 'k'.

Implementation

The substitution string is copied into 'new' starting at 'k'. Whenever a DITTO ("&" or "@<digit>") is encountered, a portion of the original text string is also copied.

Arguments Modified

new, k

Calls

addset

See Also

maksub (2), makpat (2), change (1), ed (1), se (1)

Calling Information

```
integer function chkarg (arg_num, result)
integer arg_num, result (26)
```

Library: vswtlb (standard Subsystem library)

Function

'Chkarg' scans the list of arguments supplied on the command line, starting at position 'arg_num', looking for arguments that contain a dash followed by a string of letters. For each letter in such an argument, 'chkarg' looks at the corresponding element in the 'result' array (the letters "A" and "a" correspond to element 1, "Z" and "z" to element 26). If the element is non-negative, it is set to a positive value equal to the order in which the letter was encountered in scanning the arguments, counting from 1. Otherwise, the element is left unchanged and a value of ERR is returned as the result of the function. Thus, illegal letters may be detected by setting the corresponding elements in 'result' to a negative value before calling 'chkarg'.

Scanning continues, incrementing 'arg_num', until the end of the argument list is reached, an argument not beginning with a dash is found, or an argument beginning with a dash but containing a subsequent character other than a letter is found. In the first two cases, 'chkarg' returns with the number of letters encountered as its result. In the third case, a result of ERR is returned.

Implementation

'Chkarg' does a straightforward argument scan, using 'getarg' to fetch each argument in turn. The actions taken for each argument are simply those mentioned above.

Arguments Modified

arg_num, result

Calls

getarg

See Also

getarg (2), getkwd (2)

Calling Information

```
logical function chkinp (flag)
logical flag
```

Library: vswtlb (standard Subsystem library)

Function

'Chkinp' returns the value ".true." if there are characters waiting to be read in the user's terminal buffer. Otherwise, 'chkinp' returns ".false.".

Implementation

'Chkinp' enters 64R addressing mode and executes the instruction

```
SKS '704
```

(which is trapped and interpreted by Primos). If the instruction skips, 'chkinp' reenters 64V mode and returns ".true.". Otherwise, it reenters 64V mode and returns ".false.".

Arguments Modified

flag

Calling Information

```
integer function chkstr (str, len)
character str (ARB)
integer len
```

Library: vswtlb (standard Subsystem library)

Function

'Chkstr' looks to see if the characters in a string are all printable. If an EOS character is encountered before any unprintable characters are encountered and before 'len' characters are examined, 'chkstr' returns YES; otherwise, it returns NO.

If 'len' is less than or equal to zero, 'chkstr' returns NO.

Implementation

'Chkstr' starts examining the string at the first character; as long as the character is not an EOS and is printable and 'len' characters have not been examined, 'chkstr' continues to examine the remainder of the string. When an unprintable or EOS character is found, or when 'len' characters has been examined, 'chkstr' quits; it returns YES if it has encountered an EOS character, and NO otherwise.

See Also

ctomn (2), mntoc (2)

Calling Information

```
integer function close (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Close' closes the file associated with the given file descriptor (the value returned by a call to 'open', 'create', or 'mktemp') and releases its buffer areas. If the file was open for writing, any data still buffered is written to the file. After a file is closed, its file descriptor becomes available for future use. 'Close' returns OK if the attempt to close was successful, ERR otherwise.

If an attempt is made to close a standard port (STDIN, STDOUT, etc.) 'close' will return OK, but it will *not* close the file associated with the port.

Implementation

'Close' first checks to see if the given file descriptor is a standard port descriptor. If so, the attempt to close is ignored. If the file descriptor is illegal or corresponds to an already closed file, ERR is returned. 'Flush$' is then called to force any pending writes on the file to be performed. The Primos routine SRCH$$ is used to close disk files; other file types are closed simply by updating Subsystem status areas.

Calls

flush, Primos srch$$

Bugs

Some consider the behavior on standard ports unreasonable, but it definitely seems useful.

See Also

open (2), create (2), mktemp (2), flush$ (6)

Calling Information

```
longreal function cos$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function returns the cosine of the angle whose measure (in radians) is given by the argument. The absolute value of the angle plus one-half pi must be less than than 26353588.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return is zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is implemented as a minimax polynomial approximation. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

acos$m (2), dcos$m (2), dint$p (2), err$m (2), sin$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function cosh$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This routine calculates the hyperbolic cosine of its argument, defined as cosh(x) = [exp(x) + exp(-x)]/2. Arguments which produce a value too large for single precision storage will signal the error condition. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function value is zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The algorithm was adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m, Primos signl$

See Also

dcsh$m (2), dexp$m (2), err$m (2), sinh$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function cot$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the cotangent of the angle whose measure is given (in radians) as the argument to the function. The argument must have an absolute value greater than 7.064835966E-9865 and less than 13176794.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return is zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is calculated based on a minimax polynomial approximation over a reduced argument. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

dcot$m (2), dint$p (2), err$m (2), tan$m (2),
SWT Math Library User's Guide

Calling Information

```
file_des function create (file_name, mode)
character file_name (ARB)
integer mode
```

Library: vswtlb (standard Subsystem library)

Function

'Create' creates a named file. The parameter 'mode' may be any one of READ, WRITE, or READWRITE, and specifies the action(s) that may be performed on the newly-created file. If the file name specified already exists, it is opened and then truncated to zero length. 'Create' returns a file descriptor if it was successful, ERR otherwise. The file created will have default protection keys of "a/" (owner has all permissions, non-owners have none).

By default, 'create' returns a file descriptor to a sequential access method (SAM) file when referring to a disk file. If creating a direct access method file (DAM) is desired, the 'mode' argument may be ORed with the KNDAM file key (i.e., 'mode' can be "READWRITE+KNDAM" to create a DAM file opened for reading or writing). The constant KNDAM is contained in the "PRIMOS_KEYS" include file.

Implementation

'Create' calls 'open' to open the named file, then calls 'trunc' to set it to zero length. If an error occurs during truncation, the file is closed by calling 'close'. Note that truncation will not be performed if 'mode' is READ; but then, who would create a new file for reading only, anyway?

Calls

close, open, trunc

See Also

open (2), close (2), mktemp (2), rmtemp (2)

Calling Information

```
long_int function ctoa (str, i)
character str (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Ctoa' converts the address in ASCII character representation at position 'i' of the given string to binary format. 'I' is incremented to point to the position just after the integer. If the character at position 'i' is not numeric when 'ctoa' is entered, the value zero is returned (the exceptions are blanks and tabs; these characters are ignored at the start of the number). 'Ctoa' recognizes a 32-bit address in the following format:

[f]<ring>..<word>

The presence of the character "f" at the beginning of the address indicates that the pointer fault bit is to be set. <Ring>, , and <word> are positive octal integers. A bit number following the address is ignored, if present.

Implementation

'Ctoa' scans the string, using the argument 'i' as the starting position. Leading blanks and tabs are skipped. The octal integers are collected with 'gctol'. As each element of the address is collected, it is placed in the proper bit positions of the long integer return value.

Arguments Modified

i

Calls

gctol

Bugs

Cannot return 48 bit indirect pointers.

See Also

atoc (2), other conversion routines (?*toc (2) and cto?* (2))

Calling Information

```
integer function ctoc (from, to, len)
integer len
character from (ARB), to (len)
```

Library: vswtlb (standard Subsystem library)

Function

'Ctoc' copies an EOS-terminated unpacked string from one array to another, observing a maximum-length constraint on the destination array. The function return is the number of characters copied (i.e., the length of the string in the parameter 'to').

Note that the other string copy routine, 'scopy', is not protected; if the length of the source string exceeds the space available in the destination string, some portion of memory will be garbled.

Implementation

A simple loop copies characters from 'from' to 'to' until an EOS is encountered or all the space available in the destination array is used up.

Arguments Modified

to

See Also

scopy (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
long_real function ctod (str, i)
character str (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Ctod' converts the character string in the array 'str', starting at position 'i', to double precision floating point representation and returns this value as the result of the function. The variable 'i' is incremented to a point one character beyond the string that was converted; the array 'str' is not modified. 'Str' must be an EOS-terminated unpacked character string.

'Ctod' recognizes any valid Fortran constant; in particular, leading signs are handled. Leading blanks and tabs are ignored.

Implementation

'Ctod' accumulates the integer and fractional parts of the number, throwing away leading zeros and insignificant digits and computing scaling factors if necessary. A straightforward Horner's method conversion translates each portion of the constant to binary, and finally all portions are combined and appropriately scaled. Scaling is aided by using tables of powers-of-two exponents, to preserve as much accuracy as possible.

Arguments Modified

i

Calls

gctoi

See Also

dtoc (2), ctor (2), rtoc (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function ctoi (str, i)
character str (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Ctoi' converts the integer in ASCII character representation at position 'i' of the given string to binary format. 'I' is incremented to point to the position just after the integer. If the character at position 'i' is not numeric when 'ctoi' is entered, the value zero is returned (the exceptions are blanks and tabs; these characters are ignored at the start of the number). 'Ctoi' does not recognize a leading plus or minus sign.

Implementation

'Ctoi' scans the string, using the argument 'i' as the starting position. Leading blanks and tabs are skipped. If a numeric is encountered, it is added to ten times the current value of the integer, and the scan continues; otherwise, 'ctoi' exits with the desired value.

Arguments Modified

i

See Also

itoc (2), gitoc (2), gctoi (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
long_int function ctol (str, i)
character str (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Ctol' converts the integer in ASCII character representation at position 'i' of the given string to binary format. 'I' is incremented to point to the position just after the integer. If the character at position 'i' is not numeric when 'ctol' is entered, the value zero is returned (the exceptions are blanks and tabs; these characters are ignored at the start of the number). 'Ctol' does not recognize a leading plus or minus sign.

Implementation

'Ctol' scans the string, using the argument 'i' as the starting position. Leading blanks and tabs are skipped. If a numeric is encountered, it is added to ten times the current value of the integer, and the scan continues; otherwise, 'ctol' exits with the desired value.

Arguments Modified

i

See Also

ltoc (2), gltoc (2), gctol (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function ctomn (c, rep)
character c, rep (4)
```

Library: vswtlb (standard Subsystem library)

Function

'Ctomn' is used to convert an unprintable ASCII character to its official ASCII mnemonic. The first argument is the character to be converted; the second is a string to receive the mnemonic. The function return is the length of the string placed in the second argument.

If the character passed is printable, it is copied through unchanged to the receiving string. If not, its two- or three-character ASCII mnemonic (e.g. NUL, SOH, etc.) is copied into the receiving string.

Implementation

If the character is printable, it is placed in the receiving string, which is then terminated with EOS. If the character is between 128 and 160, inclusive, or equals 255, its value is used to compute an index into a string table containing the mnemonics. The mnemonic thus selected is copied into the receiving string.

Arguments Modified

rep

Calls

scopy

See Also

mntoc (2)

Calling Information

```
integer function ctop (str, i, pstr, len)
character str (ARB)
integer i, len
packed_char pstr (len)
```

Library: vswtlb (standard Subsystem library)

Function

'Ctop' converts the EOS-terminated unpacked string in argument 'str', starting at position 'i', to packed integer form in the array 'pstr'. The argument 'len' gives the maximum length of the array 'pstr'; no more than 'len' words of this array will be modified by 'ctop'. After conversion, 'i' points to the EOS at the end of 'str', or one position past the last character packed if the maximum length of 'pstr' is exceeded.

The function return is the number of characters transferred from 'str' to 'pstr'.

Implementation

'Ctop' picks up successive characters from 'str' and packs them into 'pstr' with the standard Subsystem macro 'spchar'.

Arguments Modified

i, pstr

See Also

ptoc (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
real function ctor (str, i)
character str (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Ctor' is similar in function to 'ctoi', except that it converts floating point numbers as well as integers. The character string in 'str' is examined starting in position 'i'. Conversion stops when a character is encountered that cannot correctly appear in the number. 'I' is updated to point to the first character not included in the converted number. The value returned by the function is the real (single precision) value of the character string.

The number in 'str' may contain a leading sign, a decimal point, and an exponent. A decimal point is not required.

Implementation

'Ctod' is called to convert the character string into a double precision value. This value is converted to single precision format and returned as the value of 'ctor'.

Arguments Modified

i

Calls

ctod

See Also

input (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function ctov (str, i, var, len)
character str (ARB)
integer i, len
packed_char var (len)
```

Library: vswtlb (standard Subsystem library)

Function

'Ctov' converts *Software Tools* style EOS-terminated strings to PL/I style "character varying" strings. Character varying strings consist of a one-word length field, followed by up to 32767 words of packed character data.

The argument 'str' contains the EOS-terminated string to be converted. The integer 'i' gives the position of the first character in the string to be converted, i.e. the starting point of the substring to be packed. 'Var' is the array which is to receive the character varying string, and 'len' is the number of words in 'var' available for holding characters plus one (for the string length word). Conversion starts at the 'i'th position in 'str' and continues until an EOS is encountered in 'str' or 'var' is completely filled. The function return is the number of characters packed.

Implementation

'Ctov', like 'ctop', makes repeated calls on the standard macro 'spchar' to pack characters into the destination array. Once all characters in the string have been packed, or no room remains in the destination, 'ctov' sets the first word of the destination array to the number of characters it contains and returns this number as the function value.

Arguments Modified

i, var

See Also

other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
longreal function dacs$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse cosine of an angle. The argument to the function is the cosine of the angle, and the function returns the measure of the angle, in radians. Arguments to the function must be in the closed interval [-1.0, 1.0]. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. In the case of an error, the default return value is zero.

Implementation

The function is implemented as a rational minimax approximation on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

acos$m (2), dcos$m (2), dsqt$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function dasn$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse sine of an angle. The argument to the function is the sine of the angle, and the function returns the measure of the angle, in radians. Arguments to the function must be in the closed interval [-1.0, 1.0]. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function value is zero.

Implementation

The function is implemented as a rational minimax approximations on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dsqt$m, Primos signl$

See Also

asin$m (2), dsin$m (2), dsqt$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
subroutine date (item, str)
integer item
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Date' is used to return several interesting pieces of data that Primos keeps for the user. The first argument is a switch to select the data returned; the second is a string for receiving the data. The following values of the first argument are defined:

SYS_DATE	1	date, in format mm/dd/yy
SYS_TIME	2	time, in format hh:mm:ss
SYS_USERID	3	user's login name
SYS_PIDSTR	4	user's three digit process id
SYS_DAY	5	day of the week (e.g. "monday", "tuesday", etc.)
SYS_PID	6	process id as a binary integer in str (1)
SYS_LDATE	7	name of day, name of month, day, year
SYS_MINUTES	8	number of minutes past midnight in str (1..2)
SYS_SECONDS	9	number of seconds past midnight in str (1..2)
SYS_MSEC	10	number of milliseconds past midnight in str (1..2)

If the first argument is not one of these values, an empty string is returned.

Implementation

'Date' calls the Primos routine TIMDAT to fetch time, date, process id, and login name information. This information is then reformatted as needed.

Arguments Modified

str

Calls

Primos timdat, encode (2), mapup (2), ptoc (2), wkday (2)

Calling Information

```
longreal function datn$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the inverse tangent of an angle. The argument to the function is the tangent of the angle, and the function returns the measure of the angle, in radians. The function will not signal any errors based on input values.

Implementation

The function is implemented as a rational approximation on a modified argument value. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

See Also

atan$m (2), err$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dble$m (l)
longint l
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'dble$m' function implements something akin to the Fortran 66 'dble' function, or the Fortran 77 'dreal' function. It takes as an argument a 32 bit integer and returns a double precision floating point number of the same value. This function should always be used when converting 32 bit integers to double precision real numbers because the code generated by some of the compilers will (potentially) lose up to 8 bits of mantissa precision.

Implementation

The algorithm involved was derived from known register structure; see the source code for specifics.

See Also

SWT Math Library User's Guide

Calling Information

```
longreal function dcos$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function returns the cosine of the angle whose measure (in radians) is given by the argument. The the absolute value of the angle plus one-half pi must be less than 26353588.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return is zero.

Implementation

The function is implemented as minimax polynomial approximation. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

cos$m (2), dacs$m (2), dint$p (2), dsin$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function dcot$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the cotangent of the angle whose measure is given (in radians) as the argument to the function. The argument must have an absolute value greater than 7.064835966E-9865 and less than 13176794.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return is zero.

Implementation

The function is calculated based on a minimax polynomial approximation over a reduced argument. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

cot$m (2), dint$p (2), dtan$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function dcsh$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This routine calculates the hyperbolic cosine of its argument, defined as cosh(x) = [exp(x) + exp(-x)]/2. The absolute value of the argument must be less than 22623.630826296. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function value is zero.

Implementation

Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m, Primos signl$

See Also

cosh$m (2), dexp$m (2), dsnh$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
integer function decode (str, sp, fmt, fp, ap, a1, ..., a10)
character str (ARB), fmt (ARB)
integer sp, fp, ap
untyped a1, ..., a10
```

Library: vswtlb (standard Subsystem library)

Function

'Decode' is used to convert a character string to a number of items in various internal formats (e.g. integer, double precision floating point, address, etc.). Its function is similar to the Fortran statement of the same name.

The argument 'str' is the character string to be decoded, and 'sp' indicates the position in 'str' at which decoding is to begin. 'Fmt' is a string of format control directives (discussed below), and 'fp' indicates the position in 'fmt' of the first format control directive to be used for decoding. 'A1' through 'a10' (at most) are variables to receive decoded data; 'a2' through 'a10' are optional, and any or all may be omitted. 'Ap' indicates the next variable in the list of 'a1' through 'a10' to receive decoded data.

'Decode' performs the decoding operation until it either runs out of string to decode or of format to control the decoding. The arguments 'sp', 'fp', and 'ap' are always updated to point to the next unused character in 'str', the next unused character in 'fmt', and the next variable in the variable list, respectively.

The function return is OK if not all of the format string was used, EOF if all of the format string was used, or ERR if an input string was in error.

The format string consists of a series of "format control directives." Each directive controls the conversion of a segment of the character string into some internal form. A directive consists of the format flag character (an asterisk "*") followed by up to three comma-separated option fields, and a single character format specifier. The option fields are normally designated the "width", "base", and "delimiter character" fields. The width field controls the maximum number of characters in the input string to be converted. The base field controls the radix representation assumed for integer fields (and a few other miscellaneous options, discussed below). The delimiter character field specifies a character that may be used to terminate the conversion process for a single variable if it is encountered in the string.

The following format specifiers are available:

a

The input string must contain an address of the form "<ring_number>.<segment_number>.<offset>". The receiving variable must be a two-word address pointer.

b or y

The input string must contain a boolean constant, which may be 1 or 0, TRUE or FALSE, T or F, YES or NO, Y or N. The receiving variable must be of type integer or type logical.

d or f

The input string must contain a standard Fortran representation of a double-precision floating-point constant. The receiving variable must be of type long_real or double_precision.

g

None of the input string is examined by this format code. The argument pointer 'ap' is set to the value of the width field; this allows input items to be refilled or skipped entirely.

h

The input string must contain at least as many characters as are specified by the width field. The given number of characters are then packed into the receiving variable, which must be an array of integers larger than the number of characters divided by two (since there are two characters per word on the Prime.) The base field, if nonzero, specifies a limit on the number of words of the receiving array that will be changed; thus, if the width field is not specified, the entire input string (possibly terminated by the delimiter character) will be packed into the receiving array, but the array will be protected from overrun by the specification of its size in the base field. The code 'h' comes from the Fortran term "hollerith literal," which is the type of the receiving variable.

i

The input string must contain a representation of a short (16-bit) integer constant. If the base field is non-zero, it is assumed to be the radix used for representation of the integer. If zero, base 10 is assumed. The base specified in the format directive may be overridden in the input string by giving a radix followed by the letter "r" followed by the desired value, e.g. "2r1001" or "16rA000". The receiving variable must be of type integer.

l

The input string must contain a representation of a long (32-bit) integer constant. The syntax and semantics of this form are identical to form 'i' above, with the exception that the receiving variable must be of type long_int (integer*4).

n

The width field specifies the number of newlines in the input string to be skipped. If the end of the input string is encountered, the skipping stops. This code is most often used by the 'input' routine.

p

The syntax and semantics of this form are identical to the 'h' form above, with the exception that a period character (".") will be placed at the end of the receiving array so that its length may be determined at run time.

r

The input string must contain a standard Fortran representation of a single-precision floating point number. The receiving variable must be of type real.

s

As many characters as specified by the base field (unless the delimiter character is encountered first) are copied from the input string to the receiving variable, which must be an array of characters.

t

The string pointer variable 'sp' is set to the value of the width field, or to the length of the input string, whichever is shorter.

u

The values of the width, base, and delimiter character fields specified on this directive become the default values for the remainder of the format directives in the format string.

v

The syntax and semantics of this directive are similar to the 'h' directive above, with the exception that the receiving variable must be a PL/I-style character-varying array.

x

The number of characters specified by the width field (unless the delimiter character is encountered first) are skipped; that is, the specified portion of the input string is ignored.

Implementation

Impossible to explain to the uninitiated reader. Please see the code, and a system guru.

Arguments Modified

sp, fp, ap, a1-a10

Calls

ctoi, ctop, ctoc, length, ctoa, move$, ctov, gctoi, gctol, ctor, ctod, remark

See Also

input (2), conversion routines ('cto?*') (2)

Calling Information

```
integer function delarg (ap)
integer ap
```

Library: vswtlb (standard Subsystem library)

Function

'Delarg' deletes the command line argument indicated by 'ap'. Subsequent arguments have their positions shifted left by one. 'Delarg' returns OK if there is an argument at the position specified by 'ap', and EOF otherwise.

'Delarg' can be used by an argument parsing routine to discard arguments that it recognizes, while leaving other arguments for later action. Then, routines subsequently examining the command line arguments are not bothered by arguments already processed.

Implementation

'Delarg' simply shifts the pointers for arguments following 'ap' in the Subsystem common area down by one and then reduces the argument count by one.

See Also

getarg (2), parscl (2)

Calling Information

```
subroutine delete (symbol, table)
character symbol (ARB)
pointer table
```

Library: vswtlb (standard Subsystem library)

Function

'Delete' removes the character-string symbol given as its first argument from the symbol table given as its second argument. All information associated with the symbol is lost.

The symbol table specified must have been generated by the routine 'mktabl'.

If the given symbol is not present in the symbol table, 'delete' does nothing; this condition is not considered an error.

Implementation

'Delete' calls 'st$lu' to determine the location of the given symbol in the symbol table. If present, it is unlinked from its hash chain. The dynamic storage space allocated to the symbol's node is returned to the system by a call to 'dsfree'.

Calls

st$lu, dsfree

See Also

enter (2), lookup (2), mktabl (2), rmtabl (2), st$lu (6), dsget (2), dsfree (2), dsinit (2), sctabl (2)

Calling Information

```
longreal function dexp$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function raises the constant **e** to the power of the argument. Arguments to the function must be in the closed interval [-22802.46279888, 22623.630826296]. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return value is zero.

It should be noted that the function could simply return zero for sufficiently small arguments rather than signalling an error since the actual function value would be indistinguishable from zero to the precision of the machine. However, there is no mapping to zero in the actual function, and that is why the function signals an error in this case.

Implementation

The routine is implemented as a functional approximation performed on a reduction of the argument. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

err$m (2), exp$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dint$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'dint$m' function implements the Fortran 'dint' function. That is, it takes one double precision value and resets bits in the mantissa to remove any fractional part of the value. The return value is a double precision real.

The 'dint$m' of 1.5 is 1.0, the 'dint$m' of -1.5 is -1.0, and the 'dint$m' of anything less than 1.0 and greater than -1.0 is equal to zero.

The 'dint$m' function has no single precision counterpart in the SWT Math library. The routine, as defined, does not recognize or signal any error conditions. It is written so as to work of both 550 and 750 style machines, despite the internal difference in register structure.

Implementation

The algorithm involved was developed from known register structure; see the source code for specifics.

See Also

SWT Math Library User's Guide

Calling Information

```
DFLD  VALUE
EXT   DINT$P
JSXB  DINT$P
DFST  IVALUE
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'dint$p' function implements the Fortran 'dint' function. It is part of the SWT Math Library 'dint$m' routine and is a special shortcall entrance that can only be used by PMA code. It takes one double precision value and resets bits in the mantissa to remove any fractional part of the value. The return value is a double precision real.

Implementation

The algorithm involved was developed from known register structure; see the source code for specifics.

See Also

dint$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dln$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function implements the natural logarithm (base **e**) function. Arguments must be greater than zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If invalid arguments are supplied to the function the default return is the log of the absolute value of the argument, or zero in the case of a zero argument.

Implementation

The algorithm involved uses a minimax rational approximation on a reduction of the argument. All positive inputs will return a valid result. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

err$m (2), ln$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dlog$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function implements the common logarithm (base 10) function. Arguments should be greater than zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an invalid argument is supplied to the function the default return is the log of the absolute value of the argument, or zero in the case of a zero argument.

Implementation

The algorithm involved uses a minimax rational approximation on a reduction of the argument. All positive inputs will return a valid result. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

err$m (2), log$m (2),
SWT Math Library User's Guide

Calling Information

```
subroutine dodash (valid, array, i, set, j, maxset)
character valid (ARB), array (ARB), set (maxset)
integer i, j, maxset
```

Library: vswtlb (standard Subsystem library)

Function

'Dodash' expands character ranges given in regular expressions. 'Valid' is the set of valid characters in the expansion range (e.g. A-Z for upper case letters, 0-9 for digits, etc.). 'Array' contains the character range string, starting at position 'i'-1. 'Set' not only is the recipient of the expansion, but element 'j'-1 contains the initial character of the range. 'Maxset' is the maximum size 'set' may attain.

Implementation

The indices of the first and last characters in the range are determined, and the substring of 'valid' thus selected is copied into 'set'.

Arguments Modified

i, set, j

Calls

addset, esc, index

See Also

makpat (2), tlit (1), ed (1), se (1)

Calling Information

```
subroutine dsdump (form)
character form
```

Library: vswtlb (standard Subsystem library)

Function

'Dsdump' dumps the contents of memory managed by 'dsinit', 'dsget', and 'dsfree' to ERROUT. It is primarily intended for debugging.

The single argument is either the defined value LETTER, signifying that a character-format dump is desired, or the defined value DIGIT, signifying that an integer-format dump is desired.

Implementation

'Dsdump' simply steps through the memory area (in common block DS$MEM) printing the locations and sizes of available blocks and calling 'dsdbiu' to dump the location, size, and contents of each block that is in use. The dump terminates when the end of memory (as indicated by the contents of the first word of memory) is reached.

The routine 'print' is used for all output.

Calls

print, dsdbiu

Bugs

As advertised, the dump is only semi-readable.

See Also

dsget (2), dsfree (2), dsinit (2), dsdbiu (6)

Calling Information

```
subroutine dsfree (block)
pointer block
```

Library: vswtlb (standard Subsystem library)

Function

'Dsfree' returns a block of storage allocated by 'dsget' to the available space list. The argument must be a pointer returned by 'dsget'.

See the remarks under 'dsget' for required initialization measures.

Implementation

'Dsfree' is an implementation of Algorithm B on page 440 of Volume 1 of *The Art of Computer Programming*, by Donald E. Knuth. The reader is referred to that source for detailed information.

'Dsfree' and 'dsget' maintain a list of free storage blocks, ordered by address. 'Dsfree' searches the list to find the proper location for the block being returned, and inserts the block into the list at that location. If blocks on either side of the newly-returned block are available, they are coalesced with the new block. If the block address does not correspond to the address of any allocated block, 'dsfree' remarks "attempt to free unallocated block" and waits for the user to type a letter "c" to continue. If any other character is typed, the program is terminated.

Calls

getlin, remark

Bugs

The algorithm itself is not the best.

See Also

dsget (2), dsinit (2), dsdump (2), dsdbiu (6)

Calling Information

```
pointer function dsget (w)
integer w
```

Library: vswtlb (standard Subsystem library)

Function

'Dsget' searches its available memory list for a block that is at least as large as its first argument. If such a block is found, its index in the memory list is returned; otherwise, an error message is printed and the program terminates.

In order to use 'dsget', the following declarations must be present:

```
integer mem (MEMSIZE)
common /ds$mem/ mem
```

or the "DS_DECL" system macro can used for the declarations as follows:

```
DS_DECL (mem, MEMSIZE)
```

where MEMSIZE is supplied by the user, and may take on any positive value between 6 and 32767, inclusive. Furthermore, memory must have been initialized with a call to 'dsinit':

```
call dsinit (MEMSIZE)
```

Implementation

'Dsget' is an implementation of Algorithm A' on pages 437-438 of Volume 1 of *The Art of Computer Programming*, by Donald E. Knuth. The reader is referred to that source for detailed information.

'Dsget' searches a linear list of available blocks for one of sufficient size. If none are available, a call to 'error' results; otherwise, the block found is broken into two pieces, and the index (in array 'mem') of the piece of the desired size is returned to the user. The remaining piece is left on the available space list. Should this procedure cause a block to be left on the available space list that is smaller than a threshold size, the few extra words are awarded to the user and the block is removed entirely, thus speeding up the next search for space. If insufficient space is available, 'dsget' reports "out of storage space" and allows the user to obtain a dump of dynamic storage space if he desires.

Calls

dsdump, error, getlin, remark

Bugs

Should probably return error status to the user if space is not found. It is also somewhat annoying for the user to have to declare the storage area, but Fortran prevents effective use of pointers, so this inconvenience is necessary for now.

See Also

dsfree (2), dsinit (2), dsdump (2), dsdbiu (6)

Calling Information

```
longreal function dsin$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function returns the sine of the angle whose measure (in radians) is given by the argument. The absolute value of the argument must be less than 26353588.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

Implementation

The function is implemented as a minimax polynomial approximation. Note that for angles sufficiently small the value of the sine function is equal to the measure of the angle. Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

dasn$m (2), dcos$m (2), dint$p (2), err$m (2), sin$m (2), *SWT Math Library User's Guide*

Calling Information

```
subroutine dsinit (w)
integer w
```

Library: vswtlb (standard Subsystem library)

Function

'Dsinit' initializes an area of storage in the common block DS$MEM so that the routines 'dsget' and 'dsfree' can be used for dynamic storage allocation. The memory to be managed must be supplied by the user, by two declarations of the form:

```
integer mem (MEMSIZE)
common /ds$mem/ mem
```

or the "DS_DECL" system macro can be used for the declarations as follows:

```
DS_DECL (mem, MEMSIZE)
```

The memory size (supplied by the user) must then be passed to 'dsinit' as its argument:

```
call dsinit (MEMSIZE)
```

Implementation

'Dsinit' sets up an available space list consisting of two blocks, the first empty and the second containing all remaining memory. The first word of memory (below the available space list) is set to the total size of memory; this information is used only by the dump routines 'dsdump' and 'dsdbiu'.

Calls

error

See Also

dsget (2), dsfree (2), dsdump (2), dsdbiu (6)

Calling Information

```
longreal function dsnh$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This routine calculates the hyperbolic sine of its argument, defined as sinh(x) = [exp(x) - exp(-x)]/2. The argument must be less than 22623.630826296. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

Implementation

The algorithm involved was adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m, Primos signl$

See Also

dcsh$m (2), dexp$m (2), err$m (2), sinh$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function dsqt$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the square root of a double precision floating point value. Attempts to take the square root of negative values will result in an error. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. The default return in this case will be the square root of the absolute value of the argument.

Implementation

The algorithm involved is based on Newton's approximation method with an initial multiplicative approximation. The argument is scaled to within the range [0.5, 2.0) and then the algorithm is iterated to a solution. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

err$m (2), sqrt$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dtan$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the tangent of the angle whose measure is given (in radians) as the argument to the function. The arguments must have an absolute value of less than 13176794.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

Implementation

The function is calculated based on a minimax polynomial approximation over a reduced argument. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

dcot$m (2), dint$p (2), err$m (2), tan$m (2),
SWT Math Library User's Guide

Calling Information

```
longreal function dtnh$m (x)
longreal x
```

Library: vswtmath (Subsystem mathematical library)

Function

This routine calculates the hyperbolic tangent of their argument, defined as tanh(x) = 2/[exp(2x) + 1]. The function never signals an error and returns valid results for all inputs.

Implementation

Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m

See Also

dexp$m (2), tanh$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function dtoc (val, out, w, d)
long_real val
character out (ARB)
integer w, d
```

Library: vswtlb (standard Subsystem library)

Function

'Dtoc' converts the double precision floating point value in 'val' to a character string in 'out'. The length of the string is returned as the value of 'dtoc'.

The values of 'w' and 'd' control the format of the converted string. Generally speaking, 'd' controls the number of decimal positions or significant digits, and 'w' specifies the maximum length of the field. The following table explains the operation of 'dtoc' for different combinations of 'w' and 'd'. (Fortran and Basic programmers take note: d>12 corresponds to Basic output, 12>=d>=0 corresponds to Fortran 'F' format, and 0>d>=-12 corresponds to Fortran 'E' format)

'd'	*'w'*	*Result*
d>12	w>16	If the value is in the range 1e7>v>=1e-2, it is converted into a BASIC-like fixed-point with no trailing zeroes after the decimal point. Otherwise, it is converted into a BASIC-like exponential format with no trailing zeroes after the decimal point.
	w<=16	An error is returned.
12>=d>=0	-	If possible, the value is converted to a fixed-point format with 'd' positions after the decimal point. Otherwise, it is converted to an exponential format with as many significant digits as possible. If 'w' is less than 8, an exponential conversion is not possible and an error will be returned.
0>d>-12	w>d+6	The number is converted to an exponential format with 'd' significant digits.
	w<=d+6	An error is returned.

To return an error, 'dtoc' places a string consisting of a single question mark in 'out'.

It should be noted that 'w' is roughly equivalent to the

'size' parameter in other conversion routines such as 'itoc' and 'ltoc'; 'w' specifies the maximum number of digits that may be produced. Thus, the maximum number of characters returned in 'out' will never exceed 'w + 1'.

Implementation

'Dtoc' first scales the number into the range 1 > v >= .1. It then determines the format in which the number is to be printed and rounds the value to the proper number of digits. The digits are then extracted in character form. One of several conversion routines is then entered to take the extracted digits and add decimal points, signs, and exponents as required by the 'd' and 'w' specifications.

Arguments Modified

out

Calls

itoc

Bugs

Has been thoroughly debugged, but has not stood the test of time.

See Also

ctod (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
subroutine edit (filename, fdin, fdout)
character filename (ARB)
file_des fdin, fdout
```

Library: vedtlb (Subsystem text editor library)

Function

'Edit' accesses the Subsystem line-oriented text editor. When called, 'edit' begins an editing session, reading editing commands from the file specified by "fdin" and writing editing output on the file specified by "fdout". If "filename" is other than an empty string, 'edit' reads the file into the edit buffer before accepting editing commands.

Since the editor can now call the shell, the shared shell library, 'vshlib', must be loaded along with the editor library 'vedtlb' for any program that calls 'edit'.

'Edit' arranges to catch the 'LOGOUT$' condition. When a LOGOUT$ occurs, 'edit' saves the current contents of the edit buffer in the file =home=/<prog>.logout, where <prog> is the name of the program using 'edit'. For example, 'ed' would have the buffer saved in =home=/ed.logout, while 'moot' would have its buffer saved in =home=/moot.logout.

For complete information on editing commands, see *Introduction to the Software Tools Text Editor*

Implementation

'Edit' is the top-level routine for the Subsystem line editor.

Calling Information

```
integer function encode (str, max, fmt, a1, a2, ..., a10)
integer max
character str (max), fmt (ARB)
untyped a1, a2, ..., a10
```

Library: vswtlb (standard Subsystem library)

Function

'Encode' is a memory-to-memory data format conversion routine, patterned after the Fortran statement of the same name. It takes a number of data items, converts them to character form under the control of a format, and places the results in a designated character string.

The first argument to 'encode' is the string to receive the converted data, and the second argument is the maximum length of that string. The third argument is the format which controls conversion (discussed below). The remaining arguments are data items to be converted. The function return is the number of characters actually transferred to the receiving string.

The format consists of a number of literal characters (to be inserted into the receiving string without interpretation) and format codes (which control conversion of data items). Format codes are paired left-to-right with successive arguments that are to be converted, just as in Fortran formatted I/O. Format codes have the following general form:

```
* [width] [, [base] [, [fill] ] ] form
```

'Width' is a decimal integer whose absolute value is the minimum number of character positions in the receiving string that will be used to store the result of the conversion. If the value is zero, or insufficiently large to accommodate the representation of the data item, as many character positions as necessary, and no more, will be used. If 'width' has a positive value, the converted string is given default justification within the specified field width: numeric items are right justified, and string items are left justified. If 'width' is negative, reverse justification is used.

'Fill' is a single character (blank by default) to be used to pad the converted string to the desired width. Depending on the justification mode, enough instances of the fill character are either prepended or appended to the converted string to make up the difference between its width and the specified field width.

'Base' is a decimal integer that is interpreted differently

according to whether the item being converted is an integer or a string: for integers, the absolute value of 'base' indicates the conversion radix (in the range 2 to 16), and its sign indicates whether the item being converted is to be treated as signed or unsigned (negative values of 'base' yield unsigned conversions); for character strings, 'base' indicates the maximum number of characters that will be extracted from the string item and placed into the receiving string.

'Form' is a single-letter format code that indicates the type of conversion to be performed. Since the interpretation of the other fields depends critically on the form, each form will be discussed individually.

All three of the parameters 'width', 'base' and 'fill' may be represented either explicitly in the format string, or by the character "#", which indicates that the value is to be taken from the current item in the argument list. This allows for a limited form of run-time format specification.

Form *Function*

a Interpret the corresponding argument as an address pointer with the following format:

fr.ssss.wwwwww.bb

where 'f' is present if the pointer is invalid (i.e., would generate a fault if referenced through), 'r' is the protection ring (0-3) associated with the address, 'ssss' is the segment number (0-7777 octal) of the address, 'wwwwww' is the word number (0-177777 octal) of the address, and 'bb', if present, is the bit offset (0-17 octal) of the address. For more information on the significance of the various fields of an address pointer, see Prime publication FDR-3059: *'Assembly Language Programmer's Guide'*.

b Interpret the corresponding argument as a Boolean (Fortran LOGICAL) value. The possible results are the strings "TRUE" and "FALSE", where the number of characters transferred from the result to the receiving string is determined by 'base'. If 'base' is less than 1, only the "T" or the "F" is transferred.

c The argument to be converted is an ASCII character, right-justified and zero-filled in its word. The 'base' specifier does not apply. "*<width>c" is equivalent to "*<width>,1s".

d Interpret the corresponding argument as a double-precision floating-point number. The 'base'

specifier controls the output format. If 'base' is greater than 14, the converted text will resemble BASIC output: up to six significant digits and no trailing zeros. This is the default. If 'base' is between 14 and 0, inclusive, the text will resemble Fortran "F"-format output: 'base' digits to the right of the decimal point. If 'base' is negative, the text will resemble Fortran "E"-format output: scientific notation with "-'base'" digits to the right of the decimal point. (See the conversion routine 'dtoc' for further information on real-to-character conversion.)

g Change the current argument list pointer to 'width'. This form allows argument list elements to be reused for interpretation by multiple format codes. It is particularly useful when 'width' is specified as "#", allowing the binding of argument list elements to format codes to be deferred until run-time.

h Interpret the current argument list element as a Hollerith character string containing ASCII characters packed two-per-word. The 'base' parameter determines the number of characters to be extracted from the Hollerith string.

i Interpret the corresponding argument as a single-precision integer. The absolute value of the 'base' specifier gives the radix to be used for conversion: 2 for binary, 3 for ternary, 16 for hexadecimal, etc. If 'base' is positive, the integer is treated as a signed, two's-complement number with 15 bits of precision, plus a sign bit, with possible values in the range -32768 to +32767. If 'base' is negative, the integer is treated as an unsigned binary number with 16 bits of precision with possible values in the range 0 to 65535.

l The corresponding argument is a double-precision (long) integer. See the comments under "i" for an explanation of the action of the 'base' specifier.

n Insert 'width' NEWLINEs into the receiving string. None of the arguments in the argument list is referenced. If 'width' is less than 1, a single NEWLINE is inserted.

p Interpret the corresponding argument as a period-terminated packed character string (such as that generated by the Ratfor "string"p construct). The 'base' specifier is used as the maximum number of characters to be copied.

r Interpret the corresponding argument as a single-precision floating-point number. The comments under "d" above also apply to this form.

s Interpret the corresponding argument as an unpacked EOS-terminated character string (such as generated by "string"s in Ratfor, or as returned by 'getlin'). The 'base' specifier gives the maximum number of characters to be transferred.

t Tab to the character position in the receiving string specified by 'width'. If the position is beyond the current end of the string, pad the string to that position with instances of the 'fill' character. The 'base' parameter is not used.

u Set default values for the 'width', 'base' and 'fill' parameters. Subsequent formatting codes that do not specify these values will be interpreted as if the values specified here had been used.

v Interpret the corresponding argument as a PL/I-style varying character string. The 'base' specifier once again gives the maximum number of characters that will be transferred to the receiving string.

x Transfer an entire field of fill characters to the receiving string. The 'base' specifier is unused. The 'fill' specifier gives the character to be used for filling the field; the default is a blank.

y Interpret the corresponding argument as a YES/NO value such as those used frequently throughout the Subsystem. The possible results are the strings "YES" and "NO". The interpretation of the 'base' parameter is similar to that used with the "b" form.

The following forms are supported for compatibility with an earlier version of the 'print' subroutine. They should not be used in new code.

f Treat the argument as a double-precision floating-point number. "F" is equivalent to "d" in every way.

j The corresponding argument is a single-precision integer. "*<width>,<base>j" is equivalent to "*<width>,-<base>i".

m The corresponding argument is a long integer. "*<width>,<base>m" is equivalent to "*<width>,-<base>l".

Since 'encode' is a complex routine, a few samples may be helpful in getting the hang of its use. For example, the following call will convert two integers to decimal free-form, with a comma and space between them:

```
len = encode (str, MAXLINE, "*i, *i"s, xcoord, ycoord)
```

These calls will print an "address" and the contents of the array 'memory' at that address, in base 16 with zero fill:

```
len = encode (str, MAXLINE, "(*4,-16,0i) *4,-16,0i*n"s,
   address, memory (address))
call putlin (str, STDOUT)
```

A typical line of output from the above might be

```
(A000) 0002
```

A filename for use by 'open' might be composed like this:

```
call encode (name, MAXPATH, "=temp=/*s*2,,0i"s,
   username, sequence_number)
```

If 'username' was a string containing "SYSTEM" and 'sequence_number' contained the integer 1, the previous call would set 'name' to the string "=temp=/SYSTEM01".

Implementation

Since Fortran passes arguments to subroutines by reference, 'encode' does not need to declare the actual type of its arguments. The type is determined by scanning the format string and associating arguments with forms, left-to-right. 'Encode' calls various "something-to-character" conversion routines to translate data from internal form to character string, which it then simply places in the receiving string (checking to make sure the length of the receiver is not exceeded). 'Encode' is not simple, and a reading of the code is necessary if full understanding of its implementation is required.

Arguments Modified

str

Calls

atoc, ctoc, ctoi, ptoc, vtoc, gitoc, gltoc, rtoc, dtoc,

remark

Bugs

No more than ten items may be encoded. This routine is highly dependent on the ability of Prime's Fortran to handle calls with varying numbers of arguments.

See Also

input (2), print (2), conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function enter (symbol, info, table)
character symbol (ARB)
integer info (ARB)
pointer table
```

Library: vswtlb (standard Subsystem library)

Function

'Enter' places the character-string symbol given as its first argument, along with the information given in its second argument, into the symbol table given as its first argument. The function return value (which is ignored by almost everyone) is a pointer to the dynamic storage area containing the text of the symbol.

The symbol table used must have been created by the routine 'mktabl'. The size of the info array must be at least as large as the symbol table node size, determined at table creation time.

Should the given symbol already be present in the symbol table, its information field will simply be overwritten with the new information.

'Enter' uses the dynamic storage management routines, which require initialization by the user; see 'dsinit' for further details.

Implementation

'Enter' calls 'st$lu' to find the proper location for the symbol. If the symbol is not present in the table, a call to 'dsget' fetches a block of memory of sufficient size, which is then linked onto the proper chain from the hash table. Once the location of the node for the given symbol is known, the contents of the information array are copied into the node's information field.

Calls

st$lu, dsget, scopy

See Also

lookup (2), delete (2), mktabl (2), rmtabl (2), st$lu (6), dsget (2), dsfree (2), dsinit (2), sctabl (2)

Calling Information

```
integer function equal (str1, str2)
character str1 (ARB), str2 (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Equal' is used to compare EOS-terminated strings. The two arguments are the strings to be compared; the function return is YES if they are equal (on a character-by-character basis), NO if they are not.

Implementation

'Equal' simply loops through each of the two strings, comparing characters. If a mismatch occurs, NO is returned; otherwise, YES is returned. Comparison stops when an EOS is encountered. To be equal, strings must be of equal length (EOS's must match).

See Also

strcmp (2)

Calling Information

```
subroutine err$m (ptr_to_cfh)
pointer ptr_to_cfh
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'err$m' procedure is provided as a default handler for the SWT_MATH_ERROR$ condition. It takes a single argument, a 2 word pointer as defined by the condition mechanism, and prints information about the routine and values which signalled the fault. All output from the 'err$m' routine is sent to ERROUT. Included in the output is the name of the faulting routine, the location from which the faulting routine was called, the value of the argument involved, and the default return value to be used.

The Primos MKON$F routine can be used to set up this on-unit handler in Ratfor and Fortran 66 programs. The Primos subroutine MKON$P can be used in Fortran 77 and PL/P programs.

The user may wish to copy and modify the source code for the 'err$m' procedure so as to provide a more specific form of error handling. If this is done, it would probably be a good idea to rename the user's version to something other than 'err$m.'

Calls

print

See Also

Primos mkon$f, Primos mkon$p, Primos signl$,
SWT Math Library User's Guide

Calling Information

```
subroutine error (message)
character message (ARB)

Library:  vswtlb (standard Subsystem library)
```

Function

'Error' is a Kernighan/Plauger routine used to report fatal errors. The message passed to it is printed on ERROUT, and the program then terminates. An error status of 1000 is passed back to the command interpreter, which then decides whether or not shell program execution will proceed.

Implementation

'Error' calls 'remark' to print the error message on file ERROUT. Error status is set by a call to 'seterr', then a 'stop' statement is executed to make a normal return back to the Subsystem command interpreter. (Note that Ratfor converts 'stop' statements into calls to the subroutine 'swt'.)

Calls

remark, swt, seterr

See Also

remark (2), swt (2)

Calling Information

```
character function esc (array, i)
character array (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Esc' examines the string 'array' at character position 'i'. If there is an escape character ("@") at this point, then the next character in the array is examined. If it is a letter "n", the function return is the character NEWLINE. If it is a letter "t", the function return is the character TAB. If the character is neither "n" nor "t", or if the escape character was not present, the function return is the character itself. In all cases, 'i' is incremented to point to the next unexamined character in the string.

Arguments Modified

i

Bugs

Should probably handle "b" for backspace, arbitrary octal and hex character constants, and a few other things.

Calling Information

```
subroutine exec (path_name)
character path_name (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Exec' is a means of chaining execution to another program. The argument is a pathname specifying the Primos run file of the program to be executed. 'Exec' returns if an error was encountered, otherwise control is passed to the called program and no return is possible.

Implementation

'Exec' calls the Subsystem routine 'getto' to get to the UFD in which the file to be executed exists. The existence of the file is checked with the function 'findf$'; if the file exists, it is executed via a call to the Primos routine RESU$$. If the file is not found, or could not be reached by 'getto', 'exec' returns to the calling program. Note that since a call to RESU$$ is used, 'exec' can be used to execute P300 run-file format programs only.

Calls

getto, findf$, Primos resu$$

Bugs

Since Primos provides no way to tell if a file is executable object code, there is always a strong possibility that resuming a file of unknown type will cause an unrecoverable error.

See Also

execn (2), getto (2), findf$ (2)

Calling Information

```
subroutine execn (path_name)
packedchar path_name (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

The function of 'execn' is almost identical to that of 'exec'. The only difference is in the form of the argument passed to the two routines. 'Exec' expects an EOS terminated string; 'execn' expects a string of characters packed two per word, terminated with a period. Like 'exec', 'execn' executes the program whose location is specified by the given pathname if that is possible; if an error occurs, control returns to the calling program. On a successful call, control passes to the called program, and the calling program is lost.

Implementation

'Execn' calls 'ptoc' to unpack its argument into a temporary area; this temporary area is then passed as an argument to 'exec', which does all the real work.

Calls

ptoc, exec

Bugs

Same as 'exec'.

See Also

exec (2), ptoc (2)

Calling Information

```
longreal function exp$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function raises the constant **e** to the power of the argument. Arguments to the 'exp$m' routine must be in the closed interval [-89.415985, 88.029678]. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default function return value is zero.

It should be noted that the function could simply return zero for sufficiently small arguments rather than signalling an error since the actual function value would be indistinguishable from zero to the precision of the machine. However, there is no mapping to zero in the actual function, and that is why the function signals an error in this case.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The routine is implemented as a functional approximation performed on a reduction of the argument. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

dexp$m (2), dint$p (2), err$m (2), ln$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function expand (template, str, strlen)
integer strlen
character template (ARB), str (strlen)
```

Library: vswtlb (standard Subsystem library)

Function

'Expand' is used to convert Subsystem path templates into strings. Templates are used to make the directory structure of the Subsystem user-selectable without the expense of code modification. The first argument of 'expand' should be an EOS-terminated string containing a template to be expanded; the second argument should be a string to receive the result, and the third argument should be the maximum allowable length of the result. The function return is the length of the expanded template, or ERR if an undefined or ill-formed template is present.

The template consists of uninterpreted characters, which are passed through to the expanded version unchanged, and identifiers surrounded by equals signs, which are replaced by template text and then rescanned. If the template must contain uninterpreted equal signs, they must be "doubled" (eg. for one =, use ==).

See 'lutemp' for a list of templates currently supported.

Implementation

'Expand' maintains indices into the template, the receiving string, and a pushback buffer, which is initially empty. As long as the pushback buffer is empty, 'expand' copies characters from the template to the receiving string. When a single equals sign ("=") is encountered, characters are stored in another buffer until a trailing equals sign is seen. This buffer is passed to 'lutemp', which places the resulting template expansion in the pushback buffer. As long as the pushback buffer is nonempty, 'expand' scans it instead of the original template; this allows template expansions to contain additional templates.

Arguments Modified

str

Calls

lutemp

See Also

lutemp (6), follow (2), getto (2)

Calling Information

```
subroutine fcopy (in, out)
file_des in, out
```

Library: vswtlb (standard Subsystem library)

Function

'Fcopy' is a routine that copies the contents of one file to another. The two arguments specify the file descriptors of the source and destination files, respectively. Both files must be open with the proper access modes (i.e., READ or READWRITE access for the source, and WRITE or READWRITE access for the destination); neither is rewound before or after the copy.

Implementation

'Fcopy's strategy depends on the types of devices represented by the source and destination files; if both are disk files, the routines 'readf' and 'writef' are called repeatedly to transfer large blocks of data. For all other combinations of source and destination device types, 'getlin' and 'putlin' are called repeatedly to transfer one line at a time. Even for disk files, 'getlin' may be called, to insure that the buffer state is consistent.

Calls

getlin, mapsu, putlin, readf, writef

Bugs

There is no provision for an error return of any sort; no status is passed back to the calling program to indicate success or failure of the copy.

See Also

getlin (2), putlin (2), readf (2), writef (2)

Calling Information

```
integer function filcpy (from, to)
character from (ARB), to (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Filcpy' is used to copy a file from one place to another, insuring that the copy possesses the same attributes (protection keys, time of last modification, read/write lock, and dumped/modified bits) as the original.

The 'from' argument is the pathname of the source file; the 'to' argument is the pathname of the destination. The function return is OK if the copy succeeded, ERR otherwise.

Implementation

'Filcpy' obtains the 'from' files attributes with a call to the Primos routine ENT$RD and then opens it with a call to the Primos routine SRCH$$. An attempt is then made to open the 'to' file with the same type. If the attempt fails, the 'to' file is removed and an error exit occurs. If the destination file is a non-empty segment directory, it is then cleaned out with 'rmseg$'.

The file is copied by 'cpfil$' or 'cpseg$', if it is an ordinary file or a segment directory, respectively. If it is ordinary, it is truncated after the copy to insure that no old data remains.

Several calls to the Primos routine SATR$$ are then made to set the destination file's attributes.

Calls

getto, Primos srch$$, Primos ent$rd, ptoc, remove, rmseg$, cpseg$, cpfil$, Primos prwf$$, Primos satr$$

See Also

fcopy (2), cp (1)

Calling Information

```
type file_name = array [1..7] of char;
procedure file$p (file: text; name: file_name); extern;

Library:  vswtlb (standard Subsystem library)
```

Function

'File$p' may be called from a Pascal program to allow that program to access the redirection and pipe features of Subsystem I/O. Each call to 'file$p' can connect any Pascal file variable to any Subsystem file. The call to 'file$p' is equivalent to, and is used instead of, the Pascal "reset" and "rewrite" statements.

To use 'file$p', it should be declared as a level 1 procedure:

```
procedure file$p(var f: text; n: file_name); extern;
```

It may then be called to connect input and output, for example:

```
file$p(input,'STDIN  ');
file$p(output,'STDOUT ');
```

The name of the Subsystem file to be connected must be in upper case and space filled. It may be any of the following: STDIN, STDIN1, STDIN2, STDIN3, ERRIN, STDOUT, STDOUT1, STDOUT2, STDOUT3, or ERROUT.

Implementation

'File$p' searches a table of Subsystem file names to determine the standard unit number for the requested file. If the name is not in the table, 'file$p' calls the Primos routine ERRPR$ to signal a bad file name error. If the name is in the table, a call to 'mapsu' is made to map the standard unit number into the Subsystem unit number, and 'flush$' is called to ensure that the file has been written to disk. Then 'mapfd' is called to determine the Primos file unit associated with the file, if there is one. If there is no associated file unit, the file is on the terminal and 'file$p' initializes the Pascal file control block for a terminal file. Otherwise, the Primos routine ATTDEV is called to connect the requested file, and the Pascal file control block is initialized. After the type of file is determined (disk or terminal) the appropriate name is copied into the Pascal file control block; 'TTY' for terminal files and for /dev/null, and the Primos treename for disk files.

Arguments Modified

file

Calls

ctop, equal, flush$, gfnam$, mapfd, mapsu, mktr$, move$, ptoc, Primos attdev, Primos errpr$

Bugs

Files redirected to /dev/null are not supported.

Pascal terminal input does not behave correctly, the backspace key cannot be used to erase previously entered characters.

See Also

init$p (2)

Calling Information

```
subroutine filset (delim, array, i, set, j, maxset)
character delim, array (ARB), set (maxset)
integer i, j, maxset
```

Library: vswtlb (standard Subsystem library)

Function

'Filset' expands a character class specification in 'array' into a list of characters in 'set'. 'I' specifies the starting position in 'array', 'j' gives the starting position in 'set', and 'maxset' gives the maximum size of 'set'. Expansion stops when there is insufficient room in 'set' or when the character contained in 'delim' is encountered in 'array'.

Character sets consist of arbitrary characters, two lower-case letters separated by a hyphen, two upper-case letters separated by a hyphen, or two digits separated by a hyphen. The last three cases represent a range of characters, including the endpoints.

Implementation

Ordinary characters are simply stuffed into 'set' with calls to 'addset'. The range notation is expanded by 'dodash'.

Arguments Modified

i, set, j

Calls

addset, esc, index, dodash

See Also

dodash (2), makpat (2)

Calling Information

```
integer function filtst (path, zero, permissions, exists,
   type, readable, writeable, dumped)
character path (MAXPATH)
integer zero, permissions, exists, type
integer readable, writeable, dumped
```

Library: vswtlb (standard Subsystem library)

Function

'Filtst' may be used to perform a number of tests on a named file. The arguments specify which tests are to be performed, in a way described below. The function return is YES if all tests succeeded, NO if any one failed, or ERR if a test could not be made.

'Path' is an EOS-terminated string containing the pathname of the file to be tested. 'Zero' is -1 if the file is to be tested for non-zero length, 1 if for zero length, and 0 if it is not to be tested for length. 'Permissions' is a bit mask of protection keys, as returned by the Primos routines DIR$RD and ENT$RD. or 0 if protections are not to be tested. 'Exists' is -1 if the file is to be tested for nonexistence, 1 if for existence, and 0 if it is not to be tested for existence. 'Type' is 0 if the file's type is not to be tested; otherwise, it is the same as the file type returned by Primos ENT$RD logically or'ed with octal 100 (to distinguish between the SAM file type and the "no test" value). 'Readable' is -1 if the file is to be tested for non-readability, 1 if for readability, and 0 if no test is to be performed. 'Writeable' is -1 if the file is to be tested for non-writeability, 1 if for writeability, and 0 if writeability is not to be tested. 'Dumped' is -1 if the file is to be tested for not being dumped, 1 if for being dumped, and 0 if the dumped bit is not to be tested.

Implementation

Various calls to the Primos routines SRCH$$, ENT$RD and PRWF$$ are made to check the attributes specified by the arguments. The function return is YES if and only if the results of all tests were true, ERR if Primos detected an error during any test, NO otherwise.

Calls

getto, Primos srch$$, open, close, Primos prwf$$, Primos ent$rd, Primos at$hom

See Also

file (1), finfo$ (6)

Calling Information

```
integer function follow (path, sethome)
character path (ARB)
integer sethome
```

Library: vswtlb (standard Subsystem library)

Function

'Follow' changes the current working directory. 'Path' is a pathname assumed to be composed of nodes that are only UFDs, with no data files or segment directories. 'Follow' "attaches" (Primos terminology) to each of the directories and subdirectories named in the pathname in sequence, thus "following" a path through the file system to the last directory named. 'Sethome' is a set-home key; if zero, the home directory remains unchanged, and the pathname specifies a new working directory; if 'sethome' equals one, the pathname specifies a new home directory.

'Follow' returns ERR if there was a syntax error in the pathname or if a directory could not be attached, and OK if the attach was successful.

Implementation

'Follow' sets up an on-unit for the "BAD_PASSWORD$" condition in order to handle errors during the attaching process. If the pathname supplied is empty, 'follow' attaches back to the home directory by calling the Primos routine AT$HOM. Otherwise, 'follow' calls the routine 'getto' to reach the parent directory of the last directory in the path, and then calls the Primos routine AT$REL to take the final step in the path. If 'getto' fails to parse the pathname or reach the parent directory or if AT$REL encounters an error, 'follow' attaches back to the home directory and returns ERR; if successful it returns OK.

Calls

bponu$, ctov, ptoc, getto, Primos at$hom, Primos at$rel, Primos break$, Primos mklb$f, Primos mkonu$

See Also

getto (2), cd (1)

Calling Information

```
integer function gctoi (str, i, radix)
character str (ARB)
integer radix, i
```

Library: vswtlb (standard Subsystem library)

Function

'Gctoi' is similar to the routine 'ctoi', except that it accepts base indicators and signs. Conversion begins on the string 'str' at position 'i'. The converted integer is returned as the value of the function. 'I' will be updated to indicate the first position not used in the conversion.

Input to 'gctoi' consists of a number containing an optional plus or minus sign, an optional base indicator, and a string of digits allowable for the input base. The base indicator consists of the (decimal) radix of the desired base followed by the letter "r" (in the style of Algol 68). The digits corresponding to the numbers 10 through 15 are entered as the letters "a" through "f". If no base indicator occurs in the string, the number in 'radix' is used as the default base. Conversion stops when a character not allowable in the number is encountered.

Implementation

'Gctoi' first checks for a leading sign, and records it if found. If the first one or two digits of the number are numeric and if they are followed by a lower case "r", then they are converted to binary and used as the radix of the remaining digits; otherwise, the 'radix' argument is used. The remaining digits of the number are converted by a simple multiply-and-add-successive-digits algorithm.

Arguments Modified

i

Calls

index

See Also

gctol (2), ctoi (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
long_int function gctol (str, i, radix)
character str (ARB)
integer radix, i
```

Library: vswtlb (standard Subsystem library)

Function

'Gctol' is similar to the routine 'ctol', except that it accepts base indicators and signs. Conversion begins on the string 'str' at position 'i'. The converted integer is returned as the value of the function. 'I' will be updated to indicate the first position not used in the conversion.

Input to 'gctol' consists of a number containing an optional plus or minus sign, an optional base indicator, and a string of digits allowable for the input base. The base indicator consists of the (decimal) radix of the desired base followed by the letter "r" (in the style of Algol 68). The digits corresponding to the numbers 10 through 15 are entered as the letters "a" through "f". If no base indicator occurs in the string, the number in 'radix' is used as the default base. Conversion stops when a character not allowable in the number is encountered.

Implementation

'Gctol' first checks for a leading sign, and records it if found. If the first one or two digits of the number are numeric and if they are followed by a lower case "r", then they are converted to binary and used as the radix of the remaining digits; otherwise, the 'radix' argument is used. The remaining digits of the number are converted by a simple multiply-and-add-successive-digits algorithm.

Arguments Modified

i

Calls

index

See Also

gctoi (2), ctol (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function geta$f (ap, str, len)
integer ap, len
character str (*)
```

Library: vswtlb (standard Subsystem library)

Function

'Geta$f' fetches an argument from the Subsystem command line in a format useable by a Fortran program. The arguments are analogous to those used by 'getarg'. 'Ap' is the number of the argument to be fetched: 0 for the command name, 1 for the first argument, 2 for the second, etc. 'Str' is a string to receive the argument, while 'len' is the number of characters allocated to 'str'. The function return value is either the length of the argument string actually returned, or EOF (-1) if there is no argument in that position.

Implementation

'Geta$f' simply calls 'getarg' with the argument pointer, and then calls 'ctop' to convert the result into the proper Fortran format.

Arguments Modified

str

Calls

ctop (2), getarg (2)

Bugs

If 'len' is an odd number, 'geta$f' will return at most 'len - 1' characters of an argument.

See Also

getarg (2), geta$plg (2), geta$p (2)

Calling Information

```
type string128 = array [1..128] of char;
function geta$p (    ap:  integer;
                 var str: string128;
                     len: integer) : integer;
```

Library: vswtlb (standard Subsystem library)

Function

'Geta$p' fetches an argument from the Subsystem command line in a format useable by a Pascal program. The arguments are analogous to those used by 'getarg'. 'Ap' is the number of the argument to be fetched: 0 for the command name, 1 for the first argument, 2 for the second, etc. 'Str' is a string to receive the argument, while 'len' is the number of characters allocated to 'str'. The function return value is either the length of the argument string actually returned, or EOF (-1) if there is no argument in that position.

To use 'geta$p', it must be declared as a level 1 procedure in the Pascal program:

```
function geta$p (    ap:  integer;
                 var str: string128;
                     len: integer) : integer; extern;
```

It may then be called as a function wherever desired.

Implementation

'Geta$p' simply calls 'getarg' with the argument pointer, and then calls 'ctop' to convert the result into the proper Pascal format, after it has blank-filled 'str'.

Arguments Modified

str

Calls

ctop (2), getarg (2)

Bugs

If 'len' is an odd number, 'geta$p' will at most, return 'len - 1' characters of the argument.

See Also

getarg (2), geta$f (2), geta$plg (2)

Calling Information

```
geta$plg:   procedure (ap, str, len) returns (fixed);
   declare
      ap    fixed,
      str   character (128) varying,
      len   fixed;
```

Library: vswtlb (standard Subsystem library)

Function

'Geta$plg' fetches an argument from the Subsystem command line in a format useable by a PL/I G (or PL/P) program. The arguments are analogous to those used by 'getarg'. 'Ap' is the number of the argument to be fetched: 0 for the command name, 1 for the first argument, 2 for the second, etc. 'Str' is a string to receive the argument, while 'len' is the number of characters allocated to 'str'. The function return value is either the length of the argument string actually returned, or EOF (-1) if there is no argument in that position.

To use 'geta$plg', it must be declared in the PL/I program:

```
   declare
      geta$plg entry (fixed, char (128) var, fixed)
                     returns (fixed)
```

It may then be called as a function wherever desired.

Implementation

'Geta$plg' simply calls 'getarg' with the argument pointer, and then calls 'ctov' to convert the result into the proper PL/I format.

Arguments Modified

str

Calls

ctov (2), getarg (2)

Bugs

If 'len' is an odd number, 'geta$plg' will return at most 'len - 1' characters of an argument.

See Also

getarg (2), geta$f (2), geta$p (2)

Calling Information

```
integer function getarg (n, arg, arg_len)
integer n, arg_len
character arg (arg_len)
```

Library: vswtlb (standard Subsystem library)

Function

'Getarg' is used to retrieve the arguments supplied on the command line that invoked a program. The first argument is the ordinal of the command argument to be fetched: 1 for the first, 2 for the second, etc. The second argument is a string to receive the command argument being fetched; the third argument is the maximum length of the string. The function return from 'getarg' is the length of the command argument fetched, if the fetch was successful; EOF if the argument could not be fetched. Argument 0 is the name of the command calling 'getarg'.

Implementation

The Subsystem command interpreter maintains the list of command arguments in its linked-string storage area. 'Getarg' uses the array of pointers into this area supplied by the command interpreter to locate the desired argument, then copies the characters to the user's buffer one-by-one.

Arguments Modified

arg

Bugs

A program can have at most 256 arguments. There is no convenient way to find out how many arguments have been supplied on an invocation without searching through the entire list with calls to 'getarg'.

See Also

chkarg (2), getkwd (2)

Calling Information

```
integer function getccl (arg, i, pat, j)
character arg (ARB), pat (MAXPAT)
integer i, j
```

Library: vswtlb (standard Subsystem library)

Function

'Getccl' converts the character class specification starting at 'arg(i)' into a pattern element starting at 'pat(j)'. The pattern element consists of a character count followed by a list of all characters in the class. The function return is OK if the character class was successfully converted, ERR otherwise.

For a discussion of character classes, see either *Introduction to the Software Tools Text Editor* or *Software Tools*.

Implementation

If the first character in the class specification is a tilde, the class generated is a negated class rather than the standard class. Room is then reserved for the character count, and 'filset' is called to expand the specification into the vector of characters.

Arguments Modified

i, pat, j

Calls

addset, filset

See Also

makpat (2), addset (2), filset (2), *Introduction to the Software Tools Text Editor*, *Software Tools*

Calling Information

```
character function getch (c, fd)
character c
integer fd
```

Library: vswtlb (standard Subsystem library)

Function

'Getch' is used to get a character from a file. The first argument is assigned the value of the character fetched; the second argument is the file descriptor of the file to be read. If end-of-file occurs on the input file, the character returned is EOF. The function return is always identical to the first argument (character read or EOF).

Implementation

'Getch' calls 'getlin' with a very short line buffer (1 character + EOS). 'Getlin' thus returns one character in the buffer, which becomes the value returned by 'getch'. If 'getlin' returns EOF, 'getch' also returns EOF.

Arguments Modified

c

Calls

getlin

See Also

getlin (2), putch (2)

Calling Information

```
integer function getkwd (keyword, value, length, default)
character keyword (ARB), value (ARB), default (ARB)
integer length
```

Library: vswtlb (standard Subsystem library)

Function

'Getkwd' searches the list of arguments supplied on the command line for a string that matches the contents of 'keyword'. 'Keyword' must contain an EOS-terminated string. If a matching argument is found, the argument string that immediately follows it in the argument list is returned in the array 'value'; otherwise, the string contained in 'default' is copied into 'value'. In either case, the length of the string returned in 'value' (excluding EOS) is returned as the result of the function. 'Length' gives the size of the 'value' array in words; no more than 'length'-1 characters will be copied.

Implementation

'Getarg' is called to access each successive argument string. Each is compared to the supplied keyword, and if a match is found, 'getarg' is called again to retrieve the immediately following argument. If that argument doesn't exist or if the keyword is not found, as much of the default string as will fit is copied into the 'value' array, one character at a time.

Arguments Modified

value

Calls

equal, getarg

See Also

chkarg (2), getarg (2)

Calling Information

```
integer function getlin (line, fd [, line_length] )
integer line_length
file_des fd
character line (line_length)
```

Library: vswtlb (standard Subsystem library)

Function

'Getlin' is the primary Subsystem input routine. It is used to read a line from a file, which may be assigned to any device recognized by the Subsystem. The first argument is a string to receive data transferred from the file; the second argument is the file descriptor of the file from which data will be read; the optional third argument is the maximum length of the receiving string. Characters are transferred from the file to the string buffer until (1) end-of-file occurs, (2) a NEWLINE is encountered, or (3) the string buffer is completely full. Unless end-of-file occurs, the function return is the length of the string returned in the buffer; on end-of-file, EOF is returned. The third argument, 'line_length', is optional. If omitted, the value MAXLINE is assumed. At most, 'line_length' - 1 characters, followed by an EOS, are transferred to the buffer.

Implementation

'Getlin' first calls 'mapsu' to map any standard port descriptor it may have been passed into a Subsystem file descriptor. The file specified must be readable and not at end-of-file; if these conditions are not met, EOF is returned immediately. The Primos routine MISSIN is called to determine if the line length argument is missing; if it is, the default value of MAXLINE is assumed. Regardless of device type, 'flush$' is called to place the file buffer in a consistent state if the last operation performed on the file was not a 'getlin' or 'getch'. One of two device dependent drivers ('dgetl$' for disk or 'tgetl$' for terminal) is called to do the real work of getting data into the buffer. Note: Reads on a null device always return EOF.

Arguments Modified

line

Calls

mapsu, dgetl$, tgetl$, flush$, Primos missin

Bugs

The current optional use of the 'line_length' argument is somewhat shaky. There is need for more devices than "terminal" and "disk" (system console, for example).

See Also

dgetl$ (6), tgetl$ (6), mapsu (2), putlin (2), flush$ (6)

Calling Information

```
integer function getto (path, name, pwd, attach)
character path (ARB)
packedchar name (MAXPACKEDFNAME), pwd (3)
integer attach
```

Library: vswtlb (standard Subsystem library)

Function

'Getto' attaches to the UFD containing the file specified in the last node of the pathname 'path.' In order to make further operations on the file convenient, it packs the name of the last node in the path (usually a file name) two characters per word, left justified, blank filled into the argument 'name', and similarly packs the password (if any) into the argument 'pwd'. Passwords are always mapped to upper case.

'Getto' returns ERR if the UFD containing the file could not be attached; OK otherwise. If 'getto' returns with 'attach' set to NO, the current UFD was not changed; if YES, then the parent directory of the last file in the path became the current directory.

Implementation

'Getto' sets up an on-unit for the "BAD_PASSWORD$" condition to handle errors during the attaching process. Then it expands any pathname templates in 'path', converting the *Software Tools*-style pathname into a Primos treename by calling 'mktr$'. If the pathname supplied is empty, 'getto' attaches back to the home directory by calling the Primos routine AT$HOM. Otherwise, it loops through the nodes in the treename until it has attached to the parent directory of the last node. If the treename refers to a primary UFD (one in the master file directory of some disk), 'getto' uses the Primos routine AT$ANY to attach to the named UFD, and then attaches to the MFD on the same disk using the Primos routine AT$ABS. If the treename is a fully specified path including packname, 'getto' checks see to if the packname specifies a logical disk number. If it does, the Primos routine AT$ is called to attach to the primary UFD on the specified disk pack; otherwise the AT$ABS is called to do the attach. Then 'getto' attaches to the MFD on the same disk using AT$ABS. If the last node of the treename has not been reached, 'getto' attaches down the path by calling the Primos routine AT$REL. If any errors occur during the attaching process, 'getto' attaches back to the home directory and returns ERR.

Arguments Modified

name, pwd, attach

Calls

bponu$, ctov, expand, mapstr, mktr$, Primos at$, Primos at$abs, Primos at$any, Primos at$hom, Primos at$rel, Primos break$, Primos mklb$f, Primos mkonu$

See Also

follow (2), open (2), remove (2)

Calling Information

```
subroutine getvdn (filename, pathname [, username])
character filename (ARB), pathname (ARB), username (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Getvdn' is used to return the full pathname of a file in a user's shell variables storage directory. Such files are often used for small amounts of data that must be secure (e.g. mail files, encryption parameters, shell variables, etc.).

The 'filename' argument is an EOS-terminated string containing the simple name of a file in the variables directory. The 'pathname' argument receives the full pathname of the given file. The 'username' argument, if present, specifies the particular user whose variables directory is to be referenced. If 'username' is missing, or is equal to the user making the call to 'getvdn', then the user's Subsystem password is inserted in the returned pathname, allowing full owner rights in the variables directory.

An example: If the current user's login name is "foo" and his Subsystem password is "bar", then the call

```
call getvdn (".vars"s, pathname)
```

would return the string "=vars=/foo:bar/.vars" in 'pathname'.

Implementation

'Getvdn' calls 'ctoc' to start the pathname with the string "=vars=/", then simply uses 'scopy' to append the other items of information as needed. The Subsystem password is available in the variable 'Passwd' in the shell's common areas.

Arguments Modified

pathname

Calls

ctoc, date, equal, length, scopy, Primos missin

Bugs

Security of the variables directory can be broken by a Trojan horse.

See Also

vfyusr (2), expand (2), lutemp (6)

Calling Information

```
integer function getwrd (in, i, out)
integer in (ARB), out (ARB)
integer i
```

Library: vswtlb (standard Subsystem library)

Function

'Getwrd' retrieves the next word from the line buffer 'in' at current position 'i', and places it in 'out'. A word is a string of characters delimited by blanks or newlines (also EOS, if the word occurs at the end of the line). The new current position is updated in 'i', and the length of the word is returned as the function value.

Implementation

Any blanks, starting at the current position 'i' in the string, are skipped. Characters from 'in' are then copied to 'out', starting at position 'i', until the next character to be copied is either an EOS, a blank, or a NEWLINE. When this happens, the count of characters is returned.

Arguments Modified

i, out

See Also

ctoc (2)

Calling Information

```
integer function gfdata (key, pathname, infbuf, attach, aux)
integer key, attach
untyped infbuf, aux
character pathname (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

This function returns information about a file system entry according to the value of the parameter 'key.' There are thirteen declarations for values of 'key' in the standard SWT definitions. Their names and returned values are:

FILE_UFDQUOTA -- this key reads information about disk record quotas. The object named in 'pathname' must be a directory. 'aux' returns a value of YES (as an integer) if the object is a quota directory, otherwise 'aux' is returned as NO and the function value is ERR. 'infbuf' is an array of 6 long_ints. 'infbuf (1)' is set to the number of words per disk record on the partition (440 or 1024), 'infbuf (2)' is set to the number of records used in the directory, 'infbuf (3)' is set to the current quota, 'infbuf (4)' is set to the total number of records used in the directory and its contents, 'infbuf (5)' is set to the time-record product, and 'infbuf (6)' is set to the file system date-time modification stamp for the directory.

FILE_TYPE -- this key returns an EOS-terminated string in the parameter 'infbuf' which describes the type of the file named in 'pathname.' Standard types are "sam", "dam", "sgs", "sgd", "ufd", and "acat". Special files may return types of: "mfd", "boot", "dskrat", or "badspt". 'infbuf' must be at least 7 characters long. 'aux' is unmodified.

FILE_DMBITS -- this key returns information about the dumped bit and the Primos II modification bit. 'infbuf' must be at least 2 words long. infbuf(1) will have the value YES if the dumped bit is set, NO otherwise. infbuf(2) will have the value YES if the modification bit is set, NO otherwise. 'aux' is unmodified.

FILE_RWLOCK -- this key causes the current read/write lock to be encoded in an EOS-terminated string and returned. The format of the string in 'infbuf' is the same as used in 'lf' and 'chat.' 'infbuf' must be at least 4 characters long. 'aux' is unmodified.

FILE_TIMMOD -- this key returns the date and time of last modification. 'infbuf' must be at least 6 words long. The date and time are returned as integers. infbuf(2)

is set to the month (1 to 12), infbuf(3) is set to the day, and infbuf(1) is set to the year modulo 100. Infbuf(4) is set to the hour past midnight (0-23), infbuf(5) is set to the minute, and infbuf(6) is set to the seconds. Note that the file system date/time stamp has a resolution of only 4 seconds. 'aux' is unmodified.

FILE_ACL -- this key encodes the ACL pairs protecting the object into 'infbuf' with each pair separated by a blank. 'infbuf' should be declared as a character array of size MAXACLLIST. 'aux' should be an array MAXPATH long. 'aux (1)' gets set to an integer indicating the type of object protecting the item named by pathname: 0 is a specific ACL, 1 is an acat, 2 is a default specific ACL, 3 is a default acat, and 4 means the object is an acat. The name of the ACL object confering the protection is returned as an EOS string in 'aux (2)' and on.

FILE_ACCESS -- this key calculates the access for a specific user to the object named by 'pathname'. The user name (or group name) is specified in 'aux'; if 'aux' is simply an EOS, then access is calculated for the current user. 'infbuf' is encoded with the access rights, and should be declared to be at least 8 characters long.

FILE_PRIORITYACL -- this key encodes the current priority ACL set on the disk partition containing the object named in 'pathname'. The ACL is returned as an EOS terminated string in 'infbuf'; 'infbuf' should be declared to be at least MAXACLLIST characters long. 'aux' is unchanged.

FILE_DELSWITCH -- this key causes 'infbuf' to be set to YES if the file delete protect bit is set on, NO otherwise. The file delete switch is valid only for ACL-protected items. 'aux' is unchanged.

FILE_SIZE -- this key returned the size of the item named in 'pathname'. If the object is a file the 'infbuf' is the long_int number of words of data in the file. If 'pathname' is a ufd or segdir, then 'infbuf' is set to the number of words contained in the total structure. If the item is a ufd, then 'aux' is a long_int set to the number of words per disk records (440 or 1024); if it is a file or segdir then 'aux' is unchanged.

FILE_FULL_INFO -- this key causes 'infbuf' to be returned as an array of 24 words, structured as in the call to Primos "ent$rd". 'aux' is unchanged.

FILE_PROTECTION -- this key causes the current file protection attributes to be encoded into an EOS-

terminated string in the parameter 'infbuf.' The form of the string is the same as the form presented to the 'chat' command or returned by the 'lf' command. 'infbuf' must be at least 6 characters long.

FILE_PASSWORDS -- this key causes the current passwords for the directory named in 'pathname' to be returned as EOS terminated strings. The owner password is returned in 'infbuf' and the nonowneehr password is returned in 'aux'. Both 'infbuf' and 'aux' must be at least 7 words long.

The 'pathname' argument is any standard EOS-terminated pathname, and may contain templates. The function value is OK if the information was fetched, ERR otherwise. ERR may indicate that the file does not exist, that the caller does not have rights in the directory containing the file, a bad key, or any of a number of other file system errors.

The "attach" key is the same as in the "getto" function --- it indicates if the directory attach point had to be changed to get to the file entry being examined.

Implementation

The function uses "getto" to get to the directory containing the file, then it calls the Primos routine "ent$rd" to obtain file system information. Finally, it decodes the information into 'infbuf' based on the value of 'key.' The routines "q$read", "acl$01", "acl$50", "szfil$", and "szseg$" may also be called depending on the key supplied.

Arguments Modified

infbuf, attach, aux

Calls

ctoc, ptoc, vtoc, getto, follow, index, equal, mapdn, ctov$f, szfil$, szseg$, move$, mktr$, acl$01, acl$50, Primos nameq$, Primos gpas$$, Primos ent$rd, Primos q$read, Primos at$hom, Primos srch$$, Primos calac$

Bugs

If 'infbuf' or 'aux' are not long enough, odd behavior may result.

See Also

chat (1), lacl (1), lf (1), sacl (1), sfdata (2)

Calling Information

```
integer function gfnarg (name, state)
character name (MAXPATH)
integer state (4)
```

Library: vswtlb (standard Subsystem library)

Function

'Gfnarg' is used to fetch "file name" arguments (those not beginning with a dash) from the command line. More importantly, it observes the convention that a "-n" argument implies that file names are to be read from an input file until EOF is reached, rather than simply from the argument list.

In summary, each time 'gfnarg' is called, it returns a file name in 'name'. From its initial state, 'gfnarg' will fetch the next argument from the command line. If the argument is a file name, then it is passed back in 'name', 'state' is updated (in ways to be discussed later), and the function return is OK. If the argument begins with "-n", it indicates that file names are to be take from a file. ("-n" implies the standard port STDIN, "-n2" implies STDIN2, "-n3" implies STDIN3, and "-nfilename" implies the named file.) Successive calls of 'gfnarg' then return successive lines of the named file (sans newline at the end of each line). When EOF is reached, 'gfnarg' resumes its scan of the command line arguments. Whenever a non-filename, non-"-n" argument is encountered, 'gfnarg' returns ERR. When no more filename arguments are available, 'gfnarg' returns EOF.

As a boundary condition, if there are no arguments on the command line, 'gfnarg' returns the name "/dev/stdin1", reflecting the convention that programs invoked without arguments take their input from their standard ports.

Use of 'gfnarg' requires one initialization step. The first element of the state vector must be set to 1, then the first call to 'gfnarg' may be issued. For example:

```
state (1) = 1
call gfnarg (argument, state)
```

'Gfnarg' will maintain the state vector internally after the initialization.

Implementation

The four elements of the state vector have the following interpretations: state(1) is the current state of 'gfnarg', which may be (1) uninitialized, (2) reading arguments from the command line, (3) reading from an input file, (4) end of

command line reached, or (5) no more file arguments are available (EOF); state(2) is the index of the next argument in the command line; state(3) is the file descriptor for the current "-n" input file; state(4) is a count of the number of file name arguments returned so far. The initial state is used by 'gfnarg' to set up the other elements of the state vector. From that point on, the contents of the argument list cause changes in state. As a special case, if the end of the command line is reached without finding any file names, "/dev/stdin1" is returned and state(1) is altered to cause an EOF return on the next call to 'gfnarg'.

Arguments Modified

name, state

Calls

close, error, getarg, getlin, open, print, scopy

Bugs

Should probably return argument length, like 'getarg' does, whenever it finds a filename.

See Also

getarg (2)

Calling Information

```
integer function gitoc (int, str, size, base)
integer int, size, base
character str (size)
```

Library: vswtlb (standard Subsystem library)

Function

'Gitoc' will convert a single precision (16 bit) integer to a character string representation in any radix from 2 to 16 (inclusive). The integer to be converted may be considered as either a signed, two's-complement number with 15 bits of precision, or as an unsigned number with 16 bits of precision.

'Int' is the integer to be converted; 'str' is a character array into which the string representation will be stored; 'size' is the size of 'str'. The absolute value of 'base' is the conversion radix. If 'base' is negative, then 'int' is treated as an unsigned number; otherwise, 'int' is considered to be a signed, two's-complement number. If the specified radix is not in the range 2:16, then a decimal conversion is performed.

For a signed conversion, if the integer is less than zero, its absolute value is preceded by a minus sign in the converted string; a positive number is never preceded by a sign.

The function return is the number of characters required to represent the integer.

Implementation

'Gitoc' uses a typical divide-and-remainder algorithm to perform the conversion; that is, a digit is generated by taking the remainder when the integer is divided by the radix. For signed conversions, the absolute value of the number is first taken, the digits generated, and the minus sign inserted if needed. For unsigned conversions, the least significant bit of the number is saved, and then the number is shifted right one bit position to put it into the precision range of 15 bits (and effectively dividing the unsigned number by 2). Then, as each digit value is generated, it is doubled and added to the carry from the previous digit position (with the initial carry being the saved least significant digit) and a new carry value is generated.

Arguments Modified

str

See Also

gltoc (2), itoc (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function gklarg (args, str)
integer args (26)
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Gklarg' is used to parse a key-letter argument string. Such an argument consists of a dash ("-") followed by any number of letters (in upper or lower case).

All elements in the array 'args' must be preset to one of two values before calling 'gklarg'. Elements corresponding to allowable option letters should be initialized to zero; all others should contain -1.

'Gklarg' sets the elements of 'args' that correspond to any option letters found in 'str' to the value 1. The function return is ERR if 'str' does not begin with a dash, or if any disallowed option letters are encountered, OK otherwise.

Implementation

'Gklarg' first verifies that the string given in 'str' begins with a dash. If it does not, ERR is returned. The remainder of the string is examined character-by-character. If a letter is encountered, and the corresponding element of 'args' is nonnegative, then the element is set to one. Otherwise the value ERR is returned immediately.

Arguments Modified

args

Calls

mapdn

See Also

gfnarg (2), chkarg (2), getkwd (2)

Calling Information

```
integer function gltoc (int, str, size, base)
long_int int
integer size, base
character str (size)
```

Library: vswtlb (standard Subsystem library)

Function

'Gltoc' will convert a double precision (32 bit) integer to a character string representation in any radix from 2 to 16 (inclusive). The integer to be converted may be considered as either a signed, two's-complement number with 31 bits of precision, or as an unsigned number with 32 bits of precision.

'Int' is the integer to be converted; 'str' is a character array into which the string representation will be stored; 'size' is the size of 'str'. The absolute value of 'base' is the conversion radix. If 'base' is negative, then 'int' is treated as an unsigned number; otherwise, 'int' is considered to be a signed, two's-complement number. If the specified radix is not in the range 2:16, then a decimal conversion is performed.

For a signed conversion, if the integer is less than zero, its absolute value is preceded by a minus sign in the converted string; a positive number is never preceded by a sign.

The function return is the number of characters required to represent the integer.

Implementation

'Gltoc' uses a typical divide-and-remainder algorithm to perform the conversion; that is, a digit is generated by taking the remainder when the integer is divided by the radix. For signed conversions, the absolute value of the number is first taken, the digits generated, and the minus sign inserted if needed. For unsigned conversions, the least significant bit of the number is saved, and then the number is shifted right one bit position to put it into the precision range of 31 bits (and effectively dividing the unsigned number by 2). Then, as each digit value is generated, it is doubled and added to the carry from the previous digit position (with the initial carry being the saved least significant bit) and a new carry value is generated.

Arguments Modified

str

See Also

gitoc (2), ltoc (2), other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function gtattr (attr)
integer attr
```

Library: vswtlb (standard Subsystem library)

Function

'Gtattr' returns the value of the attribute 'attr' that the user desires. Currently, the following attribute types are accepted :

TA_SE_USEABLE - indicates whether the terminal can use the screen editor ('se'). The returned value is either YES or NO.

TA_VTH_USEABLE - indicates whether the terminal is supported by the Virtual Terminal Handler package (VTH). The returned value is either YES or NO.

TA_UPPER_ONLY - indicates whether the terminal supports only the upper case character set. The returned value is either YES or NO.

The value of each of the above attributes is set upon entry into the Subsystem, but can be changed by executing the 'term' command.

Implementation

'Gtattr' first verifies that the given attribute is a legal one; if it isn't, then NO is returned. If the attribute is legal, its value is obtained from the Subsystem common area and returned as the function value.

See Also

term (1), term_type (1), VTH routines (vt?*) (2)

Calling Information

```
integer function gtemp (str, nm, repl)
character str (ARB), nm (MAXARG), repl (MAXARG)
```

Library: vswtlb (standard Subsystem library)

Function

'Gtemp' takes a NEWLINE or EOS terminated character string in 'str' and assigns the first blank-delimited token to 'nm' and the remaining characters of the string to 'repl'. Leading and trailing blanks are removed from both 'nm' and 'repl'. Ratfor-style (beginning with a sharp sign) comments are also ignored. If the input string consists only of blanks and comments, 'gtemp' returns EOF and 'nm' and 'repl' are unmodified; otherwise 'gtemp' returns OK.

Implementation

'Gtemp' first removes any trailing comments (begun with a sharp sign, as in Ratfor) and leading and trailing blanks from the string. It then selects the first blank-delimited token from the string, and assigns it to 'nm'. Then, after removing intervening blanks, 'gtemp' assigns the remaining characters of the string to 'repl'.

Arguments Modified

nm, repl

See Also

ldtmp$ (6)

Calling Information

```
integer function gttype (ttype)
character ttype (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Gttype' obtains the user's terminal type by calling routines to (1) look in the Subsystem common area, (2) look in the =termlist= file, and (3) ask the user. The terminal type is checked in each case, and if it is invalid, it is ignored. The function returns YES if the character string representing the terminal type is returned in 'ttype' and NO otherwise. Since 'gttype' will return NO only if the user refuses to give a terminal type (by entering end-of-file), most programs just terminate with a call to 'error' if 'gttype' returns NO.

Implementation

'Gttype' calls 'ttyp$r' to obtain the terminal type from the common area. If the string is empty or if the terminal type in the common area is invalid, it calls 'ttyp$f' to obtain the terminal type in the "=termlist=" file. If no valid type is present in =termlist=, 'gttype' calls 'ttyp$q' to request the terminal type from the user.

Arguments Modified

ttype

Calls

ttyp$f, ttyp$q, ttyp$r

See Also

gtattr (2), ttyp$v (6)

Calling Information

```
integer function gvlarg (str, state)
character str (ARB)
integer state (4)
```

Library: vswtlb (standard Subsystem library)

Function

'Gvlarg' returns the next argument and updates the state vector; it is normally used in conjunction with 'gklarg' and 'gfnarg'. If the next argument begins with a hyphen, 'gvlarg' returns an empty string. 'Gvlarg' returns EOF if the argument list has been exhausted; otherwise it returns OK.

'Gvlarg' exists solely to hide the structure of the state vector when an argument must be fetched between calls to 'gklarg' and 'gfnarg'.

Implementation

Trivial.

Arguments Modified

str, state

Calls

error, getarg

See Also

gfnarg (2), gklarg (2)

Calling Information

```
integer function index (str, c)
character str (ARB)
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Index' searches the string given as its first argument for the character given as its second argument. If the character is found, its index in the string is returned; if it is not found, zero is returned.

Implementation

A simple loop checks for the character in the string; if found, an immediate return takes place. If the loop terminates normally, the value zero is returned.

Bugs

The arguments should be reversed.

Calling Information

subroutine init$f

Library: vswtlb (standard Subsystem library)

Function

A call to 'init$f' from a Fortran 66 or Fortran 77 program attaches Fortran unit 5 to the file open as standard input (either disk or terminal) and attaches Fortran unit 6 to the file open as standard output (either disk or terminal). The attachment of unit 1 to the terminal is not changed.

To use 'init$f', it must be called as the first executable statement in the main program:

```
call init$f
```

Implementation

First 'init$f' calls 'flush$' on standard input and standard output to clean up any unfinished Subsystem I/O. 'Init$f' then calls the Subsystem 'mapfd' to determine the Primos file unit attached to standard input. If 'mapfd' returns a file descriptor, 'init$f' calls Fortran 'attdev' to attach unit 5 to that Primos disk unit; otherwise, 'init$f' calls the Primos routine ATTDEV to attach unit 5 to the terminal. The procedure is then repeated for standard output and Fortran unit 6.

Calls

flush$, mapfd, mapsu, Primos attdev

Bugs

Files redirected to /dev/null are not supported.

See Also

init$p (2), init$plg (2)

Calling Information

procedure init$p;

Library: vswtlb (standard Subsystem library)

Function

A call to 'init$p' from a Pascal program attaches the Pascal file INPUT to the file open as standard input (either disk or terminal) and attaches the Pascal file OUTPUT to the file open as standard output (either disk or terminal).

To use 'init$p', it must be declared as a level 1 procedure,

```
procedure init$p; extern;
```

and then called as the first statement after the BEGIN in the main program:

```
init$p;
```

Implementation

First 'init$p' calls 'flush$' on standard input and standard output to clean up any unfinished Subsystem I/O. 'Init$p' then calls the Subsystem 'mapfd' to determine the Primos file unit attached to standard input. If 'mapfd' returns a file descriptor, 'init$p' tweaks the Pascal file control block 'p$ainp' and calls the Primos routine ATTDEV to establish the unit mapping. Next 'init$p' calls 'gfnam$' to obtain the pathname of the disk file. If 'gfnam$' returns a valid pathname, 'mktr$' is called to convert the pathname into a Primos treename. This treename is copied into the Pascal file control block so that Pascal can use it when reporting I/O errors. If 'gfnam$' returns ERR, the message 'pathname unobtainable' is copied into the Pascal file control block as the file name. Otherwise, since INPUT is already directed to the terminal, 'init$p' does nothing. This procedure is then repeated for standard output and the Pascal file OUTPUT.

Calls

ctop, flush$, gfnam$, mapfd, mapsu, mktr$, Primos attdev

Bugs

Files redirected to /dev/null are not supported.

See Also

init$f (2), init$plg (2), file$p (2)

Calling Information

```
init$plg:  procedure;
```

Library: vswtlb (standard Subsystem library)

Function

A call to 'init$plg' from a PL/I G program attaches the PL/I G file SYSIN to the file open as standard input (either disk or terminal) and attaches the PL/I G file SYSPRINT to the file open as standard output (either disk or terminal).

To use 'init$plg', it must be declared in the main program,

```
declare init$plg entry;
```

and then called before any executable statements:

```
call init$plg;
```

Implementation

First 'init$plg' calls 'flush$' on standard input and standard output to clean up any unfinished Subsystem I/O. 'Init$plg' then calls the Subsystem 'mapfd' to determine the Primos file unit attached to standard input. If 'mapfd' returns a file descriptor, 'init$plg' opens SYSIN using that file descriptor; otherwise, it opens SYSIN on the terminal. The procedure is then repeated for standard output and the PL/I G file SYSPRINT.

Calls

flush$, mapfd, mapsu, Primos p$open

Bugs

Files redirected to /dev/null are not supported.

Output on SYSPRINT not followed by a line boundary (e.g. PUT SKIP) will be ignored when the file is directed to disk. It is usually best to terminate all programs with a PUT SKIP to insure that this line boundary is present.

See Also

init$p (2), init$f (2)

Calling Information

subroutine init

Library: vswtlb (standard Subsystem library)

Function

At version 8.1 of the Subsystem, 'init' became obsolete. It remains to help users find programs which were compiled before Release 8.1. It will print the following error message:

> You are trying to run a pre-version 9 compilation.
> Please recompile and try again.

and then exit to the Subsystem.

'Init' should not be used in new compilations. The Ratfor preprocessor 'rp' no longer automatically inserts a call to 'init' in each main program it processes. Users should remove all references to 'init' from their programs, and recompile as soon as possible.

The Version 8 compatibility library, which allowed the use of programs compiled before Release 8.1, is no longer supported.

Calling Information

```
integer function input (fd, fmt, a1, a2, ...)
file_des fd
packed_char fmt (ARB)   -or-   character fmt (ARB)
untyped a1, a2, ...
```

Library: vswtlb (standard Subsystem library)

Function

'Input' is an input routine designed for ease of use. It allows the user to specify a file from which to read, a format to control input from the file, and any number of items to be read. The first argument is the file descriptor of the file to be used for input. The second argument is a format string (discussed below). The remaining arguments (zero or more) are items to be input according to format control. The function return is the number of items set as a result of the input request, or EOF if end-of-file was encountered.

The format string is a PERIOD-terminated packed character string (such as that generated by the Ratfor "string"p construct) or an unpacked, EOS-terminated string (such as that generated by the "string"s construct). The format string contains literal characters which will be output on the user's terminal if the given input file refers to the terminal device, and format control directives consisting of an asterisk (*) followed by a single lower-case letter describing the input format for the next item in the argument list. For a complete description of format control directives, please see the Reference Manual entry for 'decode'.

Note that each call to 'input' causes one call to 'getlin' to read the input text; the text read is used to fill as many items as possible, but any remaining text is lost. This corresponds to BASIC and FORTRAN input procedures.

When erroneous input is detected, 'input' outputs the incorrect value to the terminal, discards the rest of the input line, and requests reentry of the incorrect value from the the terminal. The user may type in the corrected item and continue normally.

Literal characters in the format will be ignored if the file specified by 'fd' is not directed to the terminal. This feature allows a program to prompt for input from the terminal, but suppress the prompt for input from a file. Note that if no prompting at the terminal is desired, literal characters should not be included in the format string.

A few short examples may clarify the operation of 'input'.

To input two integers, with a prompt, one might use

```
junk = input (STDIN, "Enter i and j:  *i*i"s, i, j)
```

To input an array of double-precision floating point numbers, one might use

```
i = 1
while (input (file, "*d"s, array (i)) ~= EOF) {
   i += 1
   if (i > ARRAY_SIZE)
      call error ("too many numbers to handle"p)
   }
```

Implementation

'Input' outputs prompt characters as it finds them in the format string, calls 'getlin' to obtain a string of input from the proper file, and then calls 'decode' to do the actual conversion of as many items as possible. Since the design of 'decode' was heavily influenced by the requirements of 'input', careful reading of the code for both routines is recommended.

Arguments Modified

a1, a2, ...

Calls

ctoc, decode, getlin, index, isatty, print, ptoc, putch

Bugs

At most ten items may be input. 'Input' depends heavily on the ability of Prime's Fortran to handle subroutines with varying numbers of arguments. The ability to buffer some input text to satisfy later calls would be nice, but is difficult without some static storage.

See Also

print (2), encode (2), decode (2), getlin (2), conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function isadsk (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Isadsk' returns YES if the file referenced by the file descriptor in 'fd' is a disk file; otherwise it returns NO. All file descriptors, including standard ports, can be tested.

Implementation

'Isadsk' simply looks in the Subsystem common area at the device type in the file descriptor and returns YES or NO accordingly.

Calls

mapsu

See Also

isatty (2), isnull (2)

Calling Information

```
integer function isatty (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Isatty' returns YES if the file referenced by the file descriptor in 'fd' is connected to a terminal; otherwise it returns NO. All file descriptors, including standard ports, can be tested.

Implementation

'Isatty' simply looks in the Subsystem common area at the device type in the file descriptor and returns YES or NO accordingly.

Calls

mapsu

See Also

isadsk (2), isnull (2)

Calling Information

```
integer function isnull (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Isnull' returns YES if the file referenced by the file descriptor in 'fd' is connected to the null device; otherwise it returns NO. All file descriptors, including standard ports, can be tested.

Implementation

'Isnull' simply looks in the Subsystem common area at the device type in the file descriptor and returns YES or NO accordingly.

Calls

mapsu

See Also

isadsk (2), isatty (2)

Calling Information

```
integer function isph$ (dummy)
untyped dummy

Library: vswtlb (standard Subsystem library)
```

Function

'Isph$' returns YES if the caller is a phantom user or NO otherwise. The single argument 'dummy' is not referenced and exists only because a function in FORTRAN 66 must have at least one argument.

Implementation

'Isph$' simply returns the value of the 'Isphantom' variable in the SWT common block, which is set during Subsystem initialization.

Arguments Modified

none

See Also

isph (1)

Calling Information

```
integer function itoc (int, str, size)
integer int, size
character str (size)
```

Library: vswtlb (standard Subsystem library)

Function

'Itoc' converts the integer given as its first parameter to a character string that is returned as its second parameter. The last 'size' - 1 digits of the number, and no more, are returned. The number is left justified, with a leading minus sign if the number is negative. The function return is the length of the character string returned.

Implementation

'Itoc' performs a rather standard conversion by using modular arithmetic to fetch one digit at a time from the integer value supplied. The character string generated is placed backward in the receiving field, then reversed just before exit.

Arguments Modified

str

See Also

other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function jdate (month, day, year)
integer month, day, year
```

Library: vswtlb (standard Subsystem library)

Function

'Jdate' is used to determine the Julian date corresponding to a given month, day, and year. (For example, January first of any year has Julian date 1; December 31st might have Julian date 365 or 366, depending on whether the given year is a leap year or not.) The function return is the Julian date calculated.

Implementation

'Jdate' simply adds up the number of days in all months before the month given, then adds the number of days given. If the year specified is a leap year, February is given 29 days instead of the usual 28.

See Also

date (1), wkday (2), date (2)

Calling Information

```
integer function length (str)
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Length' returns the length of the string passed as its first parameter.

Implementation

A simple loop is used to count characters until an EOS is encountered.

Bugs

Slow.

Calling Information

```
longreal function ln$m (x)
real x

Library: vswtmath (Subsystem mathematical library)
```

Function

This function implements the natural logarithm (base **e**) function. Arguments must be greater than zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled due to an invalid argument the default return is the log of the absolute value of the argument, or zero in the case of a zero argument.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The algorithm involved uses a minimax rational approximation on a reduction of the argument. All positive inputs will return a valid result. The algorithm was adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

dln$m (2), err$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function locate (c, pat, offset)
character c, pat (MAXPAT)
integer offset
```

Library: vswtlb (standard Subsystem library)

Function

'Locate' returns YES if 'c' is a member of the character class at 'pat(offset)', NO otherwise.

Implementation

A character class is stored as a size, followed by a vector of characters in the class. 'Locate' simply checks all the characters in the vector; if 'c' matches one, then the return value is YES.

See Also

makpat (2), omatch (2), amatch (2)

Calling Information

```
longreal function log$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function implements the common logarithm (base 10) function. Arguments must be greater than zero. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled due to an invalid argument the default return is the log of the absolute value of the argument, or zero in the case of a zero argument.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The algorithm involved uses a minimax rational approximation on a reduction of the argument. All positive inputs will return a valid result. It is adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

dlog$m (2), err$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function lookup (symbol, info, table)
character symbol (ARB)
untyped info (ARB)
pointer table
```

Library: vswtlb (standard Subsystem library)

Function

'Lookup' examines the symbol table given as its third argument for the presence of the character-string symbol given as its first argument. If the symbol is not present, 'lookup' returns 'NO'. If the symbol is present, the information associated with it is copied into the information array passed as the second argument to 'lookup', and 'lookup' returns 'YES'.

The symbol table used must have been created by the routine 'mktabl'. The size of the information array must be at least as great as the symbol table node size, specified at its creation.

Note that all symbol table routines use dynamic storage space, which must have been previously initialized by a call to 'dsinit'.

Implementation

'Lookup' calls 'st$lu' to determine the location of the symbol in the table. If 'st$lu' returns NO, then the symbol is not present, and 'lookup' returns NO. Otherwise, 'lookup' copies the information field from the appropriate node of the symbol table into the information array and returns YES.

Arguments Modified

info

Calls

st$lu

See Also

enter (2), delete (2), mktabl (2), rmtabl (2), st$lu (6), sctabl (2), dsinit (2), dsget (2), dsfree (2)

Calling Information

```
integer function ltoc (int, str, size)
long_int int
integer size
character str (size)
```

Library: vswtlb (standard Subsystem library)

Function

'Ltoc' is used to convert long integers to decimal character representation.

'Int' is the long integer to be converted; 'str' is the string to receive the ASCII representation; 'size' is the size of 'str'. The function return is the number of characters required to represent 'int'.

'Ltoc' duplicates the function of 'itoc' for long integers.

Implementation

Standard modular-arithmetic conversion. See 'itoc' for details.

Arguments Modified

str

See Also

other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function makpat (arg, from, delim, pat)
character arg (ARB), delim, pat (MAXPAT)
integer from
```

Library: vswtlb (standard Subsystem library)

Function

'Makpat' converts the standard character-string form of a regular expression into the internal form used by the remainder of the pattern matching routines. The argument 'arg' is the regular expression to be converted; 'from' specifies the starting position of the pattern in 'arg'; 'delim' contains a termination character which, when encountered, causes conversion to stop; 'pat' receives the internal form of the regular expression. The function returns the index of the delimiter in 'arg' if the conversion succeeded, ERR otherwise.

For a full discussion of patterns and pattern matching, see *Introduction to the Software Tools Text Editor* or, of course, *Software Tools*.

Implementation

'Makpat' traverses the regular expression a character at a time, building the internal pattern with calls to 'addset'. To build character classes, 'makpat' calls 'getccl'; to build closures it calls 'stclos'. Calls to 'esc' handle escape sequences in the regular expression. 'Makpat' treats the special cases of "*" at beginning-of-line, "%" not at BOL, and "$" not at end-of-line as regular characters.

'Makpat' takes an error return if the internal form becomes too large, if an attempt is made to use closure on an illegal pattern element, if there are too many tagged subpatterns, if not all tagged subpatterns are properly closed, or if 'delim' is never encountered.

Arguments Modified

pat

Calls

addset, esc, getccl, stclos

See Also

match (2), amatch (2), find (1), change (1), ed (1), se (1), *Introduction to the Software Tools Text Editor*, *Software Tools*

Calling Information

```
integer function maksub (arg, from, delim, sub)
character arg (ARB), delim, sub (MAXPAT)
integer from
```

Library: vswtlb (standard Subsystem library)

Function

'Maksub' converts the character representation of a substitution string starting at "arg(from)" into an internal form in 'sub'. Conversion proceeds until there is insufficient room in 'sub' to proceed or until the character in 'delim' is encountered. The function return is the next unexamined position in 'arg'.

For a full discussion of the syntax of substitution strings, see either *Introduction to the Software Tools Text Editor* or *Software Tools*.

Implementation

Straightforward scan of the substitution string. At present, the metacharacter sequences in a substitution string are "&" (meaning the string matched) and "@<digit>" (meaning the <digit>th tagged subpattern matched). 'Esc' is used to handle escape sequences; all other characters are substituted literally.

Arguments Modified

sub

Calls

addset, esc, type

See Also

makpat (2), addset (2), change (1), ed (1), se (1), *Introduction to the Software Tools Text Editor*, *Software Tools*.

Calling Information

```
character function mapdn (c)
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Mapdn' determines if the character passed as its parameter is an upper case letter or not. If not, the function return is equal to the character; otherwise, the function return is the value of the character mapped to lower case.

Implementation

'Mapdn' expects all upper case letters to be contiguous and arranged in a collating sequence with capital A low and capital Z high (internal ASCII satisfies these requirements). If the character lies between 'A'c and 'Z'c, it is mapped to lower case by adding 'a'c - 'A'c. The function return is the mapped value. The parameter is left unchanged.

Bugs

Depends heavily on ASCII character code, in exchange for speed.

See Also

mapup (2), mapstr (2)

Calling Information

```
integer function mapfd (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

Certain applications require the Primos funit number associated with a given open disk file. 'Mapfd' retrieves the funit number corresponding to a file descriptor. If the file open on the given file descriptor is not a disk file, the function return is ERR; otherwise, it is the desired funit number.

Implementation

The Primos funit associated with each file descriptor is available in the Subsystem I/O common area. 'Mapfd' simply checks to make sure the specified file descriptor corresponds to a disk file, then returns the funit.

Calls

mapsu

See Also

mapsu (2)

Calling Information

```
integer function mapstr (str, case)
character str (ARB)
integer case
```

Library: vswtlb (standard Subsystem library)

Function

'Mapstr' is used to map the case of all the letters in a string. 'Str' is the string to be mapped; 'case' is UPPER if letters are to be mapped to upper case, anything else for lower case (usually LOWER).

The length of the string is returned as the function's value.

Implementation

A loop is used to examine each character in the string; the actual mapping is done by adding or subtracting the difference between ASCII 'a' and ASCII 'A'.

Arguments Modified

str

See Also

mapup (2), mapdn (2), tlit (1)

Calling Information

```
file_des function mapsu (std_unit)
file_des std_unit
```

Library: vswtlb (standard Subsystem library)

Function

'Mapsu' is used to map standard units (such as STDIN, STDOUT, and ERROUT) to the file descriptor associated with the file to which they are currently equivalent. It is not intended for general use.

Implementation

'Mapsu' checks the file unit mapping information contained in the Subsystem I/O common area. If the parameter 'std_unit' is one of the following:

STDIN	STDOUT
STDIN1	STDOUT1
STDIN2	STDOUT2
STDIN3	STDOUT3
ERRIN	ERROUT

then the function return is the file descriptor corresponding to these standard units; otherwise, the function return is the same as the value of 'std_unit'.

Note that if the standard port mapping table contains a circular definition, the file descriptor of the user's terminal will be returned.

Calling Information

```
character function mapup (c)
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Mapup' is the inverse of 'mapdn'. If the character 'c' is a lower case letter, the function return is the corresponding upper case letter; otherwise, the function return is the same as 'c'.

Implementation

In 'mapup', as in 'mapdn', considerable use is made of the internal ASCII character code. If 'c' is between 'a'c and 'z'c, 'c' - 'a'c + 'A'c is returned; otherwise, 'c' is returned.

Bugs

Inordinate dependence on properties of character code.

See Also

mapdn (2), mapstr (2)

Calling Information

```
file_mark function markf (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Markf' is used to determine the current position of an open file. The position is normally recorded and later reused by 'seekf' for random I/O.

The single argument specifies a file whose position is desired. The function return is ERR if the position could not be determined or if "position" has no meaning for the device currently associated with the given file descriptor.

Implementation

If necessary, 'markf' calls 'flush$' to empty the Subsystem buffer belonging to the file. If the file is associated with a terminal device, 'tmark$' is called to get the position. Similarly, 'dmark$' is called if the file is a disk file. The null device is always at position zero.

Calls

mapsu, flush$, tmark$, dmark$

Bugs

'Markf' may fail between two characters in a line, because files under Primos are word-addressed, rather than byte-addressed. 'Markf' should only be used at word boundaries (for binary files) or line boundaries (for standard character files).

See Also

dmark$ (6), tmark$ (6), seekf (2)

Calling Information

```
integer function match (lin, pat)
character lin (ARB), pat (MAXPAT)
```

Library: vswtlb (standard Subsystem library)

Function

'Match' attempts to find a match for a regular expression anywhere in a given line of text. The first argument contains the text line; the second contains the pattern to be matched. The function return is YES if the pattern was found anywhere in the line, NO otherwise.

The pattern in 'pat' is a standard Subsystem encoded regular expression. 'Pat' can be generated most conveniently by a call to the routine 'makpat'.

Implementation

'Match' calls 'amatch' at each position in 'lin', returning YES whenever 'amatch' indicates it found a match. If the test fails at all positions, 'match' returns NO.

Calls

amatch

Bugs

Not exactly blindingly fast.

See Also

amatch (2), makpat (2), maksub (2), catsub (2), find (1), change (1), ed (1), se (1), *Introduction to the Software Tools Text Editor*, *Software Tools*

Calling Information

```
pointer function mktabl (nodesize)
integer nodesize
```

Library: vswtlb (standard Subsystem library)

Function

'Mktabl' creates a symbol table for manipulation by the routines 'enter', 'lookup', 'delete', and 'rmtabl'. The symbol table is a general means of associating data with a symbol identified by a character string. The sole argument to 'mktabl' is the number of (integer) words of information that are to be associated with each symbol. The function return is the address of the symbol table in dynamic storage space (see 'dsinit' and 'dsget'). This value must be passed to the other symbol table routines to select the symbol table to be manipulated.

Note that dynamic storage space must be initialized by a call to 'dsinit' before using any symbol table routines.

Implementation

'Mktabl' calls 'dsget' to allocate space for a hash table in dynamic memory. Each entry in the hash table is the head of a linked list (with zero used as a null link) of symbol table nodes. 'Mktabl' also records the nodesize specified by the user, so 'enter' will know how much space to allocate when a new symbol is entered in the table.

Calls

dsget

See Also

enter (2), lookup (2), delete (2), rmtabl (2), st$lu (6), dsget (2), dsfree (2), dsinit (2), sctabl (2)

Calling Information

```
file_des function mktemp (mode)
integer mode
```

Library: vswtlb (standard Subsystem library)

Function

'Mktemp' is used to make a temporary file. The single parameter is an i/o mode (WRITE or READWRITE). The temporary file is created in directory =temp=, so write permission in the home directory is not required. 'Mktemp' returns a file descriptor if the temporary was successfully created, ERR otherwise.

Implementation

'Mktemp' consists of a loop that calls 'create' to attempt the creation of files with names of the form "=temp=/tm?*", where "?*" represents a string of decimal digits. If such a file can be created, 'mktemp' returns a file descriptor that can be used to access it; otherwise, ERR is returned.

Calls

encode, create

See Also

rmtemp (2), close (2), create (2), open (2)

Calling Information

```
character function mntoc (buf, p, default)
character buf (ARB), default
integer p
```

Library: vswtlb (standard Subsystem library)

Function

'Mntoc' is used to convert a standard ASCII mnemonic (e.g. ACK, BEL, BS) into an ASCII character code. The argument 'buf' is an EOS-terminated string presumed to contain either a single character or a two- or three-character ASCII mnemonic (in either upper or lower case), starting at position 'p'. The function return depends on the outcome of the conversion as follows: (1) if 'buf' contains only one character, the function return is equivalent to that character; (2) if 'buf' contains an ASCII mnemonic terminated by a nonalphanumeric character, the function return is the character code associated with that mnemonic; (3) otherwise, the function return is equivalent to the character specified as the third argument ('default'). In all cases, 'p' is advanced to the first character beyond the presumed mnemonic or single character.

Implementation

The mnemonic is transferred to an internal character buffer, then used in a binary search of a string table containing the ASCII mnemonics.

Arguments Modified

p

Calls

mapstr, strbsr

See Also

ctomn

Calling Information

```
subroutine move$ (from, to, count)
integer from (ARB), to (ARB), count
```

Library: vswtlb (standard Subsystem library)

Function

'Move$' is the fastest way known to move a block of words from one place to another in memory. The argument 'from' is an array of words to be moved; 'to' is an array to receive a copy of the words in 'from'; 'count' is the number of words to be moved.

Implementation

'Move$' is written in PMA, and uses multi-word load and store instructions to move as much data as possible during each iteration of a loop.

Arguments Modified

to

See Also

scopy (2)

Calling Information

```
integer function omatch (lin, i, pat, j)
character lin (ARB), pat (MAXPAT)
integer i, j
```

Library: vswtlb (standard Subsystem library)

Function

'Omatch' attempts to match a single pattern element at "pat(j)" against a character at "lin(i)". If the match succeeds, 'i' is incremented to point to the next unexamined character in 'lin'. The function return is YES if the pattern element matched the text, NO otherwise.

Implementation

'Omatch' is essentially a case statement, treating each pattern element specially. Non-special characters are directly compared. The wild-card character matches any non-NEWLINE character in 'lin'. Beginning-of-line is matched only when 'i' is one. End-of-line is matched only when "lin(i)" is a NEWLINE or an EOS. 'Locate' is used to match character classes. If a character is matched, 'i' is incremented by one.

Arguments Modified

i

Calls

* locate, error

See Also

match (2), amatch (2), locate (2)

Calling Information

```
file_des function open (path, mode[, typ[, limit]])
character path (ARB)
integer mode, typ, limit
```

Library: vswtlb (standard Subsystem library)

Function

'Open' is the primary means of opening files for reading, writing, etc. The first argument is the pathname of the file to be opened; it must be an EOS terminated string (e.g. open('/dir/file1's....). The second argument is the mode, READ, WRITE or READWRITE. The third argument is optional and should normally be omitted. It specifies the type associated with the file. The fourth argument is optional and should normally be omitted. It specifies the number of retries that should be made when a disk file is found to be in use. 'Open' returns a file descriptor which may be used for further i/o operations if the attempt to open was successful, or ERR if the attempt failed. The user is always left in the home directory after an attempt to open.

By default, 'open' returns a file descriptor to a sequential access method (SAM) file when referring to a disk file. If creating a direct access method file (DAM) is desired, the 'mode' argument may be ORed with the KNDAM file key (i.e., 'mode' can be "READWRITE+KNDAM" to create a DAM file opened for reading or writing). The constant KNDAM is contained in the "PRIMOS_KEYS" include file.

'Open' may be used to produce file descriptors that allow access to many different devices. Depending on the pathname, the file opened may be a standard input or output port, the user's terminal, a line printer spool file, a disk file, or the null device. Such special pathnames always begin with the characters "/dev/" followed by context-dependent strings that may specify names, options, etc. For example, the pathname used to open files in the spool queue may include any of the following options:

f	use Fortran forms control
r	use raw forms control
s	use standard forms control
h	inhibit header page
j	inhibit final page eject
n	number all output lines
a/location/	select destination printer
d/time/	defer printing until specified time
b/banner/	use given string on header page
c/copies/	print given number of copies
p/form/	select form type (wide, narrow, special, etc.)

Slashes or blanks may be used to separate parameters. For

example, "/dev/lps/f/agt.b/c10" refers to a spool file with Fortran forms control, ten copies of which will be printed on the printer with name "gt.b".

Implementation

'Open' first searches for an available file descriptor and, if found, performs initialization for the file. The form of the pathname given as the first argument controls subsequent actions, as follows:

/dev/stdout /dev/stdin /dev/errout /dev/errin /dev/stdout[123] /dev/stdin[123]	For standard port names, the allocated file descriptor is freed and the port descriptor is returned.
/dev/null	The null device is opened.
/dev/tty	The user's terminal is opened.
/dev/lps?*	'Lopen$' is called to open a disk file in the spool queue
otherwise,	If the pathname does not begin with "/dev/", 'dopen$' is called to open a disk file.

Arguments Modified

typ

Calls

dopen$, lopen$, getfd$, expand, mapdn, strbsr, Primos break$

See Also

dopen$ (6), lopen$ (6), close (2), create (2), remove (2)

Calling Information

```
integer function page(fdin, prompt, eprompt, lines, fdout, options)
file_des fdin, fdout
character prompt (ARB), eprompt (ARB)
integer lines, options
```

Library: vswtlb (Standard Subsystem Library)

Function

'Page' displays the contents of a disk file in paginated form. It also allows skipping pages forward and backward as well as searching for patterns within the file. 'Page' is primarily intended for viewing a file on a high speed CRT, but it may be used from any terminal.

'Page' accepts six arguments, of which the last is optional. 'Fdin' is the swt file descriptor of a file to be displayed. 'Prompt' specifies a format string (cf. 'print', 'encode') to be used for prompting the user after each screen of text except the final page. If this format string contains a format code for an integer (e.g. "*i") then 'page' replaces it with the current page number in the actual prompt. 'Eprompt' specifies a format string to be used for prompting the user when the final page of the file is reached; it may also contain a format code for the current page number. 'Lines' gives the number of lines in a page. 'Fdout' is the swt file descriptor of the file to receive the output display; 'page' only pages output when the output file is connected to a terminal (i.e. if the output file is on disk, 'page' simply copies the file to be displayed). The final (optional) argument consists of flags that control the operation of the 'page' subroutine. The following flags may be used singly or in combination (e.g. PG_END + PG_VTH):

PG_END	Do not prompt following the final page of the file. The default action is to prompt.
PG_VTH	Use 'vth' to manage the screen. By default 'page' displays the file without using 'vth'.

If the 'options' argument is not specified, it defaults to 0; 'page' displays the file using standard I/O and prompts after the last page of the file.

If 'vth' is used to display the paginated file, 'page' ignores the 'lines' argument and fixes the number of lines per page at the maximum number that can fit on the screen.

'Page' prompts the user after each page of output, and awaits one of the following commands (note that alphabetic commands may be entered in upper or lower case):

D<pages>	Display given number of pages (default 1), prompting only after the end of the range.

E<path>	Examine the file whose pathname is <path>.
E	Examine the original file.
H or ?	Print a command summary.
M<margin>	Set column of left margin to be displayed.
N or Q	Exit with OK status.
P or ^	Redisplay previous page.
S<lines>	Set page size to specified number of lines. Display starts over on page 1.
W <path>	Write a copy of the file being displayed to <path>. The file named <path> must not already exist.
W+<path>	Append a copy of the file being displayed to <path>.
W!<path>	Write a copy of the file being displayed to <path>. If the file already exists, it will be overwritten.
X	Exit with EOF status.
Y or :	Advance to the next page.
<ctrl-c>	Exit with EOF status (does not work in 'vth' mode).
<newline>	Advance to the next page.
<page>	Display specified page number.
-<pages>	Back up given number of pages (default 1).
.	Redisplay current page.
+<pages>	Advance given number of pages (default 1).
$	Display the last page.
/<pat>[/]	Display the next page containing <pat>.
\<pat>[\]	Display the previous page containing <pat>.

The pattern <pat> is a regular expression with the full set of options found in the editor. 'Page' searches circularly from the current file position for the next page that contains the specified pattern. As in the editor, the trailing delimiter is optional. (See *Introduction to the Software Tools Text Editor* in the *Software Tools Subsystem User's Guide* for details.)

Calls

close, ctoc, ctoi, encode, fcopy, getlin, isatty, makpat, markf, match, open,print, putch, seekf, scopy, strim, vtclr, vtenb, vtgetl, vtinfo, vtinit, vtprt, vtputl, vtread, vtstop, vtupd, Primos break$, missin, mklb$f, mkon$f, pl1$nl

Bugs

Large amounts of stack space are used.

If any format code other than "*i" is used in a format string, erroneous values will be displayed.

If more than one format code is specified, 'page' gets a pointer fault error.

There is no way to change the page alignment.

The "H" command output is not paged.

See Also

pg (1), *Introduction to the Software Tools Text Editor*

Calling Information

```
integer function parscl (str, buf)
character str (ARB), buf (MAXARGBUF)
```

Library: vswtlb (standard Subsystem library)

Function

'Parscl' is used to parse most standard Subsystem command line formats automatically. It examines the command line, parses it according to instructions present in its arguments, and makes the result available to the user for further processing. This processing is normally done with the aid of a set of standard Subsystem macros, described below. All arguments handled by 'parscl' are deleted from the command line, so any remaining special cases may be handled by the user.

The argument 'str' is a string describing the syntax of the command line. The argument 'buf' is a one-dimensional array of characters normally declared with the standard Subsystem macro 'ARG_DECL'. The function return is OK if the command line parsed successfully, ERR if an illegal option was seen or a required parameter was missing.

'Parscl' handles several types of arguments. "Flag" arguments are single-letter flags, preceded by a hyphen or dash, that have no parameters and may be grouped together in a single argument; for example, "-a" or "-acq". Arguments with parameters may have a string or integer value following the single-letter, or present in the next argument in the command line. For example, "-p1", "-p 1", "-nfilename", or "-n filename". Parameters for such arguments may be optional or required. Finally, some arguments may be ignored entirely, while others may not be allowable at all.

The argument 'str' contains a specification of allowable arguments and their types. Each specification consists of an option letter (case is ignored) followed by a type in angle brackets. The following types are allowable: 'f' or 'flag' for flag arguments, 'ign' or 'ignored' for ignorable arguments, 'na' for arguments that are not allowable, 'oi' or 'opt int' for arguments with an optional integer parameter, 'os' or 'opt str' for arguments with an optional string parameter, 'ri' or 'req int' for arguments with a required integer parameter, and 'rs' or 'req str' for arguments with a required string parameter. For example, a command with the syntax

```
-u <integer> [-l <integer>] [-i [<string>]]
```

would pass the following string to 'parscl':

```
u<req int> l<req int> i<opt str>
```

Order of arguments on the command line is unimportant, as well as the case of the option letter used.

The command line is typically parsed and then examined with a number of standard Subsystem macros. 'ARG_DECL' is used to declare the buffer required by 'parscl'. "PARSE_COMMAND_LINE(str,msg)" is used to invoke 'parscl'; 'str' is passed to 'parscl' as its first argument, and 'msg' is passed to 'error' to be printed if the command line could not be parsed. For example, one might use

```
PARSE_COMMAND_LINE ("u<ri>l<ri>i<os>"s,
   "Usage:  cmd -u<upper> [-l<lower>] [-i[<file>]]"p)
```

Once 'parscl' has been called in this manner, default values for optional parameters may be supplied with 'ARG_DEFAULT_INT' and 'ARG_DEFAULT_STR':

```
ARG_DEFAULT_STR(i,"/dev/stdin1"s)
ARG_DEFAULT_INT(l, 1)
```

One may test for the presence of an argument on the command line with 'ARG_PRESENT', and retrieve argument values with 'ARG_VALUE' and 'ARG_TEXT':

```
if (ARG_PRESENT (l))
   lower = ARG_VALUE (l)
else
   lower = 1
call ctoc (ARG_TEXT (i), filename, MAXLINE)
```

Once as much as possible of this kind of argument parsing is complete, the user may examine any remaining arguments by fetching them with 'getarg'.

Implementation

'Parscl' scans the specification string and builds a 26 element array. Each element of the array corresponds to a letter A - Z and contains an integer describing the type of argument expected when that letter is encountered. If an unrecognized argument type (in angle brackets) is encountered, 'parscl' calls 'error' to print an error message.

Then 'parscl' scans the command line arguments, skipping those that do not begin with a hyphen or have a letter as the second character. Arguments that begin with hyphens are examined further. If the letter in the second position of the argument is to be ignored, it is skipped. Flag arguments are simply marked "present" in the argument buffer. Values for string parameters are stored in the argument buffer for later retrieval. Values for integer parameters are converted with 'gctoi' (thus allowing arbitrary radix representation) then stored in the argument

buffer.

So that variables can be used in the macro calls, the following macros take an integer or variable containing an integer in the range 1 to 26 rather than a letter:

```
ARG_VALUE_I (<integer>)
ARG_PRESENT_I (<integer>)
ARG_DEFAULT_INT_I (<integer>, <string>)
ARG_DEFAULT_STR_I (<integer>, <string>)
```

Calls

ctoc, delarg, error, gctoi, getarg, mapdn, putlin, strbsr

Arguments Modified

buf

See Also

delarg (2), getarg (2), gfnarg (2)

Calling Information

```
integer function parsdt (str, i, month, day, year)
character str (ARB)
integer i, month, day, year
```

Library: vswtlb (standard Subsystem library)

Function

'Parsdt' examines the string passed to it in 'str', starting at position 'i' and attempts to interpret it as a Gregorian date. The string being examined is expected to be in any of three formats: a single integer, which is interpreted as a day in the current month of the current year; a pair of integers separated by a slash (/), which is interpreted as a month of the current year followed by a day within that month; or three integers separated by slashes, which is interpreted in the obvious way.

If the string is found to be a valid date (both syntactically and semantically), the arguments 'month', 'day' and 'year' are set appropriately, and OK is returned as the function value. Otherwise, the contents of 'month', 'day' and 'year' are unpredictable and ERR is returned as the function value. In all cases, the string index 'i' is advanced beyond the last character examined in the string.

Implementation

After skipping leading blanks and checking the first non-blank character to be sure it is a digit, 'parsdt' calls 'ctoi' to convert the string to an integer. As long as there are trailing slashes, 'ctoi' is called repeatedly until a month, day and year have been parsed. If at any point a trailing slash is not encountered, 'parsdt' calls 'date' to retrieve the current date and parses the remaining items from that string.

Calls

ctoi, date

Arguments Modified

i, month, day, year

See Also

parstm (2)

Calling Information

```
integer function parstm (str, i, val)
character str (ARB)
integer i
long_int val
```

Library: vswtlb (standard Subsystem library)

Function

'Parstm' converts a standard textual time-of-day representation into the number of seconds since midnight. The argument 'str' starting at position 'i' is assumed to be an EOS-terminated string containing the time-of-day in the format "<hours>[:<minutes>[:<seconds>]]['am'|'pm']". 'Val' is a long integer variable which receives the result of the conversion. The function return is OK if the conversion succeeded, ERR otherwise. As with most conversion routines, the position argument 'i' is updated to point to the first character in the input string that is not a part of the time-of-day.

Implementation

'Parstm' simply scans the string accumulating the components of the time as it goes, calculating 'val' in the process. Errors occur if there is no leading digit or if the time specified yields more than 86,400 seconds.

Arguments Modified

i

Calls

ctoi, mapdn

Bugs

Does not check the time string for legality. Behavior at midnight and noon may not be correct.

See Also

parsdt (2)

Calling Information

```
integer function patsiz (pat, n)
character pat (MAXPAT)
integer n
```

Library: vswtlb (standard Subsystem library)

Function

'Patsiz' returns the size of the pattern element at "pat(n)".

Implementation

Characters, start tags, and stop tags have size 2; beginning-of-line, end-of-line, and wild-card character have size 1; character classes have arbitrary size encoded at the next word in the pattern; closures have size CLOSIZE.

Calls

error

See Also

match (2), amatch (2)

Calling Information

```
longreal function powr$m (x, y)
longreal x, y
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'powr$m' function raises a double precision real value to a double precision real power. The function return is also double precision. It is the same as the Fortran statement "x**y".

The function is coded so as to adhere to ANSI Fortran standards which do not allow raising negative values to a floating point power, and which do not allow zero to be raised to a zero or negative power. Other inputs may trigger an error if the result of the calculation would result in overflow. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize.

There are four cases where this function may signal SWT_MATH_ERROR$. If an attempt is made to raise a negative value to a non-zero power, then the default return value will be the absolute value of that quantity raised to the given power. If an attempt is made to raise zero to a zero or negative power, the default return is zero. If the result would overflow then the default return value is the largest double precision quantity that can be represented. If the result would cause underflow, the default return is the smallest positive value which can be represented on the machine.

Implementation

Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

err$m (2),
SWT Math Library User's Guide

Calling Information

```
subroutine print (fd, fmt, a1, a2, ...)
file_des fd
character fmt (ARB)
untyped a1, a2, ...
```

Library: vswtlb (standard Subsystem library)

Function

'Print' is an output routine designed for ease of use. It allows the user to specify a file on which to write, a format to control output to the file, and any number of items to be printed. The first argument is the file descriptor of the file to be used for output. The second argument is a format string (discussed below). The remaining arguments (zero or more) are items to be output according to format control.

The format string is a EOS-terminated character string. It contains literal characters to be printed, as well as formatting control structures. Formatting control structures consist of an asterisk (*) followed by a single lowercase letter describing the action to be performed on the next argument in the argument list. For a complete list of the available formats, see the documentation for the subroutine 'encode'.

Characters in the format string that are not associated with a format control construct are output to the file without change.

A few examples may clarify the use of 'print'. The following call will print two real numbers along with some text for identification, followed by a NEWLINE, on standard output:

```
call print (STDOUT, "x = *r, y = *r*n"s, xcoord, ycoord)
```

This example shows how a line of output may be built up by successive calls:

```
call print (STDOUT, "absolute value = "s)
if (x < 0)
   call print (STDOUT, "*i*n"s, -i)
else
   call print (STDOUT, "*i*n"s, i)
```

Further examples of formats may be found in the documentation for 'encode'.

For compatibility with earlier versions of the Subsystem, packed strings will still be accepted, but all new code should use standard EOS-terminated strings.

Implementation

Since Fortran passes arguments to subroutines by reference, 'print' does not need to know the actual type of its printable arguments. A local character buffer is declared and passed along with the arguments to 'encode', which does the actual work of conversion. A call to 'putlin' then writes the result to the specified file.

Calls

encode, ptoc, putlin

Bugs

At most ten items may be printed.

See Also

encode (2), input (2), putlin (2), other conversion routines ('?*toc' and 'cto?*') (2)

Calling Information

```
integer function ptoc (pstr, term, str, len)
packed_char pstr (ARB)
integer len
character term, str (len)
```

Library: vswtlb (standard Subsystem library)

Function

'Ptoc' is used to convert packed character strings (e.g., Fortran Hollerith literals) into the EOS-terminated unpacked form normally used by all Subsystem routines. The argument 'pstr' is the packed array to be converted. 'Term' is a "termination character"; if the termination character appears unescaped in the packed string, then the unpacking operation will be terminated. (For example, most uses of packed strings in *Software Tools* included a period as a termination character, since in general there is no other way for a subprogram to tell where a Hollerith literal ends.) The argument 'str' is an array to receive the unpacked string; its maximum length is specified by the argument 'len'.

The function return is the length of the string in 'str' (as usual, excluding the EOS character).

A note on a rather common use of 'ptoc': Many Primos routines return packed character strings that do not have a termination character, but do have a maximum length. When using 'ptoc' to convert the output of these routines, one may use EOS as the termination character to obtain a fixed-length result.

Implementation

'Ptoc' uses the standard Subsystem macro 'fpchar' to pull successive characters from the packed array. These are simply copied into the receiving string until the string is full or an unescaped instance of the termination character is found.

Arguments Modified

str

See Also

other conversion routines ('cto?*' and '?*toc'), particularly 'ctop' (2), 'vtoc' (2), and 'ctov' (2)

Calling Information

```
integer function ptov (pstr, term, vstr, len)
packed_char pstr (ARB), vstr (len)
integer len
character term
```

Library: vswtlb (standard Subsystem library)

Function

'Ptov' converts a packed character string (e.g., Fortran Hollerith literals) into a PL/I-compatible "character varying" string. Character varying strings consist of a one-word length field, followed by up to 32767 words of packed character data.

The argument 'pstr' is the packed array to be converted. 'Term' specifies a "termination character"; if the termination character appears unescaped in the packed string, then 'ptov' terminates the copying operation without copying 'term'. (For example, most uses of packed strings in *Software Tools* included a period as a termination character, since in general there is no other way for a subprogram to tell where a Hollerith literal ends.) The argument 'vstr' is an array which receives the character-varying string; 'len' gives the number of words available in 'vstr', including the leading one-word length field.

The function returns the number of characters copied into 'vstr'.

Many Primos routines return packed character strings that do not have a termination character, but do have a maximum length. When using 'ptov' to convert the output of these routines, one may use EOS as the termination character to obtain a fixed-length result.

Implementation

'Ptov' first checks that 'len' is large enough to allow it to store characters in 'vstr'. If there is room for characters in 'vstr', 'ptov' fetches successive words from 'pstr', unpacks the characters, checks for escaped characters and 'term', and then packs the characters into 'vstr'. When it encounters 'term' or if it fills 'len' words with data, 'ptov' returns the number of characters copied.

Arguments Modified

vstr

See Also

other conversion routines ('pto?*' and '?*tov'), particularly 'vtop' (2), 'ctop' (2), 'ptoc' (2), 'vtoc' (2), and 'ctov' (2)

Calling Information

```
integer function putch (c, fd)
character c
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Putch' places the character 'c' on the file specified by file descriptor 'fd'. If the attempt succeeds, 'putch' returns OK; otherwise, it returns ERR.

Implementation

'Putch' creates an internal buffer of two characters, the first being the argument 'c' and the second being an EOS. This buffer is written on the specified file by a call to 'putlin'.

Calls

putlin

See Also

putlin (2), getch (2)

Calling Information

```
subroutine putdec (n, w, fd)
integer n, w
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Putdec' prints a decimal integer in a field of width greater than or equal to the argument 'w'. The argument 'n' is the integer to be printed; 'w' is the field width; 'fd' is the file descriptor of the file to be written. If 'w' is insufficient to print the integer, enough additional space on the file is used to insure that an accurate representation is printed.

Implementation

'Putdec' calls 'itoc' to convert the integer to a character representation. Enough blanks are output by calls to 'putch' to right justify the string produced by 'itoc', then the string itself is printed by multiple calls to 'putch'.

Calls

itoc, putch

See Also

itoc (2), encode (2), print (2)

Calling Information

```
integer function putlin (line, fd)
character line (ARB)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Putlin' is the primary Subsystem output routine. The character string supplied as the first argument is placed on the file specified by the second argument. The function return is OK if the write was successful, ERR otherwise.

Implementation

'Putlin' first calls 'mapsu' to map any standard unit which may have been supplied on the call to a file descriptor. The file specified by this descriptor is checked to insure writeability. If the last operation on the given file was not a 'putlin', 'flush$' is called to place the file's buffer in the empty state. Depending on the device type associated with the file, one of the device dependent drivers 'dputl$' (for disk files) or 'tputl$' (for terminal files) is called to perform the actual data transfer. No data transfer takes place for the null device (/dev/null). The function return is the value returned by the device dependent driver chosen, and is OK for a successful transfer, ERR for an unsuccessful transfer.

Calls

mapsu, dputl$, tputl$, flush$

See Also

mapsu (2), dputl$ (6), tputl$ (6), putch (2), getlin (2)

Calling Information

```
subroutine putlit (message, delimiter, fd)
packed_char message (ARB)
character delimiter
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Putlit' provides a way to place a literal string on a file. Its first argument is a packed character string, terminated by a character specified in the second argument. The third argument is the file descriptor of the file to be used.

'Putlit' is maintained for compatibility with earlier versions of the Subsystem. In the future, 'Putlin' should be used to write literal strings.

Implementation

'Putlit' calls 'ptoc' to unpack its first argument, then calls 'putlin' to print the unpacked string on the specified file.

Calls

ptoc, putlin

Bugs

Returns no status to indicate whether or not the write was successful.

See Also

ptoc (2), putlin (2), print (2), encode (2)

Calling Information

```
longreal function rand$m (l)
longint l
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'rand$m' function returns a double precision floating value in the open interval (0.0, 1.0). The argument to the function is set to a 32 bit integer in the range (0, 2**31 - 1). The generator is a linear congruential generator with multiplier 764261123. The values returned seem to be very well distributed, both from the standpoint of spectral tests and lattice tests.

The 'rand$m' routine does not detect or signal any errors. The first time the 'rand$m' function is called, if the generator has not been initialized with the 'seed$m' procedure, a seed is derived based on the current time of day and cpu utilization. This seed is returned in the integer argument variable.

This function can serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

Derived from information presented in "A Statistical Evaluation of Multiplicative Congruential Random Number Generators with Modulus 2^32 -1" by George S. Fishman and Louis R. Moore, in the Journal of the American Statistical Association, volume 77, number 377, March 1982.

Calls

dble$m, Primos timdat

See Also

dble$m (2), seed$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function readf (buf, nw, fd)
integer buf (ARB), nw
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Readf' is used to read words from a file, which may be assigned to any device recognized by the Subsystem. (A word on the Prime is 16 bits long.) The first argument is a buffer to receive data transferred from the file; the second argument is the number of words to be read; the third argument is the file descriptor of the file from which data will be read. Words are transferred from the file to the buffer until (1) the requested number of words are transferred, (2) end-of-file occurs, or (3) **if the file from which the data is read is a tty file**, a NEWLINE is encountered.

If the file is not readable, the given file designator is invalid, or the file's error flag is set, the function return is ERR. If end-of-file is encountered on the read, the function return is EOF. Otherwise, the function return is the number of words returned in the buffer.

Implementation

'Readf' first calls 'mapsu' to convert any standard port descriptors it is passed into Subsystem file descriptors. If the last operation performed on the file was not a 'readf', then 'flush$' is called to empty the file's Subsystem buffer. Depending on the device type associated with the file, a device dependent driver ('dread$' for disk or 'tread$' for the user's terminal) is called to do the actual work of getting data into the buffer.

Arguments Modified

buf

Calls

mapsu, dread$, tread$, flush$

Bugs

Strange things may happen if you ask for more words than 'buf' can hold. The semantics of reading raw characters

from a terminal are a little shaky; since one character per word is stored in a terminal buffer, 'readf' actually reads characters from a terminal, not words. There is a need for devices other than "terminal" and "disk" (system console, for example). EOF is returned if any error occurs when reading from disk (in dread$); the user is not informed of the actual error that occurs.

See Also

dread$ (6), tread$ (6), flush$ (6), mapsu (2), writef (2), getlin (2)

Calling Information

```
subroutine remark (message)
packed_char message (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Remark' is a routine from *Software Tools* that is used to print messages on the error output file (ERROUT). The single argument is either a packed, period-terminated character string (e.g. a Fortran Hollerith literal), or an unpacked, EOS-terminated string (the standard Subsystem variety). In either case, the given string is printed on ERROUT, followed by a NEWLINE.

Implementation

If the high-order byte of the first word of the string is non-zero, then the string must be packed; 'remark' uses 'putlit' to write the string to ERROUT. Otherwise, 'remark' uses 'putlin' to write the string. Finally, 'putch' is called to print the trailing NEWLINE.

Calls

putlit, putch, putlin

See Also

putlit (2), putch (2), error (2), putlin (2)

Calling Information

```
integer function remove (pathname)
character pathname (ARB)

Library:  vswtlb (standard Subsystem library)
```

Function

'Remove' deletes the file specified by the pathname given as the first argument. If the deletion could not be carried out, ERR is returned; otherwise, OK is returned.

Implementation

'Getto' is called to attach to the UFD containing the undesirable file. The file is deleted by a call to 'rmfil$', and a call to the Primos routine AT$HOM attaches the user back to his home directory. If any call to 'rmfil$' or 'getto' fails, ERR is returned; otherwise, OK is returned.

Calls

getto, Primos srch$$, Primos at$hom

See Also

getto (2), rmtemp (2), rmfil$ (6), del (1)

Calling Information

```
integer function rewind (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Rewind' positions the file specified by 'fd' to its beginning. All internal Subsystem status indicators are reset to indicate the new condition of the file. If the attempt to rewind was successful, 'rewind' returns OK; otherwise, it returns ERR.

Implementation

'Rewind' calls 'seekf' to set the current position of the file to zero.

Calls

seekf

Bugs

Terminal file behavior is somewhat unpredictable, since the user may have typed ahead of any requests for input.

See Also

wind (2), seekf (2), markf (2)

Calling Information

```
subroutine rmtabl (table)
pointer table
```

Library: vswtlb (standard Subsystem library)

Function

'Rmtabl' is used to remove a symbol table created by 'mktabl'. The sole argument is the address of a symbol table in dynamic storage space, as returned by 'mktabl'.

'Rmtabl' deletes each symbol still in the symbol table, so it is not necessary to empty a symbol table before deleting it (unless symbol table nodes contain a pointer to dynamic or linked-string storage, which cannot be reclaimed).

Please see the manual entry for 'dsinit' for instructions on initializing the dynamic storage space used by the symbol table routines.

Implementation

'Rmtabl' traverses each chain headed by the hash table created by 'mktabl'. Each symbol table node encountered along the way is returned to free storage by a call to 'dsfree'. Once all symbols are removed, the hash table itself is returned by a similar call.

Calls

dsfree

See Also

mktabl (2), enter (2), lookup (2), delete (2), dsget (2), dsfree (2), dsinit (2), sctabl (2)

Calling Information

```
integer function rmtemp (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Rmtemp' is used to remove a temporary file created by 'mktemp'. The file specified by 'fd' is rewound, truncated to zero length, and closed. This action is as close as possible to actually deleting the file. If the attempt to close the file is successful, 'rmtemp' returns OK; otherwise, it returns ERR.

Implementation

'Rmtemp' simply calls 'rewind', 'trunc', and 'close', in that order, on the given file descriptor. If the call to 'close' fails, ERR is returned; otherwise, OK is returned.

Calls

rewind, trunc, close

See Also

mktemp (2), rewind (2), trunc (2), close (2)

Calling Information

```
integer function rtoc (v, str, w, d)
real v
character str (ARB)
integer w, d
```

Library: vswtlb (standard Subsystem library)

Function

'Rtoc' converts the (single precision) real value in 'v' to a character string in 'str'. The length of the string is returned as the value of 'rtoc'.

The values of 'w' and 'd' control the format of the converted string. Generally speaking, 'd' controls the number of decimal positions or significant digits, and 'w' specifies the maximum length of the field. The following table explains the operation of 'rtoc' for different combinations of 'w' and 'd'. (Fortran and Basic programmers take note: d>12 corresponds to Basic output, 12>=d>=0 corresponds to Fortran 'F' format, and 0>d>=12 corresponds to Fortran 'E' format)

'd'	*'w'*	*Result*
d>12	w>16	If the value is in the range 1e7>v>=1e-2, it is converted into a BASIC-like fixed-point with no trailing zeroes after the decimal point. Otherwise, it is converted into a BASIC-like exponential format with no trailing zeroes after the decimal point.
	w<=16	An error is returned.
12>=d>=0	-	If possible, the value is converted to a fixed-point format with 'd' positions after the decimal point. Otherwise, it is converted to an exponential format with as many significant digits as possible. If 'w' is less than 8, an exponential conversion is not possible and an error will be returned.
0>d>-12	w>d+6	The number is converted to an exponential format with 'd' significant digits.
	w<=d+6	An error is returned.

To return an error, 'rtoc' places a string consisting of a single question mark in 'str'.

It should be noted that 'w' is roughly equivalent to the

'size' parameter in other conversion routines such as 'itoc' and 'ltoc'; 'w' specifies the maximum number of digits that may be produced. Thus the maximum number of characters returned in 'str' will never exceed 'w + 1'.

Implementation

'Rtoc' converts the number to double precision and then calls 'dtoc'. 'Rtoc' then returns whatever 'dtoc' returns.

Arguments Modified

str

Calls

dtoc

Bugs

Has been thoroughly tested, but has not stood the test of time.

See Also

other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function scopy (from, i, to, j)
character from (ARB), to (ARB)
integer i, j
```

Library: vswtlb (standard Subsystem library)

Function

'Scopy' copies a string from one place to another. The source string begins at the 'i'th character of 'from', and extends to an EOS; the destination string begins at the 'j'th character of the string 'to'. Copying takes place by character-by-character transfer until an EOS is encountered; the EOS is transferred to the receiving string also. When it finishes, 'scopy' returns the number of characters copied, excluding the trailing EOS.

Implementation

A simple loop copies characters from one string to the other, until an EOS is seen.

Arguments Modified

to

Calling Information

```
integer function sctabl (table, symbol, info, posn)
pointer table, posn
untyped info (ARB)
character symbol (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Sctabl' provides a means of accessing all symbols present in a symbol table (c.f. 'mktabl') without knowledge of the table's internal structure. After a simple initialization (see below), successive calls to 'sctabl' return symbols and their associated information. When the return value of 'sctabl' is EOF, the entire table has been scanned.

The first argument is the index in dynamic storage of the symbol table to be accessed. (This should be the value returned by 'mktabl'.)

The second and third arguments receive the character text of and integer information associated with the symbol currently under scan.

The fourth argument is used to keep track of the current position in the symbol table. It must be initialized to zero before 'sctabl' is called for the first time for a given scan. Furthermore, if the scan must be terminated early (before 'sctabl' returns EOF) the dynamic storage area pointed to by this argument must be freed, e.g. with "call dsfree (posn)".

The function return is EOF when the entire table has been scanned, and not EOF otherwise.

Implementation

If 'posn' is zero, 'sctabl' allocates two words of dynamic memory and assigns their location to it. These words are used to keep track of (1) the hash table bucket currently in use and (2) the position in the bucket's list of the next symbol. If a symbol is available in the current list, 'sctabl' returns its data and records the position of the next symbol in the list; otherwise, it moves to the next bucket and examines that list. If there are no more buckets in the symbol table, the position information pointed to by 'posn' is returned via a call to 'dsfree' and EOF is returned as the function value. Incidentally, 'posn' is set to zero when the end of the table is encountered.

Arguments Modified

symbol, info, posn

Calls

dsget, dsfree, scopy

Bugs

A call to 'enter' must be made to update the information associated with a symbol. If new symbols are entered or old symbols deleted during a scan, the results are unpredictable. The argument order is bogus; all the other symbol table routines have the table pointer as the last argument.

See Also

lookup (2), delete (2), mktabl (2), rmtabl (2), st$lu (6), dsget (2), dsfree (2), dsinit (2)

Calling Information

```
integer function sdrop (from, to, length)
character from (ARB), to (ARB)
integer length
```

Library: vswtlb (standard Subsystem library)

Function

'Sdrop' copies all but 'length' characters from the 'from' string into the 'to' string and returns as its result the number of characters copied. If 'length' is positive, the omitted characters are relative to the beginning of the 'from' string; if it is negative, they are relative to the end of the string.

Arguments Modified

to

Calls

ctoc, length, scopy

See Also

stake (2), index (2), substr (2), drop (1)

Calling Information

```
subroutine seed$m (u)
untyped u
```

Library: vswtmath (Subsystem mathematical library)

Function

The 'seed$m' procedure is used to reset the pseudo-random number generator to a known state. It is called with any 4 byte value which is not equal to 32 bits of zero. The seed can therefore be 4 characters, a long pointer, a long integer, or a real number. If the input is identical to zero then the SWT_MATH_ERROR$ condition is signalled. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. 'Seed$m' does not return a value.

Implementation

Based on the structure of the 'rand$m' routine; see the source for specific details.

Calls

Primos signl$

See Also

err$m (2), rand$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function seekf (pos, fd[, xra])
file_mark pos
file_des fd
integer xra
```

Library: vswtlb (standard Subsystem library)

Function

'Seekf' is used to position the file pointer to a designated word. The first argument is an integer value which specifies the relative or absolute positioning value (depending on the value of the third argument, 'xra'); if 'xra' equals ABS then positioning is from the beginning of the file, or if 'xra' equals REL then positioning is from the current position. The second argument is the file descriptor of the file whose file pointer is being manipulated. The third argument is optional. If omitted, the value ABS is assumed. The function return is OK if the positioning is successful, EOF if the end-of-the-file is reached, or ERR if 'fd' is an invalid file descriptor, if the error flag for the file is set, if the file device type is disk and 'xra' is ABS and 'pos' is negative, if the device type is terminal and 'xra' is ABS, or if the device type is terminal and 'pos' is negative.

Implementation

'Seekf' first calls 'mapsu' to map any standard port descriptor it may have been passed into a file descriptor for further processing. The Primos routine MISSIN is called to determine if the 'xra' argument is missing; if so, then absolute positioning is assumed. Depending on the device type associated with the file, a device dependent driver is called: 'dseek$' (for disk) or 'tseek$' (for terminal). The device dependent drivers do the actual work of positioning.

Calls

mapsu, dseek$, tseek$, flush$, Primos missin

Bugs

EOF is returned if any error occurs when reading from disk (in dseek$); the user is not informed of the true nature of the error.

Do not seek to end-of-file on a terminal file; all further input from the terminal will be ignored.

See Also

dseek$ (6), tseek$ (6), flush$ (6), mapsu (2)

Calling Information

```
subroutine seterr (stat)
integer stat
```

Library: vswtlb (standard Subsystem library)

Function

'Seterr' is used to set the error status code variable in the Subsystem common area. This variable is examined by the Shell when it regains control after the execution of a user program; if the value of the status code is greater than or equal to 1000, then the Shell assumes a fatal error has occurred and shuts down all currently active shell programs.

See Also

error (1), error (2)

Calling Information

```
integer function sfdata (key, pathname, infbuf, attach, aux)
integer key, attach
untyped infbuf, aux
character pathname (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

This functions sets information for a file system entry according to the value of the parameter 'key'. There are currently nine declarations for the values of 'key' in the standard SWT definitions. Their names and actions are:

FILE_UFDQUOTA -- This key sets quota information on a given directory. The object named in 'pathname' must be a directory. 'Infbuf' is a long integer value that is the maximum number of records that may be used under the given directory. If there already exist more records under the directory than this value, the quota is still set but no more may be used until enough records to reduce the number below this value are deleted. 'Aux' is not used.

FILE_DMBITS -- This key sets the dumped bit on the specified file object. 'Aux' is not used.

FILE_RWLOCK -- This key sets the read/write lock according to the string in 'infbuf'. 'Aux' is not used. Legal values for 'infbuf' are:

"n-1" - The lock is set to N readers or 1 writer.
"n+1" - The lock is set to N readers and 1 writer.
"n+n" - The lock is set to N readers and N writers.
"sys" - The lock is set to the current system default.

FILE_TIMMOD -- This key sets the modification date and time according to the values in 'infbuf'. 'Infbuf' is an array of 6 integers containing the year, month, day, hours, minutes, and seconds, respectively, to which to set the file modification date and time.

FILE_ACL -- This key sets the ACL attributes for the given file object. The attributes are set according to the strings contained in 'infbuf' and 'aux', which are the same format as used in 'sacl' and 'lacl'. If 'infbuf' and 'aux' are both empty (i.e. - contain only an EOS), 'pathname' is default protected if it is a normal file object or deleted if it is an access category. If 'infbuf' is empty and 'aux' is an existing file object, 'pathname' will be protected like 'aux' if 'aux' is a normal file object and protected by 'aux' if 'aux' is an access category. If 'aux' is

empty then 'pathname's protections will be modified according to 'infbuf'. If 'aux' and 'infbuf' are specified, the protections for the file 'aux' will be obtained and altered by the specifications in 'infbuf' and placed on the file 'pathname'. The file protections for 'aux' are not modified.

FILE_PRIORITYACL -- This key sets the priority ACL attributes for the partition named in 'infbuf'. If 'aux' is empty, the current priority ACL is deleted, otherwise the acl is set to the contents of 'aux'.

FILE_DELSWITCH -- This key controls the setting of the delete protect switch on the file 'pathname'. 'Infbuf' is an integer that contains YES to set the protection switch, or NO to turn it off.

FILE_PROTECTION -- This key controls the setting of the password protection bits according to the string in 'infbuf'. Protection bits are read, write, and truncate, represented by the characters 'r', 'w', and 't', respectively. In addition, the letter 'a' represents the string "trw" or all permissions. 'Infbuf' contains a string of owner permissions followed by a '/' and a string of non-owner permissions. The '/' may be omitted if no non-owner permissions are to be granted.

FILE_PASSWORDS -- This key sets the owner and non-owner passwords of the directory 'pathname' to the value of 'infbuf' and 'aux', respectively.

'Pathname' is any standard EOS-terminated pathname, and may contain templates. The function value is OK if the subroutine was successful and ERR otherwise. ERR may indicate the file does not exist, the caller does not have the necessary permissions, or any of the numerous file system errors possible.

The "attach" key is the same as in the 'getto' function; it indicates if the directory attach point had to be changed to get to the file entry being examined.

Implementation

The function resets the subsystem error code and expands the templates. If the directory attach point needs to be changed, 'getto' is called and the attach switch is saved. Then the information is setup and the appropriate Primos subroutines are called to make the file system changes.

Arguments Modified

attach

Calls

ctoc (2), ctop (2), ctov (2), equal (2), expand (2), follow (2), gtacl$ (6), index (2), mktr$ (2), mapstr (2), parsa$ (6), Primos ac$cat, Primos ac$dft, Primos ac$lik, Primos ac$set, Primos at$hom, Primos cat$dl, Primos pa$del, Primos pa$set, Primos q$set, Primos satr$$

See Also

chat (1), lacl (1), lf (1), sacl (1), gfdata (2)

Calling Information

```
integer function shell (fd)
file_des fd
```

Library: vshlib (shell routine library)

Function

'Shell' takes an open file descriptor as its only argument. The shell reads command lines from this file descriptor, and attempts to execute the commands.

This is the main routine for the user level shell as well. For details on how to use the shell, see *The User's Guide for the Software Tools Subsystem Command Interpreter*.

Having the shell as a subroutine opens up many possibilities not previously available to the programmer. However, care must be exercised when using the shell. In particular, since EPF's are not currently supported, it is quite possible for two user programs called from different invocations of the shell to destroy each other's code and/or data. For example, if you run 'se' on one file, and then from 'se' you run the shell, and from the new shell you run 'se' on a second file, the second 'se' will overwrite the data of the first 'se' (effectively destroying your first editing session). This is because the data for both instances of 'se' are loaded into the same area of (virtual) memory. There is currently no safe way to get around this, other than to be careful about what programs you run from subsidiary invocations of the shell. (The screen editor should be relatively safe from most programs (besides another 'se'), since its data is not loaded into the default segment (segment 4000), and the code is in a shared segment.)

When EPF's are supported, it is recommended that everything then be reloaded in EPF format. This will allow you to invoke programs from subsidiary shells without having to worry about what segment a program loads in. Until then, it is recommended that you do not use this subroutine for programs that will be loaded in segment 4000, since as soon as you run another external program which also loads in 4000, the first program will be destroyed. (When the second program exits, you will end up back in the lowest level of the shell.)

'Shell' returns OK if everything went well, otherwise it returns ERR.

Implementation

Too complicated to describe here.

Arguments Modified

None

See Also

The User's Guide for the Software Tools Subsystem Command Interpreter, sh (1), subsys (2)

Calling Information

```
longreal function sin$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function returns the sine of the angle whose measure (in radians) is given by the argument. The absolute value of the angle must be less than 26353588.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is implemented as a minimax polynomial approximation. Note that for angles sufficiently small the value of the sine function is equal to the measure of the angle. Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

asin$m (2), cos$m (2), dint$p (2), dsin$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function sinh$m (x)
real x

Library: vswtmath (Subsystem mathematical library)
```

Function

This routine calculates the hyperbolic sine of its argument, defined as sinh(x) = [exp(x) - exp(-x)]/2. The argument must be smaller than 22623.630826296. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The algorithm was adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m, Primos signl$

See Also

cosh$m (2), dexp$m (2), dsnh$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function sqrt$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This function calculates the square root of a floating point value. Attempts to take the square root of negative values will result in an error. The default return in this case will be the square root of the absolute value of the argument. All other arguments are in range and return valid results. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The algorithm involved is based on Newton's approximation method with an initial multiplicative approximation. The argument is scaled to within the range [0.5, 2.0) and then the algorithm is iterated to a solution. Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

Primos signl$

See Also

dsqt$m (2), err$m (2),
SWT Math Library User's Guide

Calling Information

```
integer function stake (from, to, length)
character from (ARB), to (ARB)
integer length
```

Library: vswtlb (standard Subsystem library)

Function

'Stake' copies the number of characters specified by 'length' from the 'from' string into the 'to' string and returns as its result the number of characters copied. If 'length' is positive, the characters are copied from the beginning of 'from'; if it is negative, they are copied from the end of 'from'.

Arguments Modified

to

Calls

ctoc, length, scopy

See Also

sdrop (2), index (2), substr (2), take (1)

Calling Information

```
integer function stclos (pat, j, lastj, lastcl)
character pat (MAXPAT)
integer j, lastj, lastcl
```

Library: vswtlb (standard Subsystem library)

Function

'Stclos' inserts a closure entry into a pattern being built by 'makpat'. This involves shuffling the last pattern entry far enough to allow a closure entry to be inserted, then linking the closure entry to the last closure entry in the pattern. 'Pat' is the pattern being built; 'j' is the current end of 'pat'; 'lastj' is the index in 'pat' of the last element inserted (the one that is controlled by the closure); 'lastcl' is the index of the last closure entry in 'pat'. The function return is equal to 'lastj', which is the index of the new closure after insertion is completed.

Implementation

A simple loop shuffles the last element down; several calls to 'addset' then create the closure entry and link it to the previous closure.

Arguments Modified

pat, j

Calls

addset

See Also

addset (2), makpat (2)

Calling Information

```
integer function strbsr (pos, tab, offs, object)
integer pos (ARB), offs
character tab (ARB), object (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Strbsr' is used to perform a binary search on a table created by the Ratfor 'string_table' declaration. The first argument is the position array, the second is the array of string text and additional information, the third is the offset of the string text in the 'tab' array (i.e., the number of words of additional data associated with each entry), and the last argument is a string containing the text to be sought.

The function return is the index of the element in the 'pos' array that indexes the appropriate entry in 'tab' if 'object' was found; EOF otherwise.

See the *User's Guide for the Ratfor Preprocessor* for a description of the 'string_table' declaration.

WARNING: the string table passed to 'strbsr' must be *sorted*, otherwise the binary search will fail.

Implementation

'Strbsr' is a straightforward binary search routine, using 'strcmp' to determine lexical ordering of strings.

Calls

strcmp

Bugs

Opaquely documented.

See Also

strlsr (2), *User's Guide for the Ratfor Preprocessor*

Calling Information

```
integer function strcmp (str1, str2)
character str1 (ARB), str2 (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Strcmp' is a generalized string comparison routine. The two arguments are EOS-terminated strings to be compared; the function return is 1 if 'str1' is less than 'str2' (according to the ordering of ASCII characters), 2 if 'str1' is equal to 'str2', and 3 if 'str1' is greater than 'str2'.

If one string is a proper initial substring of the other, the longer string is always found to be greater.

Implementation

Character-at-a-time comparison loop. Function return depends on which string hit EOS first, or on ASCII ordering of character codes.

See Also

equal (2), length (2)

Calling Information

```
integer function strim (str)
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Strim' is used to trim trailing blanks and tabs from the EOS-terminated passed as its first argument. The function return is the length of the trimmed string, excluding EOS.

Implementation

One pass is made through the string, and the position of the last non-blank, non-tab character remembered. When the entire string has been scanned, an EOS is planted immediately after the last non-blank.

Arguments Modified

str

See Also

stake (2), sdrop (2), substr (2)

Calling Information

```
integer function strlsr (pos, tab, offs, object)
integer pos (ARB), offs
character tab (ARB), object (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Strlsr' is used to perform a linear search on a table created by the Ratfor 'string_table' declaration. The first argument is the position array, the second is the array of string text and additional information, the third is the offset of the string text in the 'tab' array (i.e., the number of words of additional data associated with each entry), and the last argument is a string containing the text to be sought.

The function return is the index of the element in the 'pos' array that indexes the appropriate entry in 'tab' if 'object' was found; EOF otherwise.

See the *User's Guide for the Ratfor Preprocessor* for a description of the 'string_table' declaration.

Implementation

'Strlsr' is a straightforward linear search routine, using 'strcmp' to determine lexical equality of strings.

Calls

strcmp

Bugs

Opaquely documented.

See Also

strbsr (2), *User's Guide for the Ratfor Preprocessor*

Calling Information

```
integer function substr (from, to, first, length)
character from (ARB), to (ARB)
integer first, length
```

Library: vswtlb (standard Subsystem library)

Function

'Substr' copies the portion of the 'from' string specified by the 'first' and 'length' arguments into the 'to' string and returns the length of 'to' string as its result. 'First' specifies the starting character position in 'from'; if it is positive, it indicates a position relative to the beginning of the string, whereas if it is negative, the indicated position is relative to the end of the string. 'Length' specifies the number of characters to be copied; if positive, 'length' characters *starting* with the one selected by 'first' are copied; if negative, 'length' characters *ending* with the one selected by 'first' are copied. If the specified substring overlaps either the beginning or the end of 'from', 'to' will be shorter than 'length' characters.

Arguments Modified

to

Calls

length

See Also

stake (2), sdrop (2), strim (2), take (1), drop (1), substr (1)

Calling Information

```
integer function subsys (command)
character command (ARB)
```

Library: vshlib (shell routine library)

Function

'Subsys' takes an EOS terminated string as its only argument. It then passes this string on to the shell to be executed.

'Subsys' returns ERR if it could not put the command where the shell can get to it. Otherwise, it passes on the return code from the shell's execution of the command.

Implementation

'Subsys' first create a temporary file with 'mktemp'. It writes the command to be executed to the file, rewinds the open file descriptor, and then calls the 'shell' subroutine on that descriptor. Finally, it calls 'rmtemp' to close the temporary file.

Calls

mktemp, shell, rmtemp

See Also

mktemp (2), shell (2), rmtemp (2)

Calling Information

```
subroutine svdel (name)
character name (ARB)
```

Library: vshlib (shell routine library)

Function

'Svdel' deletes a shell variable at the current lexic level of the shell. 'Name' contains the name of the variable to delete. 'Svdel' cannot delete a variable global to the current scope. Changes to shell variables that control values in the SWT common block ("_eof" for example) occur immediately. If one of these values is deleted, the value in the common block reverts to the value it contained in the previous scope.

Implementation

If the variable does not exist at the current lexic level, the subroutine returns; otherwise, it deallocates the space in the internal variable common block for the name and value and then returns. If the variable is used to control a location in the SWT common block, the current value is replaced by the value it had when the current scope was invoked.

Arguments Modified

none

See Also

other sv?* routines (2)

Calling Information

```
subroutine svdump (fd)
file_des fd
```

Library: vshlib (shell routine library)

Function

'Svdump' outputs all internal shell variable information at all lexic levels on the open file descriptor 'fd'.

Implementation

'Svdump' scans the hash table for each lexic level from the first level to the current level and prints the hash chains (including the variable names, values, and locations) in the internal shell variable array.

Arguments Modified

none

Calls

print

See Also

other sv?* routines (2)

Calling Information

```
integer function svget (name, value, maxval)
character name (ARB), value (maxval)
integer maxval
```

Library: vshlib (shell routine library)

Function

'Svget' looks up and returns the value of the most recent declaration of the shell variable 'name'. 'Value' is the array to receive the value and 'maxval' is the maximum amount of space (including the EOS) in the receiving string. The function returns the length of the returned string 'value' if the variable is found and EOF otherwise.

Implementation

'Svget' searches for 'name' from the current lexic level back to the first lexic level, stopping when it locates the first (most recent) definition. Any previous declarations are ignored. If the variable is not located then the function returns EOF; otherwise, as much of the value as possible is copied into the receiving buffer and the number of characters transferred is returned.

Arguments Modified

value

Calls

ctoc

Bugs

Should probably return the lexic level of the variable located.

See Also

other sv?* routines (2)

Calling Information

```
integer function svlevl (level)
integer level
```

Library: vshlib (shell routine library)

Function

'Svlevl' returns the current lexic level of the shell in 'level'. The function return is also the lexic level.

Implementation

The lexic level information is retrieved from the internal shell variable common block and returned. The value will be in the range from one to some maximum value (currently ten).

Arguments Modified

level

See Also

other sv?* routines (2)

svmake (2) --- create a shell variable at the current lexic level 05/27/82

Calling Information

```
integer function svmake (name, value)
character name (ARB), value (ARB)
```

Library: vshlib (shell routine library)

Function

'Svmake' creates a shell variable 'name' at the current lexic level of the shell with the value 'value'. The function returns the lexic level at which the variable has been created. If the variable controls a value kept in the SWT common block, the value in the common block is updated to reflect the new value of the variable.

Implementation

First, 'svmake' checks the existence of the variable at the current lexic level. If it exists, then the function returns immediately; otherwise it allocates space in the variable area for the name and value. If the variable controls a location in the SWT common block, 'svmake' saves the current value in the SWT common and copies the new value in its place.

Arguments Modified

none

Calls

length, ctoc

See Also

other sv?* routines (2)

Calling Information

```
integer function svput (name, value)
character name (ARB), value (ARB)

Library:  vshlib (shell routine library)
```

Function

'Svput' sets the value of existing shell variable 'name' or creates a new variable with the specified value at the current lexic level if 'name' does not already exist. The function returns the lexic level of the variable that was set. If the variable controls a value kept in the SWT common block, 'svput' updates the value in the common block to reflect the new value of the variable.

Implementation

If the variable exists at any lexic level, 'svput' replaces the previous value. If the variable does not exist, 'svput' calls 'svmake' to create the variable at the current lexic level. If the variable controls a location in the SWT common block, 'svput' saves the current value in the SWT common and copies the new value in its place.

Arguments Modified

none

Calls

svmake

See Also

other sv?* routines (2)

Calling Information

```
integer function svrest (file, trace)
character file (ARB)
bool trace
```

Library: vshlib (shell routine library)

Function

'Svrest' takes a file written by 'svsave' and attempts to merge the variables in the file with those on the current lexic level. Variables already in existence at the current level will not be replaced. 'File' is the name of the file containing the 'svsave'd variables. If 'trace' is set, 'svrest' produces a trace of the restoration consisting of each variable followed by its value printed on the terminal. The function returns ERR if the file cannot be read or if it is misformatted; otherwise, the function returns OK.

Implementation

If the file cannot be opened then 'svrest' returns ERR, otherwise it reads pairs of lines containing the names and values of the variables. For each pair of the lines it calls 'svmake' to merge the variables with the existing ones. If it reads a name without a corresponding value, it closes the file and returns ERR.

Arguments Modified

none

Calls

close, open, print, svmake

See Also

other sv?* routines (2)

Calling Information

```
integer function svsave (file, trace)
character file (ARB)
bool trace
```

Library: vshlib (shell routine library)

Function

'Svsave' takes the shell variables at lexic level 1 and writes them to 'file'. Setting 'trace' produces a trace of the variables being saved on the users terminal. The trace consists of the name of each variable being saved followed by its value. The function returns ERR if the file could not be opened and OK otherwise.

Implementation

If the file can't be opened, then the function returns an error; otherwise, the current level of shell variables is traversed and written to the file.

Arguments Modified

none

Calls

close, open, print, putch, putlin, trunc

See Also

other sv?* routines (2)

Calling Information

```
integer function svscan (name, maxlen, info [, offset] )
character name (maxlen)
integer maxlen, info (3), offset
```

Library: vshlib (shell routine library)

Function

'Svscan' provides the user with a way of retrieving a list of the shell variables that are currently declared. Each call to 'svscan' returns one variable name. The first and second arguments are the returned name and the maximum length (including the EOS) that the name can attain. The third argument is a three word array that 'svscan' uses to keep track of its position in the internal shell variable data structure. The user should set the first element of this array to zero before the first call to 'svscan' and afterwards should leave it alone. The last argument is an optional offset from the current lexic level of the shell at which to scan for the shell variables. If 'offset' is omitted, 'svscan' scans the current level. The function returns the length of the returned shell variable name, or EOF if all variables have been returned. The user should not make any subroutine calls that will change any shell variables between the first call to 'svscan' and the final one. Doing so may cause duplicate names to be returned or may cause some names to be skipped.

Implementation

If the first element of the information array is 0, 'svscan' initializes the rest of the array. Otherwise it checks information in the array 'info' for validity and then scans the variable data structures for the next shell variable starting at the previous position. If all shell variables have already been returned, 'svscan' returns EOF, otherwise it copies as much of the variable name as possible to the user's receiving buffer and returns the number of characters copied as the function return.

Arguments Modified

name, info

Calls

ctoc

See Also

other sv?* routines (2)

Calling Information

subroutine swt

Library: vswtlb (standard Subsystem library)

Function

'Swt' is called by all Subsystem programs to return to the Subsystem command interpreter.

Implementation

'Swt' calls 'rtn$$' to return to the Subsystem command interpreter.

Calls

rtn$$

See Also

call$$ (6)

Calling Information

```
integer function sys$$ (cmd, cominput)
character cmd (ARB)
file_des cominput
```

Library: vswtlb (standard Subsystem library)

Function

'Sys$$' passes the Primos command in 'cmd' to the Primos shell with a call to the Primos routine CP$. The second argument 'cominput' specifies the file unit from which the command takes its input. If no change in command input is desired, the argument should be ERR.

The function return is ERR if the status returned by CP$ is greater than zero (a fatal error), and OK otherwise.

Implementation

'Sys$$' converts the command to a varying character string with a call to 'ctov'. If 'cominput' isn't the value ERR, the command input is switched to that file; otherwise, the command is just executed, with no change being made as to where the command input is coming from. After making a call to the Primos routine MKONU$ to create an on-unit for the Primos REENTER$ condition, it calls CP$ to process the Primos command.

Calls

ctov, flush$, mapfd, mapsu, Primos break$, Primos comi$$, Primos cp$, Primos mkonu$

Bugs

If the user's program is loaded in segment 4000, then only Primos internal commands may be executed with 'sys$$'. External commands will destroy the current memory image, and may destroy the user's current Primos environment, requiring that the user reset it, using the Primos command "RLS -ALL".

When Primos supports EPFs, this restriction will be lifted (on programs loaded with 'bind').

See Also

ldtmp$ (6)

Calling Information

```
longreal function tan$m (x)
real x

Library: vswtmath (Subsystem mathematical library)
```

Function

This function calculates the tangent of the angle whose measure is given (in radians) as the argument to the function. The arguments must have an absolute value of less than 13176794.0. The condition SWT_MATH_ERROR$ is signalled if there is an argument error. An on-unit can be established to deal with this error; the SWT Math Library contains a default handler named 'err$m' which the user may utilize. If an error is signalled, the default return value will be zero.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

The function is calculated based on a minimax polynomial approximation over a reduced argument. Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dint$p, Primos signl$

See Also

cot$m (2), dint$p (2), dtan$m (2), err$m (2), *SWT Math Library User's Guide*

Calling Information

```
longreal function tanh$m (x)
real x
```

Library: vswtmath (Subsystem mathematical library)

Function

This routine calculates the hyperbolic tangent of its argument, defined as tanh(x) = 2/[exp(2x) + 1]. The function never signals an error and returns valid results for all inputs.

This function is intended to serve as a single precision function although it returns a double precision result. The function has been coded so that any value returned will not overflow or underflow a single precision floating point value. The double precision register overlaps the single precision register so it is possible to declare and use this function as simply a "real" function.

Implementation

Adapted from the algorithm given in the book *Software Manual for the Elementary Functions* by William Waite and William Cody, Jr. (Prentice-Hall, 1980).

Calls

dexp$m

See Also

dexp$m (2), dtnh$m (2),
SWT Math Library User's Guide

Calling Information

```
logical function tquit$ (flag)
logical flag
```

Library: vswtlb (standard Subsystem library)

Function

'Tquit$' checks to see if there is a pending program interrupt as a result of the user having entered a BREAK or CTRL-P. If there is, a value of .true. is returned in 'flag' and as the function's result; otherwise, a value of .false. is returned.

Before 'tquit$' can be used, the Primos system call BREAK$ must have been called with an argument of .true.. Before a program exits, BREAK$ should be called with an argument of .false..

Implementation

'Tquit$' calls the Primos routine QUIT$ to detect the interrupt. If one has occurred, it also calls the Primos routine TTY$RS to clear the terminal output buffer and T1OU to echo a NEWLINE.

Arguments Modified

flag

Calls

Primos quit$, Primos tty$rs, Primos t1ou

Calling Information

```
integer function trunc (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Trunc' is used to truncate writeable files at their current position; that is, to delete the remainder of the file. The argument is the file descriptor of the file to be truncated; the function return is OK if the truncation was successful, ERR otherwise.

Implementation

'Flush$' is called to empty the buffers associated with the given file. If the file to be truncated is a terminal file or the null device, then an immediate successful return is taken. If the file to be truncated is a disk file, it is truncated by the Primos routine PRWF$$. If the return from PRWF$$ is good, 'trunc' returns OK; if not, 'trunc' returns ERR.

Calls

flush$, mapsu, Primos prwf$$

Bugs

Behavior on terminal files is somewhat questionable.

See Also

remove (2), rmtemp (2)

Calling Information

```
character function type (c)
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Type' returns the type of the character given as its first argument: LETTER if the character was a letter, DIGIT if the character was a digit, and the character itself if it was anything else.

Implementation

'Type' checks the type of character by using a Ratfor 'select' statement listing all the letters in one alternative and all the digits in another. If the character falls within the first range, LETTER is returned; if it falls within the last range, DIGIT is returned; if it is outside of both, the function return is the character itself.

Calling Information

```
integer function vfyusr (lognam)
character lognam (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vfyusr' is used to verify that a given login name corresponds to an authorized user of the Subsystem. The single argument is the login name of the user to be checked. The function return is OK if the given name was validated, ERR otherwise.

Implementation

'Vfyusr' opens the Subsystem user list file "=userlist=" (nominally in "//extra/users") and simply reads it until the given user name is found or EOF is encountered.

Calls

close, ctoc, getlin, length, mapstr, open, remark, strcmp

Calling Information

```
subroutine vtbaud (rate)
integer rate
```

Library: vswtlb (standard Subsystem library)

Function

'Vtbaud' is used to set the terminal baud rate for other VTH routines. 'Rate' can be from 50 to 19200. A number lower than 50 will be set to 50 and one higher than 19200 will be set to 19200. This value is used to determine the delay times for special functions such as screen clearing and cursor positioning.

Implementation

After truncating 'rate' to the boundary conditions, the value is saved in the SWT common blocks for later use.

See Also

other vt?* routines (2)

Calling Information

```
subroutine vtclr (srow, scol, erow, ecol)
integer srow, scol, erow, ecol
```

Library: vswtlb (standard Subsystem library)

Function

'Vtclr' is used to clear a rectangle on the users terminal. The arguments are the starting row 'srow', starting column 'scol', ending row 'erow', and ending column 'ecol'.

Implementation

After boundaries are checked and truncated (to 1 for values less than 1, and MAXCOL and MAXROW for values greater than their respective dimension) a small loop simply writes sequences of blanks on the screen using 'vt$put'.

Calls

vt$put

See Also

other vt?* routines (2)

Calling Information

```
integer function vtdlin (row, cnt)
integer row, cnt
```

Library: vswtlb (standard Subsystem library)

Function

'Vtdlin' deletes 'cnt' lines starting at line 'row' on the screen. If 'cnt' is not given, it defaults to 1. Unlike other 'vth' functions, 'vtdlin' makes the changes on the user's terminal immediately (ie - with no call to 'vtupd'). 'Vtdlin' will take advantage of a terminal's hardware delete line function, if one is available, otherwise it will simulate it with whatever other functions the terminal possesses. The function return is ERR if 'row' is off the screen or 'cnt' is negative and OK otherwise.

Implementation

'Vtdlin' first ensures that 'row' is on the screen and that 'cnt' is positive. If a hardware delete line function is available, the subroutine simply positions to the correct place on the screen and outputs the appropriate number of line deletes. If hardware delete is not available, the subroutine redraws the appropriate sections and attempts to use a hardware clear to end-of-line function to clear the bottom sections of the screen. If no hardware clear to end-of-line is available, the subroutine just redraws the screen using blanks to clear the correct sections.

Calls

move$, vt$del, vt$out, vtmove

Arguments Modified

none

See Also

Other vt?* routines (2)

Calling Information

```
subroutine vtenb (row, column, length)
integer row, column, length
```

Library: vswtlb (standard Subsystem library)

Function

'Vtenb' enables input in a field with a particular length, starting at the given (row, column) on the screen. Any areas of the screen that are not enabled by 'vtenb' cannot be used for entry of data; therefore 'vtenb' must be called before 'vtread' can be used.

Implementation

'Row' is checked as being on the screen, and if not, an immediate return is executed and input is not enabled at that location. 'Column' and 'length' are checked as being on the screen, and if the values specified "run off" the screen, the length is truncated to the border of the screen. Input is then enabled starting at (row, column) for 'length' characters, or to the border of the screen.

Bugs

Allows only one input area per line.

See Also

other vt?* routines (2)

Calling Information

```
integer function vtgetl (str, row, column, length)
character str (ARB)
integer row, column, length
```

Library: vswtlb (standard Subsystem library)

Function

'Vtgetl' transfers data from the internal screen buffer to a string supplied by the user. 'Row' and 'column' locate the starting position of the input field on the screen, and the argument 'length' specifies its length. The function return is the actual length of the of the string returned in 'str'. Note that 'vtgetl' doesn't actually perform a read; it simply returns what is in the internal screen buffer. 'Vtread' must be called beforehand to allow the user to enter data.

Implementation

A check is made to see that the 'row' argument is within bounds, and if not, the string returned is EOS and the length returned is 0. If the 'column' and/or 'length' arguments cause a request that is off the screen, the string is truncated to the edge of the screen buffer. Then a loop simply retrieves characters from the screen buffer and places them in 'str', and the length of the retrieved string returned.

Arguments Modified

str

See Also

vtread (2), and other vt?* routines (2)

Calling Information

```
integer function vtilin (row, cnt)
integer row, cnt
```

Library: vswtlb (standard Subsystem library)

Function

'Vtilin' inserts 'cnt' blank lines at line 'row' on the screen. If 'cnt' is not given, it defaults to 1. Unlike other 'vth' functions, 'vtilin' makes the changes on the user's terminal immediately (ie - with no call to 'vtupd'). 'Vtilin' will take advantage of a terminal's hardware insert line function, if one is available, otherwise it will simulate it with whatever other functions the terminal possesses. The function return is ERR if 'row' is off the screen or 'cnt' is negative and OK otherwise.

Implementation

'Vtilin' first ensures that 'row' is on the screen and that 'cnt' is positive. If a hardware insert line function is available, the subroutine simply positions to the correct place on the screen and outputs the appropriate number of line inserts. If hardware insert is not available the subroutine attempts to use a hardware clear to end-of-line function to clear sections of the screen and then redraw the rest of the screen. If no hardware clear to end-of-line is available the subroutine just writes blanks to clear the correct section and then redraws the rest of the screen.

Calls

move$, vt$del, vt$out, vtmove

Arguments Modified

none

See Also

Other vt?* routines (2)

Calling Information

```
integer function vtinfo (key, info)
integer key, info (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtinfo' is used to return certain needed information from the VTH common blocks. The first argument is a key to tell 'vtinfo' what set of information to return. The second argument is an array to return the information. The function return is OK if a correct key is given, and ERR otherwise.

'Vtinfo' currently supports the following value(s) for 'key':

VT_MAXRC — returns the maximum row and col values for the user's terminal in the first two words of 'info'.

VT_WRAP — returns YES in 'info' if the terminal wraps to the next line after putting a character in the last column and NO otherwise.

VT_HWINS — returns YES in 'info' if the terminal has hardware insert capabilities.

VT_HWDEL — returns YES in 'info' if the terminal has hardware delete capabilities.

VT_HWCEL — returns YES in 'info' if the terminal has hardware clear to end-of-line capabilities.

Implementation

The key is checked as being legal, and then the requested information is simply copied from the VTH common blocks.

Arguments Modified

info

See Also

other vt?* routines (2)

Calling Information

```
integer function vtinit (term_type)
character term_type (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtinit' initializes the terminal characteristic common blocks for the virtual terminal handler. It must be called before any of the other routines are used. The single argument is the returned terminal type of the users current process. The value returned from 'vtinit' is OK if the descriptor file for that type of terminal is found, and is in the correct format, and ERR otherwise.

Implementation

'Vtinit' first calls 'vtterm' to initialize the terminal characteristic tables and return the terminal type. If 'vtterm' couldn't initialize the tables, then 'vtinit' returns ERR. 'Vtinit' then proceeds to clear the screen buffers, status information, input enabling, and turns off terminal echo, and then returns OK.

Arguments Modified

term_type

Calls

vtterm, Primos duplx$

See Also

other vt?* routines (2)

Calling Information

```
subroutine vtmove (row, col)
integer row, col
```

Library: vswtlb (standard Subsystem library)

Function

'Vtmove' moves the cursor on the terminal to position 'row', 'col' with the least cost. 'Vtinit' should have been called beforehand to set up the terminal characteristics in the virtual terminal handler. If the coordinates given are off of the screen, no positioning will be done.

Implementation

'Vtmove' first checks if relative movement would be faster, and if so, relatively positions the cursor, otherwise it calls 'vt$pos' to absolutely position the cursor.

Calls

vtpos, vtout

See Also

other vt?* routines (2)

Calling Information

```
subroutine vtmsg (msg, type)
character msg (MAXLINE)
integer type
```

Library: vswtlb (standard Subsystem library)

Function

'Vtmsg' is used to place an arbitrary message in the "status line" (if one has been enabled). If there has been no status line enabled, 'vtmsg' has no effect. Messages are 'typed' with simple integers; each new message overwrites any old one with the same 'type'. Messages with different types are simply shuffled to different places on the status line.

Implementation

'Vtmsg' first checks to see if the status line has been enabled and, if not, simply returns. The status line is then scanned for another message with the same type. If one is found, it checks to see if the new message will fit in place of the old one, and if not, if proceeds to shuffle the existing messages around to attempt to fit them on. If another message is not found with the same type, it just looks for enough space on the status line to place the new message, shuffling the others around, if necessary. If there isn't enough space in which to place the message, as much of it as is possible is placed on the status line.

Calls

length, vt$put

See Also

length (2), and other vt?* routines (2)

Calling Information

```
integer function vtoc (var, str, len)
integer var (ARB), len
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtoc' is used to convert a PL/I character-varying string into a standard Subsystem EOS-terminated string. The first argument is the character-varying string to be converted; the second is a string to receive the result; the third is the maximum length of the result string. The function return is the number of characters in the result string after the conversion.

Implementation

'Vtoc' uses the standard Subsystem macro 'fpchar' to pull characters from the PL/I string one at a time, and place them in the result string. Conversion stops when the result string fills or when all the characters in the PL/I string have been moved.

Arguments Modified

str

See Also

other conversion routines ('cto?*' and '?*toc') (2)

Calling Information

```
integer function vtop (vstr, pstr, len)
packed_char vstr (ARB), pstr (len)
integer len
```

Library: vswtlb (standard Subsystem library)

Function

'Vtop' converts a PL/I-compatible "character varying" string into a packed character string. Character varying strings consist of a one-word length field, followed by up to 32767 words of packed character data.

The argument 'vstr' is the character-varying string to be converted. 'Pstr' is an array which receives the packed string; 'len' gives the number of words available in 'pstr'.

The function returns the number of characters copied into 'pstr'.

Implementation

'Vtop' first checks that 'len' is large enough to allow it to store characters in 'str' and then computes the number of characters it can copy. If there is room for characters in 'pstr', 'vtop' copies successive words from 'vstr' into 'pstr' until it fills 'len' words or runs out of characters in 'vstr'. If 'vstr' contains an odd number of characters, 'vtop' pads the last word with 0's (an EOS character).

Arguments Modified

pstr

See Also

other conversion routines ('pto?*' and '?*tov'), particularly 'ptov' (2), 'ctop' (2), 'ptoc' (2), 'vtoc' (2), and 'ctov' (2)

Calling Information

```
subroutine vtopt (option, str)
integer option, str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtopt' sets a number of optional parameters for the VTH screen. The currently available values for 'option' are:

STATUS_ROW enable a "status row"; 'str' should be the row number on which the "status row" is to be displayed.

DISPLAY_TIME enable/disable the time display in the "status row"; 'str' should be YES to enable the display and it should be NO to disable the display.

UNPRINTABLE_CHARS display a printable representation of normally unprintable characters; 'str (1)' should contain the replacement character (as a character variable).

SET_TABS set tab stops for input; 'str' should contain an EOS string containing non-blank characters where tab stops are to be set.

Implementation

'Vtopt' simply sets certain variables in the VTH common block to allow the rest of the routines to keep up with the attributes. For STATUS_ROW, the "status row" is written out and a flag indicating that there is a status row, is set. For the rest of the options, though, flags are set to indicate each option.

Calls

vt$put

See Also

other vt?* routines (2)

Calling Information

```
subroutine vtpad (len)
integer len
```

Library: vswtlb (standard Subsystem library)

Function

'Vtpad' simply clears the rest of a field with blanks, starting at the current cursor position. The single argument required is the length of the field to clear, from the current cursor position.

Implementation

'Vtpad' simply checks how much more room is on a line with respect to the length to pad and the current cursor position and then stores blanks into the new screen, making sure that padding doesn't go past the end of the screen.

Calls

vt$put

See Also

other vt?* routines (2)

Calling Information

```
integer function vtprt (row, col, fmt, a1, a2, ... , a10)
integer row, col
character fmt (ARB)
untyped a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

Library: vswtlb (standard Subsystem library)

Function

'Vtprt' is used to place formatted strings into the screen buffers. 'Row' and 'col' tell where the resulting string is to be placed. 'Fmt' is a formatting string like that used in encode to define how the final string is to look. It can be either an EOS terminated string, or a packed string. The remaining arguments are the items to be put on the screen according to formatting control.

'Vtprt' works equivalent to the Subsystem subroutine 'print'. The user is advised to look at the documentation for 'print' for a full explanation of the formatting capabilities of 'vtprt'.

Implementation

'Vtprt' checks for the legality of the position (row and col), and returns ERR if it isn't legal. It then checks the string for being packed, or unpacked. If it is packed, 'ptoc' is called to unpack the character string. 'Encode' is then called to do the formatting, and then 'vt$put' is called to place the string in the screen buffer. The size of the resulting string is the function return.

Calls

ptoc, encode, vt$put

See Also

encode (2), print (2), ptoc (2), other vt?* routines (2)

Calling Information

```
subroutine vtputl (str, row, col)
integer row, col
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtputl' is used to place a string of characters into the screen buffer, in the specified position. The first argument is the EOS-terminated string of characters to be displayed; the second and third arguments are the (row, column) position on the screen where the first character of the string is to be displayed. 'Vtputl' only places the string into the screen buffer; 'vtupd' must be called before any changes to the internal buffer are reflected on the screen.

Implementation

'Vtputl' simply calls 'vt$put' with the same arguments, plus the length of the string, and then returns.

Calls

vt$put, length

See Also

length (2), and other vt?* routines (2)

Calling Information

```
integer function vtread (crow, ccol, clr)
integer crow, ccol, clr
```

Library: vswtlb (standard Subsystem library)

Function

'Vtread' starts reading characters from the user's terminal into the screen buffers. 'Vtenb' must be called before 'vtread' to enable input areas. 'Crow' and 'ccol' are the places at which to start reading. 'Clr' is a flag to let 'vtread' know if the user wants the input areas cleared before reading. If 'clr' is YES, then the input areas are cleared before reading, otherwise they are left as they are.

Implementation

'Clr' is checked to decide whether or not to clear the input areas, and if so, proceeds to call 'vt$put' to place blanks in these areas, and calls 'vtupd' to update these changes. It then positions to the input area at the given row and column. If there is no input area defined there, it positions to the next one defined. If there are no input areas defined, the function return is set to zero and 'vtread' returns. If an input area has been defined, it calls 'vt$get' to read characters from the terminal and place them on the screen, until a termination character is typed (RETURN, KILL_RIGHT_AND_RETURN, MOVE_UP, MOVE_DOWN) and then returns the termination code as the function return.

Calls

vtput, vtget, vtupd

Arguments Modified

none

See Also

Introduction to the Software Tools Text Editor (Se section), and other vt?* routines (2)

Calling Information

```
subroutine vtstop
```

Library: vswtlb (standard Subsystem library)

Function

'Vtstop' resets terminal attributes when terminating the VTH portion of a program.

Implementation

'Vtstop' first positions to the first column of the last row on the user's terminal. 'Vtstop' then retrieves the previous terminal attributes from the VTH common block (where they are saved by 'vtinit') and restores the attributes by a call to the Primos routine DUPLX$.

Calls

vtmove, Primos duplx$

See Also

other vt?* routines (2)

Calling Information

```
integer function vtterm (term_type)
character term_type (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Vtterm' is used to read in the characteristics for a terminal and put that information into the common blocks used by the other VTH routines. It obtains the user's terminal type and attempts to open the terminal characteristic file. If it succeeds, the function return is OK, otherwise ERR. The user should not call this routine himself, but should use the routine 'vtinit' as the interface into this routine.

Implementation

'Vtterm' calls 'gttype' to return the user's terminal type. It attempts to open the file "=vth=/<term_type>" for reading, and if it can't, it returns ERR. Otherwise, it reads the file, decodes the symbolic characteristics, places them in the VTH common block, and then returns OK.

Arguments Modified

term_type

Calls

close, ctoc, ctoi, encode, equal, getlin, gtattr, gttype, length, mntoc, open, strbsr, vtalc, vtier

See Also

other vt?* routines (2)

Calling Information

```
subroutine vtupd (clr)
integer clr
```

Library: vswtlb (standard Subsystem library)

Function

'Vtupd' is used to update the changes from the last (old) VTH screen buffer to the new screen buffer on the terminal screen. The argument is a flag which tells whether or not to clear the screen and redraw, or only update the screen with the changes. If 'clr' is YES, the entire screen is cleared and completely redrawn. If 'clr' is NO, only the changes needed are made on the screen.

Implementation

After making the changes to the screen, the new screen buffer image is copied into the old screen buffer image for the next time around. 'Vtupd' is reasonably efficient in updating the screen and copying the old screen to the new screen.

Calls

vtclr, vtout, date, vtmove, vtmsg

See Also

other vt?* routines (2)

Calling Information

```
integer function wind (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Wind' (pronounced w-eye-nd) is the opposite of 'rewind': it positions a file's pointer to the end of the file, rather than the beginning. The argument is the file descriptor of the file to be wound. The function return is OK if the wind was successful, ERR otherwise.

Implementation

'Wind' calls 'seekf' with an extremely large position argument, thus setting the file pointer to EOF. The return value is whatever 'seekf' returns.

Calls

seekf

See Also

rewind (2), trunc (2), seekf (2)

Calling Information

```
integer function wkday (month, day, year)
integer month, day, year
```

Library: vswtlb (standard Subsystem library)

Function

'Wkday' is used to return the day-of-the-week corresponding to a given date. The three arguments completely specify the date: the month (1-12), day (1-28, 29, 30, or 31), and year (e.g. 1980). The function return is the ordinal number of the day-of-the-week (1 == Sunday, 7 == Saturday).

Implementation

Zeller's Congruence.

See Also

date (2), day (1)

Calling Information

```
integer function writef (buf, nw, fd)
integer buf (ARB), nw
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Writef' is used to write words to a file, which may be assigned to any device recognized by the Subsystem. (A word on the Prime is 16 bits long.) The first argument is a string of words to be written to the file; the second argument is the number of words to be written; the third argument is the file descriptor of the file to which data will be written. Words are transferred from the string buffer to the file until 'nw' words are written. The function return is ERR if the file is not writeable, if the given file descriptor is invalid, or if the file's error flag is set; otherwise, the function return is the number of words written (nominally the same as 'nw'). Exceptions: a write on the null device (/dev/null) always returns EOF, and an error detected by Primos causes a return of EOF.

Implementation

'Writef' calls 'mapsu' to convert standard port numbers into file descriptors. If the last operation performed on the file was not a 'writef', 'flush$' is called to empty the file's buffers. Depending on the device type associated with the file, one of the device dependent drivers 'dwrit$' (for disk files) or 'twrit$' (for terminal files) is called to perform the actual data transfer.

Calls

mapsu, dwrit$, twrit$, flush$

Bugs

Support for more devices would be nice.

EOF is returned if any error occurs when writing to disk (in dwrit$); the user is not informed of the actual error that occurs.

See Also

mapsu (2), dwrit$ (6), twrit$ (6), flush$ (6)

Section 3 - Locally-Supported Commands

This section is devoted to the description of locally-supported Subsystem commands. Georgia Tech's locally-supported commands reside in the directory "=lbin=" and are supplied on the Software Tools release tape as an interesting example of locally developed commands.

Documentation for each command is organized under the following headings. Note that a heading will be omitted if it contains no additional information.

Header Line

The command's name, function, and the date of last modification to the documentation.

Usage

A description of the syntax permitted on the command line. The notation used in this description is identical to that used in Section 1 of this manual.

Description

A detailed coverage of the capabilities and operation of the command.

Examples

A few short examples of the command.

Files

A list of the names of special files used by the command.

Messages

A listing of important error messages or diagnostic information issued by the command.

Bugs

Known bugs in the operation of the command.

See Also

References to further information or related commands.

Usage

ap <file> [<start_address>]

Description

'Ap' reads the relocatable binary file in <file>, and produces absolute code, in hexadecimal, on its first standard output. The optional <start_address> is where it will relocate the code to. The default starting address is zero.

Messages

"Usage: ap ..." if called improperly.

Bugs

Locally supported.

This description may not be entirely accurate, since this command has long been undocumented.

See Also

Whatever other programs are used for the "Allen and Paul Model 1" terminal.

Usage

as11 [g][l[a]]

Description

'As11' converts an assembly language program into a GT40 load stream. If the load stream is sent to the GT40 while it is running its ROM bootstrap (start address 166000 octal), the program will be loaded and executed (assuming no errors) on that machine.

The single argument to 'as11' is a list of option letters. The GT40 load stream will be produced only if "g" is specified. The "l" option will cause a listing to be produced. The listing is essentially a trace of the assembly process, including what code is generated into what addresses and what values are assigned to symbols. The line number of the source line responsible for each action is given. The source code is not listed. If "la" is specified, the listing will be produced for all passes, otherwise only for the last pass. The listing for all passes feature exists mainly for debugging the assembler.

If a listing is requested, it goes to standard output (1). The GT40 load stream, if requested, is written to standard output 2. As usual, error messages go to standard output 3.

For those familiar with other PDP-11 assemblers, here are the main distinguishing characteristics of 'as11':

Statements are separated with semicolons or newlines.

There is no ".word" directive; expression statements are used instead.

Comments are Ratfor-style: initiated by a sharp ("#") and closed by the next newline.

Symbols may have up to 100 characters in their names, all of which are significant. They may begin with digits so long as they do not meet the syntax for numbers. Among the characters that are legal in symbols are the letters and digits, ".", "_", and "'" so long as it is not leading.

There are Knuth-style local labels, e. g. "1h", "1f", "1b".

Numbers can be octal, decimal, hexadecimal, or default base. The default base, which defaults to 10, is settable by assignment to the symbol named "base'". Explicit indication of base is leading zero for octal, trailing "." for decimal, and trailing "'" for hex. Hex digits are from the set 0-9A-F (capital letters).

Strings are typed in single quotes, with a pair of single quotes within the string representing a single quote. One- or two-character strings can be used as literal constants.

Expressions are evaluated left-to-right (this is subject to drastic change) but with unary operators being evaluated before binary. Operators are "+", "-", "*", "/", "&" (bit-by-bit AND), "|" (OR), "~" (read "without" and meaning AND NOT) "<" (shifted left by), and ">" (shifted right by). The sense of unary operators is binary operators with a default left argument. This works so that unary "-" means minus, "~" works as NOT, and "< n" is as "1 < n", i. e. a word with only bit n on. There is no way to control grouping in expressions. Missing operators in expressions are interpreted as "+".

Arbitrary assignment to the location counter is allowed. Code is output in the order that it is specified. This makes it possible to start a display while the load is still going on.

Symbols are predefined (in lower case) for the PDP-11/20 instruction set as well as for all the VT40 (GT40 display processor) instructions and for important addresses on the GT40 (device registers, etc.). Upper case symbols are predefined for all the ASCII control codes (CR, LF, etc.).

Defining data structures is simplified with two new pseudo-ops named ".struct" and ".reserve", for details on which see the 'as11' reference manual. As an example,

```
.struct tnode    left 2, right 2, info 80
```

is equivalent to

```
tnode = 0
left = tnode; tnode = tnode + 2
right = tnode; tnode = tnode + 2
info = tnode; tnode = tnode + 80
```

Multiple assignments to symbols, whether by assignment statement (with "=") or by label (with ":"), are completely legal. The most recent definition is the one in effect.

The assembler makes sufficient passes to define all symbols, if possible. An arbitrary number of passes may be required. The location counter can be undefined over portions of passes other than the last.

Examples

```
source> as11 gl >listing >object >errors
term.s> as11 g 2>object
as11 la
try.s> as11
```

Messages

"Usage: as11 ..." for invalid argument syntax.
"-" at the start of each pass.
"can't happen" messages for conditions not expected by the author.
"In gtok: string too long" for an overlong string.
"Too many tokens in source program" indicates that a table needs to be enlarged.
"Too many tokens put back" should not occur.
"Unrecognized statement"
"Syntax error in .byte directive"
"Bad syntax in .struct or .reserve directive"
"Bad syntax in string statement"
"Bad syntax in assignment"
"Instruction syntax"
"Bad syntax in branch instruction"

Bugs

Locally supported.

See Also

focld (3), as6800 (3), as8080 (3), *PDP-11 Cross Assembler Reference Manual*

Usage

as6800 [-{l | d}]

Description

'As6800' is an extremely simplified cross-assembler for the Motorola 6800 microprocessor.

It is designed to assemble the output of the SSPL compiler in a single pass, producing a file of relocatable object code with a symbol table (suitable for later processing by 'mot' or 'lk'). It is also suitable for limited assembly language coding by hand (allowing the SSPL run-time package to be assembled).

'As6800' takes a source program on its first standard input and produces an object program on the file ".o". The assembler differs from Motorola's standard in the following ways:

1. Instruction mnemonics that make use of an accumulator symbol ("a" or "b") may not be separated from the accumulator symbol. Thus, "add a #5" is illegal; the correct form is "adda #5".
2. Labels may appear in any mixture of upper and lower case. Character case is significant. Instruction mnemonics must appear in lower case. Labels may include the underscore (_) and the grave accent (`) characters; those labels beginning with underscore are not written to the object file symbol table and may be used freely as temporaries. All characters in labels are significant.
3. Binary and octal number representations are not supported. Decimal integers may be used, or hexadecimal integers preceded by a dollar sign ($).
4. The following pseudo-operations are totally unsupported: end, pag, opt, equ. The fcb and fdb pseudo-ops have been replaced by the 'byte' and 'word' pseudo-ops (otherwise identical in function). The rmb pseudo-op has been replaced by the 'res' pseudo-op, with identical function. The 'org' pseudo-op may not be used to decrease the location counter; only forward origins are allowed.
5. Arithmetic expressions more complex than labels or simple integers are not supported. (SSPL does not generate them.)
6. Comments are indicated by preceding commentary text with a percent sign (%). This rule applies uniformly; all comments must be preceded by a percent sign.
7. Unless the "-l" option is specified, 'as6800' does not produce an assembly listing. When "-l" is used, the assembly listing is produced on standard output one.
8. Multiple statements may be place on one line provided they are separated by semicolons (;).
9. Unless the "-d" option is specified, the 'direct'

(page-zero) addressing mode is not used (since instructions using it cannot be relocated). When the "-d" option is used, the user must take care that no direct-address forward references are made.

The object file produced by 'as6800' contains a number of "segments," each consisting of a one-byte segment header, two bytes of segment size, and the text of the segment. Each byte of the object file occupies one 16-bit word in the physical file.

Examples

```
rtr.as> as6800
mux.s> lex | sspl | opt6800 | as6800
```

Files

".o" for the object code file

Messages

Many error messages, hopefully some of which are self-explanatory.

Bugs

Locally supported.

See Also

sspl (3), opt6800 (3), as8080 (3), as11 (3), mot (3), lk (3)

Usage

as8080

Description

'As8080' is an extremely simplified cross-assembler for the Intel 8080 microprocessor.

It is designed to assemble the output of the SSPL compiler in a single pass, producing a file of relocatable object code with a symbol table (suitable for later processing by 'intel' or 'lk'). It is also suitable for limited assembly language coding by hand (allowing the SSPL run-time package to be assembled).

'As8080' takes a source program on its first standard input and produces an object program on the file ".o". The source program differs from Intel's standard format in the following ways:

1. Instruction mnemonics and labels may appear in any mixture of upper and lower case. Case is significant in labels, but not significant in instruction mnemonics or register names. Labels must be defined with trailing colons (:) and may include the underscore (_) and the grave accent (`) characters; those labels beginning with underscore are not written to the object file symbol table and may be used freely as temporaries.
2. Register pairs may optionally be referred to as bc, de, hl, and sp.
3. Binary and octal number representations are not supported. Decimal integers may be used, or hexadecimal integers preceded by a dollar sign ($).
4. The following pseudo-operations are totally unsupported: end, equ, macro, org, set. The db and dw pseudo-ops have been replaced by the 'byte' and 'word' pseudo-ops (otherwise identical in function).
5. Arithmetic expressions more complex than labels or simple integers are not supported. (SSPL does not generate them.)
6. Comments are indicated by preceding commentary text with a sharp sign (#). This rule applies uniformly; all comments must be preceded by a sharp sign.
7. 'As8080' does not produce an assembly listing.
8. Multiple statements may be place on one line provided they are separated by semicolons (;).

The object file produced by 'as8080' contains a number of "segments," each consisting of a one-byte segment header, two bytes of segment size, and the text of the segment. Each byte of the object file occupies one 16-bit word in the physical file.

Examples

```
rtr.as> as8080
mux.s> lex | sspl | opt8080 | as8080
```

Files

".o" for the object code file

Messages

Many error messages, hopefully some of which are self-explanatory.

Bugs

Locally supported.

See Also

sspl (3), opt8080 (3), as6800 (3), as11 (3), intel (3), lk (3)

Usage

basys

Description

'Basys' is a program designed to simulate a basic computer system configuration consisting of a central processing unit, a disk drive and a central memory. It is a distribution driven simulator. In other words, the work load under which the system is to be simulated is characterized by probability distributions for such factors as the time between job arrivals, the priority of a job, the amount of memory required by a job, the number of i/o requests a job will make, and so on.

When invoked, 'basys' asks the user for the following information:

- **simulation length**
  The number of seconds for which the simulation run should last is given here. Note that this is simulated time, not real time.

- **memory size**
  This is the size of main memory in K-words. All jobs generated by the simulator have memory requirements in the range 2K to 70K words; thus, the memory size specified should probably be greater than or equal to 70.

- **time to compact memory**
  This is the amount of time in microseconds required to compress memory to make space available for a job.

- **per word disk transfer time**
  This is the time in microseconds required to transfer one word between disk and memory.

- **disk access time**
  This is the time in milliseconds required to position the disk head over the desired sector before a transfer takes place. It includes both seek time and rotational latency.

- **i/o overhead time**
  This is the overhead time in microseconds involved in the initiation of an i/o request.

- **mean job interarrival time**
  The time in milliseconds between job arrivals is generated from an exponential distribution whose mean is specified here.

- **mean cpu time per job**
  The number of milliseconds of cpu time required by each job is generated from a normal distribution whose mean and standard deviation are specified by this parameter and the next.

- **standard deviation of cpu distribution**
  The standard deviation (in milliseconds) of the per-job cpu time distribution.

- **mean number of i/o requests per job**
  The number of i/o requests that a job will make is determined by a normal distribution for which the mean value is specified here.

- **standard deviation of i/o distribution**
  The standard deviation of the distribution from which the per-job number of i/o requests is generated is specified here.

- **minimum record size**
  This is the minimum number of words in a single i/o transfer. All transfers requested by a single job involve the same number of words.

- **maximum record size**
  The maximum number of words in an i/o transfer is specified here.

- **event trace**
  If the user responds with "yes", 'basys' will print on standard output two a listing of all pertinent events as they are scheduled and as they occur. Events traced are job arrival, memory request, cpu request, cpu release, disk request, disk release and job termination. WARNING: under reasonable durations of simulation, the volume of output produced by this option is prohibitively large.

Upon completion of the simulation, 'basys' prints on standard output one a three part report. The first part is a summary of system parameters as specified by the user. The second part is a table of job descriptions, sorted by ascending job number. The table gives a profile of each job that entered the system during the simulation run. Finally, a summary of how well the system performed is printed. Included in this part are statistics on utilization of the three system resources: memory, cpu and disk, as well as statistics on the wait queues for each of these resources.

Examples

```
parameters> basys
basys  >report  >event_trace
```

Bugs

Meaningless results accompany meaningless input.

Locally supported.

Usage

```
bind [-(a|b|f|n|p|u)] { <binary file>
                        -d [ <entry name> ]
                        -l <library file>
                        -m [ <map file> ]
                        -i
                        -t
                        -s <loader command> }
                        [ -o <output file> ]
```

Description

'Bind' calls the Primos EPF loader (BIND) from the Software Tools subsystem.

The following global options indirectly affect the production of loader commands:

-a Modify the load sequence to include run-time support for Pascal programs. This option may be used with '-b' and '-p' for mixed-language programs.

-b Modify the load sequence to include run-time support for C programs. (The load of the C main program is triggered by the appearance of the first binary file or library.) This option may be used with '-a' and '-p' for mixed-language programs. Besides loading the C run-time library, "ciolib", this option automatically loads the SWT math library, "vswtmath", and the shared shell library, "vshlib".

-d Cause all unresolved external references at the end of the load to be resolved with Primos direct entry links.

-f Generate a full load map after commands are complete. The name of the map file will be the same as the name of the output file with the ".o" suffix (if any) replaced by ".m". This option performs the same action as the options "-t -m" at the end of the argument list.

-n Do not declare the SWT common blocks or load the default libraries unless the '-i' and '-t' options are encountered. This allows the loading of non-Subsystem programs or the insertion of additional loader commands at the beginning and end of the load.

-p Modify the load sequence to include run-time

support for PL/I subset G programs. This option may be used with '-a' and '-b' for mixed-language programs.

-u Generate a load map of undefined symbols after the default libraries have been loaded.

-w Modify the load sequence to include run-time support for Prime C programs.

The following local options are examined in the order presented and directly produce commands to the loader:

<binary file> specifies a binary code file to be loaded.

-l <library file> specifies a library file to be loaded.

-s <Bind loader command> allows arbitrary loader commands to be inserted in the command stream

-m <map file> presents a map command to the loader. If <map file> is omitted, the first "<binary file>.m" is assumed. (If <binary file> ends with ".b", the "b" is replaced with an "m".)

-i causes the inclusion of the initial sequence of Subsystem program loader commands (the definition of Subsystem common block locations) to be included, regardless of the "-n" global option.

-t causes the inclusion of the terminal sequence of Subsystem program load commands (the default library loads) to be included, regardless of the "-n" global option. If the "-n" option is not specified, the sequence of commands will be included at this point, so that loader commands may be inserted after the libraries have been loaded. This option may be used with the "-m" option to generate a full load map.

-o <output file> specifies the output file for the results of the load. If omitted, the first "<binary file>.o" is assumed. (If <binary file> ends with ".b", the "b" is replaced with an "o".)

Commands are presented to the loader in the order in which they are encountered in the command line, except for "-o", which appears only at the end of the command stream.

Examples

```
bind -u rf.b -t -m
bind sol.b -o sol -d
bind test.b -d at$ -o test
bind sh.b -s "ma -symbols" -o sh -f
```

Bugs

'Bind' pays no attention to standard ports.

'Bind' must be able to create files in the current directory.

All files specified must be disk files.

EPF's are not currently supported and Primos BIND is not currently documented. Use of this command is discouraged until Prime supports EPF's.

See Also

fc (1), pc (1), plgc (1), f77c (1), pmac (1), x (1), rfl (1), ld (1)

Usage

block [-c <char>] [-w <width>]

Description

'Block' reads lines of text from standard input, converts them to large block letters, and writes them on standard output. Each character produced is 5 columns in width by 9 lines in height, with 2 blank columns between consecutive block letters and 3 blank lines between consecutive lines of block letters.

The character used to construct the block letters may be specified with the "-c <char>" argument sequence; the default character is an asterisk (*). Similarly, the length (in regular characters) of the lines produced by 'block' may be specified with the "-w <width>" sequence. If omitted, a default width of 75 columns is assumed. Input lines that will not fit on a single output line are broken into as many consecutive lines as necessary.

Normally, 'block' ignores control characters in the input stream. The two exceptions to this rule are NEWLINEs, which force a new output line, and BACKSPACEs, which may be used to produce underlined, boldfaced or other overstruck characters.

Examples

```
echo "@n@n  In Use" | block
cal 1981 | block -w132 >/dev/lps
```

Messages

"Usage: block ..." for invalid argument syntax.

See Also

banner (1)

broadcast (3) --- send a Primos message to a user on all machines 07/20/83

Usage

broadcast [(<user> | all) [<message>]]

Description

The 'broadcast' command allows users to send a Primos message on all systems that are running the SWT process 'ring'. If the first argument is "all", 'ring' broadcasts the message to all users on each machine in the ring, otherwise <user> is the user name of the recipient of the message. The remaining arguments constitute the text of the 80 character message to be broadcast. If omitted, 'broadcast' reads one line from standard input (STDIN) and broadcasts it. If no arguments are given, 'broadcast' reads one line from STDIN and broadcasts it to all users.

Examples

broadcast all System going down in 5 minutes. Please log off.

broadcast jeff Your wife called. The house burned down.

Messages

Cannot transmit BROADCAST request
: Something interfered with the transmission of the BROADCAST command to the 'ring' process. This should never happen.

Message complete
: The BROADCAST command has been successfully attempted on all systems in the ring.

Message has been transmitted
: The BROADCAST command has been transmitted to the 'ring' process.

Networks are not configured
: The system is not configured to support PRIMENET.

Request to <system> failed
: The attempt to broadcast the message on system <system> failed.

Request to <system> succeeded
: The attempt to broadcast the message on system <system> succeeded.

Ring connection has been terminated
: The connection to the 'ring' process has been cleared.

Unable to connect to ring node
The current system is not running a 'ring' process.

You are not validated to BROADCAST
Your user number is not allowed to use the BROADCAST command.

Bugs

Will not work if the current system is not running 'ring'.

Message is sent by the 'ring' process because that is the process which actually executes the Primos 'message' command.

See Also

execute (3), setime (3), terminate (3)

Usage

bug

Description

'Bug' allows users to report any problems they may encounter with system commands or libraries. It creates a standardized report whose existence is announced to the system administrator at each login until the report is examined. 'Bug' prompts for such information as the user's name, a description of the bug, and the name of any file (e.g., program source code or comoutput file) which helps illustrate the bug.

Examples

bug

Files

=bug=/r??? for storage of the bug report
=bug=/s??? for storage of the user's environment at the time the bug was reported

See Also

raid (3)

Usage

cal [<month>] [<year>]

Description

'Cal' prints a Gregorian calendar for any month or any year of the user's choice. When invoked without arguments, a calendar for the current month of the current year is printed in the following format:

```
      May         1980

   Su Mo Tu We Th Fr Sa
                  1  2  3
    4  5  6  7  8  9 10
   11 12 13 14 15 16 17
   18 19 20 21 22 23 24
   25 26 27 28 29 30 31
```

In addition, either the name of a month, or a year, or both may be specified on the command line to select the calendar to be printed. If a month name is given without a year, a calendar for that month in the current year is produced. If a year is given without the name of a month, then a calendar for the entire year is printed.

'Cal' will accept any unique initial abbreviation for the name of a month. Also, if a year between 0 and 99 is specified, 'cal' assumes the 20th century.

Examples

```
cal
cal october
cal 1980 | col -c 3 -w 20 -l 11 | pr
cal 1981 | block -w 132 >/dev/lps
```

Messages

"Usage: cal ..." for invalid argument syntax.

Bugs

Can't produce calendars for other than the 20th century.

See Also

date (1), day (1)

Usage

```
chown [-s[<depth>]] <owner> { <pathname> }
```

Description

'Chown' is used to change the owner password of one or more directories (UFDs) in the file system. <owner> may consist of up to six characters; shorter strings are padded with blanks, and lower-case letters are converted to upper-case.

In order to use this command successfully, one must be able to attach to the named directories with owner privileges. In a standard Primos environment, this means the current owner password must be included in the pathname for each specified directory. In a Ga. Tech Primos environment, this means that the user must currently own the specified directories, or they must be public.

The "-s" option, if specified, causes 'chown' to traverse the file system subtree rooted in the named directory, changing the owner password of each directory it encounters. The depth of this traversal may be limited by appending a positive integer to the "-s" (e.g., "-s3").

Specifying no <pathname> arguments is the same as specifying the pathname of the current directory.

Examples

```
chown "" =mail=
chown ""
chown system =src= =src=/lib
chown -s system =aux=
```

Messages

"Usage: chown ..." for bad argument syntax.
"owner name too long" for illegal <owner>.
"<pathname>: can't change owner" for not being owner of <pathname>.
"<pathname>: bad pathname" for not being able to access the directory containing <pathname>.

Bugs

In a standard Primos environment, 'chown' sets the non-owner password to zeros when it changes the owner password.

See Also

cd (1), chat (1), lacl (1), sacl (1), passwd (3), Primos spas$$

Usage

cron

Description

'Cron' allows the system administrator to have shell files executed automatically at a given time. It does this by creating phantoms to execute the file. The attributes (name, project, and associated groups) of the phantom can be specified, along with the time, by the administrator. 'Cron' must be started from the system console to retain the privileges needed to spawn phantoms for alternate user numbers. It could be started as a final step in the boot procedure by phantoming it from the boot initialization file (c_config or primos.comi).

'Cron' gets its information from the file "=cronfile=". It periodically (currently, every minute) wakes up and reads the information in this file. If it finds a job that should be run, it collects the attributes (name, project, and groups) that the job is to have, creates a temporary file in the directory "=crondir=" to hold the commands, copies the file to execute into the temporary file, and calls 'sph' to attempt to spawn the executing process. Before the spawn attempt, 'cron' prints information, on STDOUT, as to what file is being phantomed for whom, and the attributes of the phantom.

The file "=cronfile=" contains a line for each job to be processed. Each line contains a sequence of space seperated sequences of numbers that describe the time and day to phantom the process, the user name and project name to phantom, the name of the file to phantom, and the list of groups the phantom should have. Each space seperated time description may contain up to 10 comma seperated fields indicating multiple times when this job is to be run. The times specify, in order, the minute, hour, day, day of the month, and month that a command is to be run. If a time does not matter, it can specified by a '*' in the appropriate column. In addition to the job descriptions, the file may contain comments by simply placing a '#' in the first column of the line. For example, if "=cronfile=" contained

```
# min hrs  day dat mth user   proj file          groups
    0 6,18  *   *   *  jeff   lab  //acct/fix    guru
```

then the first line would be ignored (because of the '#' in column 1) and the second line would cause the shell file "//acct/fix" to be executed by "jeff" under project "lab" with the ".guru" group at 6am and 6pm every day. A more complicated example would be

```
# min hrs  day dat mth user   proj file          groups
   15  *   1,7  *   12 arnold lab  //system/die  scum
```

which would have "arnold" with project "lab" and group ".scum" run the file "//system/die" every 15 minutes after the hour on weekends (Sunday = 1, Saturday = 7) during the month of December (January = 1, December = 12). Another example might be a program to be run on the first day of the year, to send a happy new year message to everyone. This might look like

```
# min hrs  day dat mth user   proj file
   0   0    *   1   1  system lab  //message
```

which would run "//message" on January 1 at midnight.

When specifying the time, care must be taken to prevent a command from being over-executed. If in the second example the entry had been

```
# min hrs  day dat mth user   proj file          groups
   *   *   1,7  *   12 arnold lab  //system/die  scum
```

then the file would have been executed every minute during the weekends in December.

Messages

"can't open =cronfile=" when "=cronfile=" does not exist or is unreadable.

"can't open <file>" when the file to spawn does not exist or ir unreadable.

"can't create <file>" when it can't create temporary files in the directory "=crondir=".

Any message that 'sph' can generate.

Examples

cron

Bugs

Locally supported until Prime supports EPF's and the SPAWN$ subroutine call.

Generates lots of output if "=cronfile=" does not exist.

See Also

sph (5)

Usage

```
des  (-e | -d)  [-k <key>]  {filename}  >output_file
```

Description

'Des' is an implementation of the National Bureau of Standards Data Encryption Standard. It can be used for protection of on-line copies of sensitive information.

The first argument must be "-e" (for "encryption") or "-d" (for decryption). The DES is not self-inverting like the exclusive-or algorithm used in 'crypt'; the encryption and decryption processes are different and one or the other must be selected.

The optional argument sequence "-k <key>" can be used to specify a key to control the encryption or decryption process. If a key is not specified on the command line, 'des' will print a prompt message, turn off the terminal's character echo, and read the key. Furthermore, after the key has been read, 'des' will prompt the user for key validation; the key must then be re-entered to insure that no typographical errors occurred during the original key entry.

Remaining arguments must be the names of files containing information to be encrypted or decrypted. Filenames may be read from standard input as well as from the command line; see the reference manual entry for 'cat' for further information. If no filenames are specified, 'des' takes data from its first standard input. DO NOT use this feature to read data from the terminal; 'des' uses binary I/O with the unfortunate side effect that end-of-file cannot be generated from a terminal keyboard.

The output of 'des' is always produced on its first standard output. Because 'des' processes data in binary rather than ASCII form, its output will not be displayed correctly on a terminal. Always direct the output of 'des' to a disk file or into a pipe for further processing.

Examples

```
des -e -k turkey document >document.des
des -d -k turkey document.des >original_document
```

Messages

"Usage: des ..." for improper argument syntax.
"keys do not match" if the validation key entry does not match the original key entry.

Bugs

Binary I/O is needed to handle the arbitrary bit-patterns output by the DES algorithm without considerable expansion of the output text. Unfortunately, binary I/O from and to the terminal does not behave rationally at all.

The present implementation is very slow, averaging about 1730 bits per CPU second throughput.

See Also

crypt (1)

Usage

dmach <hex file>

Description

'Dmach' is a simulator for the microprogrammed Burroughs D-machine described in *Microprogramming Primer*, by Harry Katzan, Jr. (McGraw-Hill, 1977). Because of the detailed treatment of the machine architecture and programming techniques given in that text, these topics are not covered here.

'Dmach' requires one command line argument: the name of a file that contains the hexadecimal representation of a microprogram. The file will typically have been generated by the 'translang' command, which is the translator for the symbolic microprogramming language described by Katzan. (See the documentation for 'translang' for further details.)

Upon invocation, 'dmach' prompts the user for information that it needs to control the simulation environment and to provide a trace of its activities. What follows is a description of the prompt messages that are printed and the proper user responses:

Begin execution at address:

The user should enter the microprogram address at which execution is to begin. Possible values lie in the range 0 to 1023 (decimal).

Number of clock cycles to simulate:

The response is used as an upper bound on the number of microprogram clock cycles that are to be simulated. If the microprogram consumes this many cycles without executing a halt instruction, simulation is terminated.

Number of cycles between traces:

This determines the number of clock cycles that occur between each trace point.

Begin tracing at address:
End tracing at address:

These parameters delimit a region of the microprogram that is to be traced. Trace output is generated at trace points only if the address of the microinstruction just executed falls within this region.

Print S-memory and registers in octal?

If the response is "yes", the values of registers and

S-memory locations will be represented as unsigned octal numbers in the trace and memory dump output; otherwise, the representation is signed decimal.

Dump S-memory upon termination?

If the response is "yes" the user is given the opportunity to view the contents of S-memory when simulation is terminated; otherwise, no dump is provided.

```
Load S-memory -- Enter <return> to quit
Starting address (1-2048):
Ending address           :
```

The user should enter the starting and ending addresses of a contiguous block of S-memory to be initialized before execution of the microprogram begins. 'Dmach' then issues a series of prompts of the form

```
Smem (n) =
```

where n is the address of a cell to be initialized. Such a prompt is issued for each cell in the block delimited by the starting and ending addresses, and then another pair of addresses is requested. This dialogue continues until the user enters an empty line for the starting address.

All numeric items are expected in decimal; however, if input in another radix is desired, the item may be preceded by the radix followed by a letter "r". For example:

```
8r177
```

is interpreted as octal 177 (127 decimal).

Since all input is read from 'dmach's first standard input port, the responses may be stored in a file to eliminate the need for redundant typing. Normally, such a file would contain one response per line. When input is being read from a file, no prompt messages are printed.

After the last block of S-memory has been initialized, microprogram execution begins. Depending upon the user's responses to the above questions, trace output is printed at selected trace points in the microprogram. This output takes the following form:

```
Phase 3 Mpcr: 0
Clock: 2     A1: 0       B   : 0       Ctr: 0    Br1: 0
Mpcr : 1     A2: 0       Mir : 0       Lit: 0    Br2: 0
Ampcr: 0     A3: 0       Bmar: 0       Sar: 0    Mar: 0
Mst Lst Abt Aov Cov Sai Rdc Gc1 Gc2 Lc1 ... Ex1 ... Int
 F   F   F   T   F   T   F   T   F   F  ...  F  ...  F
```

The various fields should be self explanatory for users familiar with the D-machine architecture. The only field present in the trace output not covered by Katzan is the 'Bmar' field, which simply contains the address of the last S-memory location fetched or written by the microprogram. It is obtained by concatenating one of the two base registers 'Br1' or 'Br2' with the memory address register 'Mar'.

Upon termination of the microprogram, either because of executing a HALT instruction or exceeding the specified number of clock cycles, 'dmach' prints one final trace sequence summarizing the final machine state. If a memory dump was requested during the initial dialogue, a series of prompts similar to that for S-memory initialization is issued and the user is allowed to inspect selected blocks of the final S-memory.

Examples

```
dmach mult.h
parameters> dmach microprogram.h
dmach divide.h >trace_output
```

Messages

"Usage: dmach <hex file>" for incorrect argument syntax.
"Error -- Bad external operation" when an external operation other than memory read or write is requested by the microprogram.
"Error -- S-memory address out of range" when the address supplied for a memory read or write command is not in the range 1 to 2048.
"Error -- Memory not ready for read" when a memory read command is issued before the previous one is complete.
"Error -- Memory not ready for write" when a memory write command is issued before the previous one is complete.

See Also

translang (3), *Microprogramming Primer*

Usage

dprint {-c <copies> | -j | -l <length> | -s | -x} { <file_spec> }

Description

'Dprint' prints files on the user's terminal, making the assumption that the terminal is a Diablo model 1610 or 1620. Printing is done bi-directionally, optimizing motion of the print head and platen as much as possible.

The following options are available to control 'dprint's behavior:

-c If present, the next argument must be an integer; 'dprint' will produce the specified number of copies of each file that it prints.

-j 'Dprint' will cause a page eject following each file (or copy of a file, when multiple copies are specified). Normally, no extra space is inserted between successive files or copies of the same file.

-l If present, the next argument must be an integer; 'dprint' will use that number as the number of lines per physical page. It is important that this number match the form length selected on the terminal itself, else anomalous behavior may result. 'Dprint' assumes there are 66 lines per page, corresponding to the standard 11 inch form length.

-s This option causes 'dprint' to pause at the top of each page and sound the terminal's audible alarm, allowing the user to insert a new piece of paper. Printing continues when the user types an ACK character (ctrl-f).

-x This option prevents the initial page eject. This option is useful when printing on special forms (mailing labels, etc.). Since the diablo does not support relative vertical motion this option should only be used when the paper/form is initially at top of form.

The remaining arguments specify files to be printed. Most often, one or more pathnames will be given, indicating that the named files are to be printed, or there will be no other arguments, indicating that input is to be read from standard input. The full syntax of the <file_spec> construct is described in the entry for 'cat' in section 1 of the Reference Manual.

It is assumed that the paper has been mounted so that a form feed will advance to the first line on the next page. This may be done by pressing the 'set tof' switch (in the upper right corner of the keyboard) after the paper has been

positioned properly.

In addition to optimizing print head motion, 'dprint' provides an extended character set of Greek letters and mathematical symbols to support the special character functions of the Software Tools Subsystem text formatter, 'fmt'. These special graphics are accessed by normal ASCII character codes with their most significant bit turned *off*. (Note that the normal Prime convention is that this bit is always turned on for text characters.) The following table shows the correspondence between ASCII character codes, formatter functions, and special graphics:

Character	*'Fmt' Function*	*Graphic*
a	alpha	lower-case Greek alpha
b	beta	lower-case Greek beta
d	delta	lower-case Greek delta
D	DELTA	upper-case Greek delta
e	epsilon	lower-case Greek epsilon
n	eta	lower-case Greek eta
g	gamma	lower-case Greek gamma
G	GAMMA	upper-case Greek gamma
8	infinity	"infinity" symbol
+	integral	integration symbol
l	lambda	lower-case Greek lambda
L	LAMBDA	upper-case Greek lambda
u	mu	lower-case Greek mu
^	nabla	inverted delta (APL del)
~	not	EBCDIC-style "not" symbol
v	nu	lower-case Greek nu
w	omega	lower-case Greek omega
W	OMEGA	upper-case Greek omega
-	partial	partial differential symbol
p	phi	lower-case Greek phi
P	PHI	upper-case Greek phi
y	psi	lower-case Greek psi
Y	PSI	upper-case Greek psi
3	pi	lower-case Greek pi
4	PI	upper-case Greek pi
r	rho	lower-case Greek rho
s	sigma	lower-case Greek sigma
S	SIGMA	upper-case Greek sigma
t	tau	lower-case Greek tau
h	theta	lower-case Greek theta
H	THETA	upper-case Greek theta
x	xi	lower-case Greek xi
z	zeta	lower-case Greek zeta

These extended graphics are produced by fractional motions of the platen and print head and overstriking of standard ASCII graphics. Best results are obtained when the paper is being fed by the platen and pinch roller and not by the pin-feed mechanism.

Examples

```
help -p dprint | dprint
dprint junk
dprint -s -l 80 journal_article
dprint -c 5 -j hand_out
```

Messages

"Usage: dprint ..." for incorrect argument syntax.
"<filename>: can't open" if given file could not be opened for reading.

Bugs

If interrupted by the BREAK key while printing, 'dprint' may hang, waiting for the Diablo to acknowledge the last group of characters sent. To clear this condition, it is simply necessary to type a ctrl-f at the keyboard. If the ctrl-p key is used instead of BREAK, this condition normally does not occur.

When multiple copies of a file are requested using the "-c" option, 'dprint' obliges by rewinding the input file and re-reading it. If the input is being taken from a standard input port, and that port is not connected to a rewindable device (i.e., a disk file), then only one copy is produced.

Error messages are produced on the standard error output port, which is normally directed to the terminal. If it is undesirable to have these messages interspersed with the contents of the printed files, error output should be redirected to a file.

Does not currently handle the full set of 'fmt' special characters.

See Also

cat (1), copy (1), fmt (1), print (1), sprint (3)

Usage

execute [(<system> | all) [<command>]]

Description

The 'execute' command interfaces to the SWT 'ring' process to allow users to execute a SWT command on any system that is running 'ring'. If the first argument is "all", 'ring' executes the command on all machines in the ring; otherwise it specifies the system name of the machine on which to execute the command. Any remaining arguments comprise the text of the commands to be executed. If these arguments are omitted, 'execute' reads one line from standard input (STDIN) and executes it. If no arguments are given, 'execute' reads one line from STDIN and executes it on all systems.

In order to execute a command on another system, the 'ring' process on the target system starts up a phantom with the requesting user's name. This phantom has the same home and current directories as the 'ring' process, and therefore the command line should contain a 'cd' command to attach to the correct directory. 'Ring' has no way to determine if a user is validated to log on to a system. It simply creates a phantom, and the 'swt' command will kill the process if it finds no "vars" directory.

Note that the output from the phantom that 'ring' creates is not transmitted back to the user who requested the service. In order to save that output, it must be written to a file and delivered to the user in some other fashion (e.g. 'mail').

Examples

execute gt.a sema drain -32

execute all "cd; lf | mail roy"

Messages

Cannot transmit EXECUTE request
: Something interfered with the transmission of the EXECUTE command to the 'ring' process. This should never happen.

Command complete
: The EXECUTE command has been successfully attempted on all systems in the ring.

Command has been transmitted
: The EXECUTE command has been transmitted to the 'ring'

process.

Networks are not configured
The system is not configured to support PRIMENET.

Request to <system> failed
The attempt to execute the command on system <system> failed.

Request to <system> succeeded
The attempt to execute the command on system <system> succeeded.

Requested system is not in the ring
The system on which the commands were supposed to execute is not in the ring.

Ring connection has been terminated
The connection to the 'ring' process has been cleared.

Unable to connect to ring node
The current system is not running a 'ring' process.

You are not validated to EXECUTE
Your user number is not allowed to use the EXECUTE command.

Bugs

Will not work if the current system is not running 'ring'.

Cannot determine whether or not user is validated to log on to the requested system(s).

Starts up phantoms in the 'ring' directory rather than in the user's directory.

See Also

broadcast (3), setime (3), terminate (3)

Usage

fixp [-z] [-mu | -ml] [-cl | -cd] [-u] [<infile> [<outfile>]]

Description

The Prime operating system adopts a convention whereby every ASCII character has the most significant of eight bits (the so-called "parity" bit) always set "on". This can lead to some difficulties when dumping a tape from another system since not all systems use the eighth bit in this manner.

'Fixp' is a translation program designed to transform foreign files to SWT and Primos compatible text files. It has some other interesting capabilities, discussed below. It is written in PMA to utilize the cpu's character instructions and thus operates very quickly. By default, 'fixp' always sets the parity bits of every character to "on".

The "-z" option causes all null characters, except those used for padding at the end of a string, to be deleted. This option does not cause ASCII nulls (octal '200) to be deleted if they are present in the input, it simply deletes total null (octal 0) characters from the text stream. This option can be used when stripping padding from 'se' temporary files during recovery operations, for instance.

The "-mu" and "-ml" options map the output into upper (mu) or lower (ml) case. This operation is at least an order of magnitude faster than 'tlit' and thus may be useful by itself.

The "-cl" and "-cd" options are for files which have carriage return characters in them. The "-cd" option simply drops any carriage returns. The "-cl" option turns all carriage returns into linefeed/eol characters.

The "-u" option strips all control characters from the output except for linefeed/eol characters. Thus, 'fixp' may be used as a filter for removing all but visible characters from a text stream.

By default, 'fixp' creates a standard compressed-ASCII file. If the "-u" option is given then the output is NOT compressed but contains the actual blanks that would normally be compressed out. It should be noted that such files can take up many times more disk space than the equivalent compressed files, and thus this option should be used carefully.

If no output file is given 'fixp' sends output to STDOUT. If no input file is given then 'fixp' takes input from STDIN.

If no options are given, defaults are no case mapping, include all carriage returns unchanged, include all control

characters unchanged, and leave all padding zeros in place (except that parity bits get turned on so they become ASCII nulls).

Examples

```
from_tape> fixp -mu -cl >text
fixp //spaf/foo //dan/bar_none
```

Messages

"Usage: fixp ..." for invalid argument syntax.
"<name>: Cannot open" for files that cannot be opened.
"Error in fixp" when an error condition other than EOF occurs with the input.

Bugs

The non-compression of blanks with the "-u" option might be considered a bug.

The program code is self-modifying and should not be put in protected or shared memory regions without modification.

Due to the behavior of 'readf', when 'fixp' takes input from the terminal there is no way to trigger the EOF condition and thus the program will never end! Large scale buffering is used so you may not immediately observe any output, either.

'Fixp' cannot reverse its actions, it can't turn the parity bits back off.

Locally supported.

See Also

mt (1), tlit (1)

Usage

focld <filename>

Description

'Focld' is used to prepare a FOCAL program stored on the Prime for acceptance by FOCAL-GT/RT. 'Focld' is used from FOCAL in the following manner:

Type control-f.
Type your "kill" character.
Without hitting return, type: focld <filename>
Type control-t.
Hit return.
Wait while the program loads.
Type control-t and control-f and you will be talking to FOCAL.

It is advisable to study section 3.1 and figure 3-1 of the *FOCAL-GT/RT User's Manual* to understand the above procedure and to develop the procedure for saving FOCAL programs.

Examples

focld life
focld chess

Bugs

Locally supported.

See Also

as11 (3), *FOCAL-GT/RT User's Manual*

Usage

imi <object_file> [<relocation>]

Description

'Imi' takes the output of the Motorola 6800 cross-assembler ('as6800'), relocates it to the desired starting address (0000 by default), and generates a down-line load stream suitable for use by the International Microsystems, Inc., prom programmer.

Note that the relocation address is given in hexadecimal. Other bases may be specified by preceding the address with the desired base followed by the letter "r," e.g. "8r100000".

Examples

imi mux
imi highloader 4000

Messages

"Can't open" for unreadable object file.
"Usage: imi ..." for missing arguments.
"badly formed code file" for erroneous code files.

Bugs

Locally supported.

See Also

as6800 (3), lk (3), intel (3), mot (3)

Usage

intel <object_file> [<relocation>]

Description

'Intel' takes the output of the Intel 8080 cross-assembler ('as8080'), relocates it to the desired starting address (0000 by default), and generates an Intel standard format object tape on its first standard output.

'Intel' is useful for down-line loading assembled code to development systems equipped with a standard ROM loader.

Examples

intel mux
intel highloader 16384

Messages

"Can't open" for unreadable object file.
"badly formed code file" for erroneous code files.

Bugs

Locally supported.

See Also

as8080 (3), lk (3), mot (3)

Usage

kill <pid>

Description

'Kill' logs out the user (process) whose process number is given as the first argument. It is a shorthand for "x lo -<pid>".

Examples

kill 19

Messages

"Usage: kill ..." for a missing argument.

See Also

x (1), ph (1)

Usage

last [-t] [-c] [-v] [-l<#>] [-n | -n<#> | {<pathname>}]

Description

This program allows the user to print the last (or first) "n" lines of a file, or of standard input. In addition, it does a high speed count of the number of lines in the file and can be used simply to size the file. A combination of text printing and counting may be chosen.

"-t" -- prints the lines of text requested (last or first "n" lines).

"-c" -- prints the number of lines in the file followed by a blank line.

"-v" -- print the pathname of the file. The pathname is printed after the count (if the count is requested) and is followed by a blank line.

"-l<#>" -- <#> is any number between -32767 and +32767. If the number is positive, then the last <#> lines are printed as text. Otherwise (<#> is negative), the first abs(<#>) lines are printed as the text. Therefore, the top 10 lines could be printed with "-l-10".

"-n or -n<#>" -- standard format to input a list of filenames to be used as arguments.

All output, except error messages, goes to STDOUT. Default options are: last -l20 -t

Examples

```
=userlist=> last
last -t -l50 [lf -c] | sort | uniq >ends
echo [last -c [file]]" lines in "[file]
```

Files

If input is from STDIN and is from the terminal then the input is copied into a file opened with 'mktemp' before positioning is done (since the terminal cannot be "positioned").

Messages

"<name>: Cannot open" for files that cannot be opened.
"Usage: last ..." for incorrect argument usage.

Bugs

Very peculiar behavior will occur if 'last' is used on something other than text (as in binary image files). It also will not work correctly on files which do not have the character parity bits set on (Prime and SWT standard).

Locally supported.

See Also

tail (1), tc (1), slice (1)

Usage

lfo {<process id> | <user name>}

Description

'Lfo' prints the following information on STDOUT for each process number and for all processes owned by each user name specified:

1. the user name and process-id;
2. his accumulated cpu and io times;
3. his initial, current, and home directories;
4. every open file unit and its associated pathname.

If no arguments are specified, 'lfo' lists the information for as many processes as possible. The normal user may list only his own processes, while a system administrator may list any process on the system.

Examples

```
lfo 1 15 16 23
lfo snodgrass silverlips
lfo 3 upi 8
lfo
```

Messages

"pathname not obtainable" for files opened on a remote system.

Bugs

Locally supported.

Requires Georgia Tech modified Primos.

Usage

lib { <object_file> }

Description

'Lib' is used to concatenate the object code files produced by the 'as6800' and 'as8080' cross-assemblers. This is usually done to generate a library suitable for use by the link editor 'lk'.

If arguments are given on the command line, 'lib' concatenates the named files and places their concatenation on the file "lib.out". Otherwise, file names are taken from standard input, and their contents are concatenated and placed on "lib.out".

Examples

```
files .o$ | lib
lib rtrlib new_routine.o
```

Files

"lib.out" is always the output file.

Messages

"Can't open" for unreadable or unwritable files;

Bugs

Locally supported.

See Also

as6800 (3), as8080 (3), size (3), symbols (3), lk (3)

Usage

lk -(6800 | 8080) { [-(i | l | n)] file }

Description

'Lk' is used to link together the object code files produced by the 'as6800' and 'as8080' cross-assemblers. It produces a file of the same type as the input files, so the output of 'lk' may itself be linked with other code files.

If no arguments are given on the command line then they are taken from standard input. 'Lk' always writes its output to the file "l.out".

The first argument selects the machine that the object files have been compiled for: "-6800" refers to the Motorola 6800, while "-8080" indicates the Intel 8080. The remaining options select the mode the linker is running under and the input files that are to be linked. The available modes are:

-i	INCLUDE	This is the default mode. When in this mode, all object segments encountered in the files specified are linked together onto "l.out".
-l	LIBRARY	In this mode, the arguments are taken to be libraries, that is, concatenations of object code files made with 'lib'. Each segment is examined to see if it satisfies a previously unresolved external reference, and is linked in only if it does.
-n	NAMELIST	Any file read in this mode is considered to be a program that can be expected to be resident at the time that the output file is to be run on the target machine. Any entries encountered in this file's symbol table that satisfy a previously unresolved external reference are used to resolve that reference, but the segment itself is not linked in.

Examples

lk -6800 tpart1.o tpart2.o -n mux.o -l [libs]

Files

"l.out" for the output code file.

Messages

"Usage: lk ..." for invalid argument syntax.
Many other error messages, hopefully some of which are self-explanatory.

Bugs

Locally supported.

See Also

as8080 (3), intel (3), size (3), symbols (3)

Usage

lz [-i] [-l <page_size>]

Description

'Lz' post processes output from 'fmt' for the Xerox 9700 laser printer owned by the Georgia Tech Office of Computing Services. In particular, it outputs the necessary control statements to get actual boldfacing and italics. *These control statements are Georgia Tech specific.*

The 'fmt' input files should expect a page that is 100 columns wide by 87 lines down. The laser printer automatically supplies 1/2 inch margins on all sides, so the **.m1** through **.m4** values in 'fmt' need to be set appropriately, as well as the page and margin offsets, and the left and right margins.

'Lz' does actual underlining for text that is underlined. If the '-i' option is supplied, 'lz' will print underlined text in italics, instead.

The length of the output page can be given with the '-l' option. 'Lz' defaults to 87 lines per page.

Examples

fmt =fmac=/evl =doc=/guide/ed | lz >file_for_xerox

Messages

"Usage: lz ..." for improper arguments.

Bugs

Locally supported.

See Also

fmt (1), os (1), =fmac=/evl, =fmac=/evl2

Usage

memo [[[-t] <user>] [-d <display_cond>] [-e <erase_cond>]]

Description

'Memo' allows a user to send dated memos to himself or to another user. 'Memo' differs from 'mail' in that "display conditions" and "erase conditions" may be specified for memos; i.e., the user has control over when a memo is displayed and how long it will be displayed before it is deleted.

The simplest usage is just "memo". This form checks the current user's memo file, displays any memos whose display conditions yield "true", and deletes any memos whose erase conditions yield "true". Normally, a user would include this form of the 'memo' command in his "_hello" shell variable, so it would be executed whenever he enters the Subsystem.

The other forms of the command are used to send a memo. The "-t" ("to") option, followed by a valid user login name, specifies the intended recipient of the memo. The "-t" may be omitted if desired. If no user name is specified, then the memo is sent to oneself. The "-d" option, if given, must be followed by a boolean display condition (discussed below). When this condition is "true", the memo will be displayed. The default display condition is "always". The "-e" option, if given, must be followed by a boolean erasure condition (also discussed below). When this condition is "true", the memo will be removed from the user's memo file, regardless of whether or not it has ever been displayed. The default erasure condition is "always".

Display and erasure conditions are boolean expressions involving variables concerned with the current time and date. The allowable syntax is as follows:

```
expression -> secondary { '&' secondary }
secondary -> primary { '|' primary }
primary -> '~' '(' expression ')'
        |  '(' expression ')'
        |  relation
        |  'always'
        |  'never'
relation -> arithprim relop arithprim
relop -> '=' | '==' | '~=' | '<>' | '<' | '>' | '<=' | '>='
arithprim -> integer_constant
        |  symbolic_constant
        |  time_variable
symbolic_constant ->
           sunday | sun
        |  monday | mon
        |  tuesday | tue
```

```
                |  wednesday | wed
                |  thursday | thu
                |  friday | fri
                |  saturday | sat
                |  january | jan
                |  february | feb
                |  march | mar
                |  april | apr
                |  may
                |  june | jun
                |  july | jul
                |  august | aug
                |  september | sep
                |  october | oct
                |  november | nov
                |  december | dec
          time_variable ->
                |  month       # the current month, 1-12
                |  day         # the day of the month, 1-31 (usually)
                |  year        # the current year, e.g. 80
                |  dow         # the day of the week, 1-7
                |  hour        # the hour of the day, 0-23
                |  minute      # the minute of the hour, 0-59
```

Some examples of conditions might be helpful.

The condition "always" is always true. Thus, if used as a display condition, the memo will always be displayed. If used as an erase condition, the memo will be immediately deleted (although it may well have been displayed first). The condition "(month=March)&(day>3)" will be true only on days in March after March third (in any year). The condition "(dow=Friday)" will be true on any Friday, false otherwise. The condition "(month=feb)&(day=2)" will be true on Groundhog Day. The condition "(dow=mon)&(day=13)" will be true on those months in which Friday the 13th falls on Monday.

Examples

```
memo
```

```
memo system
 See about fixing 'lps'
 <Control-C>
```

```
memo -d "(month=mar)&(day<9)" -e "(month>=mar)&(day>=9)"
 See "In the Name of the Father" at Alliance Studio
 <Control-C>
```

Files

=extra=/memo/<login_name> for storing memos
=userlist= for verifying user names

Messages

"bogus character in expression" an unrecognizable character appeared in a display or erasure condition
"undefined variable" the parser encountered a variable that was not a symbolic constant or time variable, as defined in the lists above
"illegal user name" the named user is not in the Subsystem user list
"illegal user name or improper argument syntax" the command line could not be parsed properly
"can't open memo file" the user's memo file could not be opened
"memo file not available" the addressee's memo file could not be opened
"Usage: memo ..." command line was undecodable
"stack overflow" a condition was too complex to evaluate fully
And several self-explanatory messages from the expression parser.

Bugs

Needs a more concise syntax for expressing dates. Is subject to all the security problems of 'mail'. Lacks the ability to file copies of memos away when they are removed from the active memo file.

See Also

mail (1), to (1), stacc (1)

Usage

mkclist [-s]

Description

'Mkclist' lists the commands in "=lbin=", "=bin=", "=ubin=", and the internal shell commands, sorts them into order, and places them in "=ubin=/clist". This file is used by the backstop program as the file of commands to search through.

The template "=ubin=" must refer to the user's personal command directory. By default, the system-wide template "=ubin=" refers to "//=user=/bin".

The "-s" option causes 'mkclist' to omit "=ubin=" from the list of commands and place the resulting list in "=extra=/clist", thus creating the system default command list.

Examples

mkclist

Messages

"Usage ..." for improper arguments

Bugs

Bombs if =ubin= does not exist.

See Also

bs (5), bs1 (5), guess (5)

Usage

mon [<sample interval>]

Description

'Mon' is a program which continuously observes and displays the various Primos databases and certain statistics on performance, which it computes. It accepts one optional command line argument, along with several single character commands, once 'mon' is running. The command line argument is the number of clock seconds to wait between each sampling of the Primos databases. If omitted, the interval defaults to 30 seconds. The single character commands, during the program's run, determines which of several formats 'mon' will use to display the information. The commands are:

l	Use LONG format (default)
s	Use SHORT format
m	Use SHORT MEMORY format
c	CLEAR the screen and redraw the data
q	\
cntrl-p	>Quit
BREAK	/
x	Execute a PRIMOS command
?	Display the available commands

The "l", "s", and "m" commands adjust the format and information displayed. The "l" and "s" commands display the statistics with per process CPU usage in long and short formats respectively. The "m" command displays the statistics with per process memory usage (pages presently in memory).

The "c" command will clear and redraw the screen. This is useful after Primos messages or Primos commands have been executed via the "x" command (see below).

'Mon' will run continuously until interrupted by either a "q", a BREAK, or a control-p being typed. It will then position the cursor to the bottom of the screen and terminate.

The "x" command allows the execution of PRIMOS commands while 'mon' is running. It is similar to the Subsystem's "x" command. This feature allows system administrators to see how changing scheduling parameters affects system performance.

The "?" command will display the available single letter commands and then wait until any character is typed before continuing.

SYSTEM WIDE TIME STATISTICS:

For each of these, total time is displayed in hours, minutes seconds and hundredths of seconds, along with the change in time since last interval was displayed. The change is displayed in seconds and hundredths of seconds. The format of these statistics does not change with a change in display format.

Up Time	Total clock time since last boot.
User Cpu Time	Total cpu time used by normal user processes since last boot.
I/O Time	Total I/O time charged since last boot.

OTHER SYSTEM WIDE STATISTICS:

For Disk Accesses and Page faults, the total number since boot is displayed, as well as the number and rate (per second) during the last sample interval. The format of this information does not change with a format change.

Disk Accesses	Total disk accesses since the system was booted.
Page Faults	Total number of page faults since the system was booted.
Buffer Hit Rate	The number of disk records that were found in the in-memory associative buffers as a percentage of the total number of disk records requests during the last sample interval.

PER-PROCESS STATISTICS:

For each process, the user name, user number, total cpu time used, change in cpu time since last interval, and percentage of the cpu time used since the last interval are displayed. In addition to each logged in user process, data is displayed for other internal system processes that are active: the Clock process, two Idle processes (the system backstop processes; 1 for the main cpu, and one for the attached processor, if it exists), two Mpc processes (for line printers, mag tapes, and other unit record devices), the Amlc process (asynchronous line driver), the Smlc process (synchronous line driver), and the Ring process (the network driver).

Examples

mon

mon 15

Messages

"Usage: mon ..." for invalid argument syntax.
"Terminal type '<term_type>' not supported" when <term_type> is not supported by VTH.

Bugs

Since 'mon' can't lock the Primos databases that it reads the data returned isn't guaranteed to be 100% accurate. Bogus values may appear when a user logs in and out, but these will disappear during the next interval. Accuracy will improve with longer display times.

If more users than the screen can display are logged in, the higher user numbers may be cut off on the bottom of the screen. The short formats can display 50% more users, but even that isn't quite enough for 128 processes.

Since 'mon' requires breaks being enabled in order to stop, it ensures that breaks are enabled. If a user has pending breaks, 'mon' will terminate as soon as it enables breaks, rather than continuing.

See Also

x (1)

Usage

moot [-a <user>]

Description

'Moot' is a teleconference management program. It allows users to send messages to one another or to a group of users participating in a "conference". Messages may then be received automatically and reviewed at will. Authorized users may create and destroy conferences, add users to conferences, etc. 'Moot' is conversationally oriented, to reduce learning time.

To participate in any of the active teleconferences, type

moot

'Moot' will then ask you for your name. Respond by typing whatever variant of your name you prefer, but don't use any semicolons. We recommend using your calling name and your last name, to prevent conflicts. 'Moot' will then ask you for a password, which you must cite whenever you reenter the teleconference. (Note that the password is never printed on the terminal.)

If you want to see if a particular user is currently in 'moot' without getting in yourself, you can use the "-a <user>" option to check.

'Moot' allows you to abbreviate anything (user names, conference names, commands); simply use initial substrings of each word in the item to be abbreviated. For example, the command "add member" can be abbreviated (unambiguously) as "a m", "add m", "a memb", etc. (However, *do not* abbreviate your name the first time you enter the teleconference.)

If at any time you are unsure how to proceed, enter a line consisting only of a question mark; 'moot' will attempt to elaborate on the input it expects. Ambiguous command abbreviations will also provoke further information.

Once you have entered the conference successfully, 'moot' will prompt you for a command by printing the character ".". You may enter any of the following commands:

add conference

This command is used to create a new conference. 'Moot' will request a conference title and a status (open or closed, presently ignored), then prompt for the names of files to be used for storage of the conference. The file names supplied should be full pathnames, with passwords as necessary.

delete conference
: This command will delete a named conference, including all of its text storage files.

add member
: Membership in a closed conference (currently the only type supported) is by invitation only. The "add member" command allows a 'moot' user to join a particular conference, either as a full participant or just an observer (without the ability to submit messages to the conference).

delete member
: This command removes a member from a closed conference.

list conferences
: This command lists the names of all currently active conferences.

list users
: This command lists the names and times-of-last-entry for all teleconference users.

list members
: This command lists, for each member of the current conference (see "join"), the number of the last entry seen, the time of last entry, and the status (observer or participant).

enter
: When a user wishes to send a note to another user or to the members of a conference, he must first enter the text to be sent. This command prompts for a subject heading, cross-reference information, and finally for the text itself. Text entry continues until the user types a control-c (the standard Subsystem end-of-file signal). The text so entered fills the user's text buffer, which may be sent to another user with the "mail" command or submitted to a conference with the "submit" command.

edit
: This command invokes the Subsystem line editor 'ed', allowing the user to edit the text in his 'moot' text buffer or to read in text previously prepared and placed on a file. (See the *Introduction to the Software Tools Text Editor* for a tutorial on the use of 'ed'. 'Ed's commands are a subset of the screen editor's.) Note that the first two lines of the text buffer are used to store the subject and cross-reference information. Also note that to save changes made to the text buffer, you must issue a 'w' command before leaving edit mode with the 'q' command.

join

When a user wishes to review information from or send information to the other members of a conference, he must "join" that conference. When the "join" command prompts for a conference name, simply type the name of the conference to be joined. If no conference name is specified, the user is returned to command level (where he may send and review personal notes to other users).

review

The "review" command allows the user to re-examine messages sent by other participants in a conference. The messages to be reviewed are specified by typing a message number (e.g. "4") or a range of message numbers (e.g. "1-999") "Review" uses the Subsystem library routine 'page' to display the message text. In practice, this means that the display will be formatted for a CRT screen, with output stopping after each screenful of information. The user is then prompted for a command. The most commonly used commands are carriage return (to advance to the next page), -<pages> (to back up a given number of pages), +<pages> (to advance a given number of pages), and q (to quit displaying information). For a full description of acceptable commands, see the 'help' entry for 'pg' or 'page'.

index

The "index" command allows the user to get a list of messages and a brief description of the topic of each message. The user must be currently in a conference for this command to work. The messages are listed with their sequence number within the conference, name of the sender, subject of the message, and date of submittal to the conference. The message list is formatted for the CRT screen similar to the output of "review"; the Subsystem routine 'page' is used to display the list a screenful at a time.

submit

"Submit" takes the contents of the user's text buffer (created by "enter text") and submits it to the current conference. The text buffer is not altered, so multiple "submits" may be used to send messages repeatedly (say, to different conferences).

authorize

This command allows a user to change the operations that another user may perform. The operations are listed, one at a time, along with the current value (Y or N for yes or no); the correct response is "y" to enable an operation,

"n" to disable it, "d" to set it to the default value, or simply carriage return to leave it unchanged.

status
: The "status" command lists the names of all currently logged-in 'moot' users and the name of the current conference, if one has been joined.

mail
: The "mail" command allows the user to send the contents of his text buffer as a private communication to another user. The addressee will receive the letter automatically; he may review it later with the "letter" command.

letter
: This command is used to review personal notes sent by the "mail" command. The index numbers of the notes to be reviewed are specified in the same manner as those for the "review" command. The notes on output pagination under "review" also apply to "letter."

quit
: When the "quit" command is issued, the user is logged out from 'moot' and returned to the Subsystem.

In general, multiple inputs may be typed ahead by separating them with semicolons. However, the first parameter of a command must not be separated from the command; for example, you should type "join <conference>" or "review <entry>" or "mail <user>" without separators.

Examples

```
moot
Please enter your name: George Burdell
Please enter your password:
Welcome to the Moot.
.enter
Subject: bogus messages
Xref: none
Text:
This is an entirely bogus message.
<control-c>
.mail
Addressee: a a
.list conferences
Empirical Metaphysics
.join empirical metaphysics
.review 1-1729
(volumes of stuff would appear here)
.q
```

Files

Everything in =extra=/moot.u

Messages

Too many to document at the moment.

Bugs

Null inputs match anything, since the null string is an initial substring of every string.

There is no way to alter a user's name or password.

'Moot' uses semaphore 1 for mutual exclusion; if this semaphore is messed up, 'moot' will fail in fairly unpredictable ways.

When 'moot' fails, it tends to prevent the user from subsequently logging in.

Inputting a TAB character tends to hang 'moot' in an infinite loop with breaks disabled, thus preventing the user from stopping the loop.

See Also

sema (1)

Usage

mot <object_file> [<relocation>]

Description

'Mot' takes the output of the Motorola 6800 cross-assembler ('as6800'), relocates it to the desired starting address (0000 by default), and generates a Motorola standard format object tape on its first standard output.

'Mot' is useful for down-line loading assembled code to development systems equipped with the MIKBUG/MINIBUG ROM (or any of the many other commercially available ROM's using the same load format).

Unlike MIKBUG's Punch command, 'mot' generates an "L" before and an "S9" after the object text, thus permitting convenient loading of multiple files without operator intervention.

Examples

```
mot mux
mot highloader 16384
```

Messages

"Can't open" for unreadable object file.
"Usage: mot ..." for missing arguments.
"badly formed code file" when an error is found in the code file.

Bugs

Locally supported.

See Also

as6800 (3), lk (3), intel (3)

Usage

mv <source> <destination>

Description

'Mv' moves a file from one location or name to another, by copying it and then deleting the original. The deletion action is not performed if the copy was not successful; the copy is left untouched if the deletion could not be performed.

Examples

```
mv //my/file //your/file
mv old current
```

Messages

"Usage: mv ..." for improper command syntax.
"Can't copy ..." if the copy operation was unsuccessful.

Bugs

'Mv' can not move whole directories, use 'cp' with the "-s" option.

See Also

cp (1), del (1), if (1)

Usage

nodes

Description

'Nodes' prints all network node names in the current Primenet configuration, if networks are enabled.

Messages

"network not configured" for no network.
"x$stat error. Shouldn't happen" for unexpected network errors.

Examples

nodes

Bugs

Locally supported.

See Also

ns (3), nstat (3)

Usage

ns

Description

'Ns' prints the status of all active virtual circuits. For each virtual circuit, it prints the name and process id of the user holding it, the port number, the virtual circuit number, and the system to which it connects.

Messages

"network not configured" for no network.
"x$stat error. Shouldn't happen" for unexpected network errors.

Examples

ns

Bugs

Locally supported.

See Also

nodes (3), nstat (3)

Usage

nstat <node> [us | di | ne | wh | lo]

Description

'Nstat' uses the X.25 Primenet calls to send a request to a server phantom on the destination system. Currently, <node> may be the systems "gt.a", "gt.b", "gt.c", and "gt.d". The remote server process then executes a status command and routes the information back through the virtual circuit to the calling program. Options include:

us (default) causes the equivalent of the Subsystem 'us' command on the other system; same as a Primos STATUS USERS command.

di causes a status of active disks; same as a Primos STATUS DISK command.

ne causes a status of the network; same as a Primos STATUS NET command.

wh causes a verbose status listing of users; equivalent to the Subsystem 'who' command being executed for the named system.

lo causes the server phantom on the named system to logout. This command may only be issued by user "system".

Examples

```
nstat gt.b us
nstat gt.e di
```

Files

The server process uses the file "=temp=/netstat", and some files in the account(s) "//nstat"

Messages

"Usage: nstat ..." for improper arguments.
Various other messages depending on network status.

Bugs

Locally supported; experimental

See Also

Primenet guides, X.25 routines

Usage

otd <file_name>

Description

'Otd' is a program that reads relocatable binary files and prints their contents in human-readable form. It is useful for analyzing the output of high level language compilers. You can use 'otd' to compare the quality of two compilers' code generation phases, or for debugging your own compilers.

Examples

otd hello.b

Messages

"usage: otd ..." if called with not arguments.

"filename: can't open" if it can't open the file.

"bad object file" for a an object file format it does not understand.

"inconsistent block size"

"unrecognized block type ..."

"block size ... exceeds buffer space"

Bugs

Does not read its standard input port if called with no arguments.

See Also

VCG User's Guide

Usage

p4c

Description

'P4c' is the third release of the Georgia Tech Pascal compiler for the Prime 400. The present version takes Pascal source code from its first standard input and produces a listing on its first standard output, and an equivalent Fortran program on its second standard output.

The Pascal language is fully described in *Pascal Users Manual and Report* , Second Edition, by K. Jensen and N. Wirth, Springer-Verlag, 1976. Most error diagnostic numbers produced by the compiler conform to those listed in the book. The remainder of this description covers the important differences between the language described in the Report and the one implemented at Georgia Tech.

Options

The behavior of the Pascal compiler is controlled through the use of special comments of the form:

(*$<option sequence> any comment *)

An option sequence consists of a series of option settings separated by commas with no embedded blanks. Available options include:

B+ Turn on code generation. (Generated Fortran code is written to standard output port two.)
B- Turn off code generation.
L+ Turn on source listing. (The listing is written to standard output port one.)
L- Turn off source listing.
P+ Include pointer validity checks in the generated code.
P- Do not include pointer validity checks.
R2 Include full range and subscript checks in the generated code.
R1 Include optimized range and subscript checks (assume subrange variables contain valid values).
R0 Do not include any range or subscript checks.

Default values for these options are equivalent to an occurence of "(*$B+,L+,P+,R1*)" at the beginning of a program.

Any number of option setting comments may appear in a program. Note, however, that turning off code generation for any portion of a program may render the generated code generally unreliable.

Identifiers

The identifiers in a program may be of arbitrary length and may include the underscore character ("_") and upper and lower case letters (no distinction is made between upper and lower case). However, only the first eight characters of an identifier are significant.

Keywords

Keywords are recognized in any mixture of upper and lower case.

Packing

The 'packed' attribute that may be applied to arrays and records is not implemented, with the single exception of character strings. A quoted character string has the type

packed array [1..<length of string>] of char

This is the only type of character array that may appear in an output procedure call (e.g. write, writeln).

In addition, the standard procedures 'pack' and 'unpack' are not implemented and cause the compiler to issue a diagnostic.

The 'packed' keyword may be used freely without error in the declaration of types and variables; however, it currently has no effect except as described in the case of character arrays.

Files

The use of user defined files has been implemented. Currently, the only restriction is that fields within a record may not be defined as files. Remember that a Pascal file must be passed as an argument in the program heading if it is to exist after the program terminates execution. This is especially important when using pathnames, since a file must be permanent to be linked to a pathname.

The use of pathnames has been implemented by an extension of the RESET and REWRITE procedures. To use a pathname, a file of the correct type must be defined within the program and this file must be an parameter in the 'program' heading. Therefore, the complete syntax for RESET or REWRITE is :

RESET(<filename>{,"<pathname>"});

Note that the pathname is enclosed within *double* quotes. The pathname may be up to 100 characters in length and may contain any ASCII character. Two consecutive double quotes act as a single double quote. All references to the pathname (in read, write, etc) are made through <filename>.

There are also four predefined text files, two for input and two for output, that the programmer has access to. For input, the files 'input' and 'keyboard' are available and correspond to Subsystem standard input ports one and two, respectively. For output, the files 'output' and 'prr' are available and correspond to Subsystem standard output ports one and two, respectively.

Newly added is the predefined file 'keyboard' (in the place of prd). This file is used for reading from the terminal. (note that the name 'keyboard' was taken from the UCSD PASCAL compiler). This file differs from standard Pascal files in that it does not incorporate a lookahead buffer. Since input does incorporate a lookahead buffer, the program will expect a character before it will start to execute if 'input' is used. To make these files accessable within a program, it is only necessary to include their names as formal parameters in the 'program' declaration. For example

```
program test (input, output, keyboard, prr)
```

would make all four files available within the program.

The standard procedure PAGE has been implemented. This procedure prints a form feed in the specified file (as opposed to a '1' is column 1 as described in the standard). This allows the user to insert form feeds as needed in a file of type TEXT (file of char). If the file specified as an argument is not of type TEXT, an error will be issued.

Remember: There are essentially two types of files in PASCAL; local files, those that are local to the program or a particular procedure or function, and external files, which are files that exist before and/or after the program is executed. In order that a file be external, it must be passed as a parameter to the 'program' in the program heading. If pathnames are used, the file that the pathname is linked to must also be in the 'program' heading (NOT the pathname itself!).

The Heap

Two standard procedures, 'mark' and 'release' are provided for rudimentary management of the heap. Each takes a single parameter whose type must be of the form

^<any type>

'Mark' copies the current value of the heap pointer into the argument; 'release' does the opposite: it copies the contents of the argument into the heap pointer. This scheme is only effective in an environment where storage is allocated and released in a (more or less) last-in-first-out manner (which is exactly the situation in a recursive descent compiler).

Procedures/Functions as Parameters

Neither procedures nor functions may be passed as parameters to other procedures or functions.

Non-local Gotos

'Goto' statements whose target label is not within the scope of the current procedure are not supported.

Procedures Time and Date

Two new (non-standard) procedures have been implemented. These are the procedures time and date. Both procedures take a single argument which must be an (unpacked) array of 8 characters. The variable into which the corresponding value is to be returned must not be subscripted in the procedure call. An example is:

```
Time (thetime);
```

where the variable thetime is defined as:

```
thetime: array [1..8] of char;
```

returns the current time in variable thetime. Procedure DATE works the same way. The time returned is in HH:MM:SS 24-hour format and the date is in MM/DD/YY format.

Execution of Pascal Programs

The full journey from Pascal source code to executable program involves three stages: compilation by 'p4c', a trek through the Fortran compiler, and linking/loading by 'ld'. The following sequence of commands illustrates a possible scenario:

```
copy.p> p4c >copy.l >copy.f
fc copy.f
ld copy.b -l p4clib -o copy
```

Special notice should be taken of the "-l p4clib" argument sequence in the 'ld' command; it is mandatory for the completion of linking.

This procedure may be abbreviated to a single command through the use of the 'p4cl' command. Detailed information on its usage is available from 'help'.

Examples

```
prog.p> p4c >prog.l >prog.f
xref.p> p4c 2>xref.f | sp -f
```

Messages

Numbered error diagnostics for syntactic or semantic errors. Messages corresponding to the numbers are given in the User's Manual.

Bugs

Produces code that is too huge and too slow to be considered useful by anybody.

Locally supported.

See Also

p4cl (3), *Pascal User's Manual and Report*

Usage

p4cl <source> <ld arguments>

Description

'P4cl' is a shell program that invokes the necessary sequence of commands to convert a Pascal source file into an executable program. The first argument, <source> specifies the file that will contain the final executable version of the program. The Pascal source code is assumed to be in the file named <source>.p. If 'p4cl' is invoked with no <source> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'. Any further arguments appearing on the command line are passed directly on to the loader, 'ld'.

The Pascal compiler, 'p4c', is first invoked to convert the contents of the source file into an equivalent Fortran program and write it into the file named <source>.f. A compilation listing is also generated on the file <source>.l. (This listing contains Fortran forms control characters and may be printed with 'sp' using the "-f" option.)

The Fortran compiler is then invoked, using 'fc', to produce a binary relocatable version of the program in the file <source>.b, which is then prepared for execution by 'ld'.

Examples

```
p4cl copy
p4cl xref -t -m xref.m
```

Bugs

Locally supported.

See Also

p4c (3), fc (1), ld (1)

Usage

passwd [-s[<depth>]] <non-owner pwd> { <pathname> }

Description

'Passwd' is used to change the non-owner password of one or more directories (UFDs) in the file system. <Non-owner pwd> may consist of up to six characters; shorter strings are padded with blanks, and lower-case letters are converted to upper-case.

In order to use this command successfully, one must be able to attach to the named directories with owner privileges. In a standard Primos environment, this means the current owner password must be included in the pathname for each specified directory. In a Ga. Tech Primos environment, this means that the user must currently own the specified directories, or they must be public.

The "-s" option, if specified, causes 'passwd' to traverse the file system subtree rooted in the named directory, changing the non-owner password of each directory it encounters. The depth of this traversal may be limited by appending a positive integer to the "-s" (e.g., "-s3").

Specifying no <pathname> arguments is the same as specifying the pathname of the current directory.

Examples

```
passwd "" =mail=
passwd ""
passwd secret =src= =src=/lib
passwd -s xyzzy =aux=
```

Messages

"Usage: passwd ..." for bad argument syntax.
"password too long" for illegal <non-owner pwd>.
"<pathname>: can't change password" for not being owner of <pathname>.
"<pathname>: bad pathname" for not being able to access the directory containing <pathname>.

Bugs

In a standard Primos environment, 'passwd' sets the owner password to spaces when it changes the non-owner password.

See Also

cd (1), chat (1), sacl (1), chown (3), Primos spas$$

Usage

phone [<person> | <business>]

Description

'Phone' finds the telephone number (or numbers) belonging to a given person or business, provided the number has been placed in the on-line phone directory.

If no person or business is given, the entire phone directory is listed.

Examples

```
phone ack radio
phone ed rupp
phone
```

Files

=phonelist= for directory of telephone numbers

Bugs

Locally supported.

Usage

ptar [-xvt] [-f <file>]

Description

'Ptar' reads input in the form produced by the Unix 'tar' (**t**ape **ar**chiver) program, and rebuilds the directory subtree(s) encoded on the tape, if possible. 'Ptar' *must* have write permission in the current directory in order to create the files and directories it is dumping from the tape.

'Ptar' knows about the Unix "." (current directory) and ".." (parent directory) directory structures. In the first case, the leading "./" is simply stripped off the file name which is currently being recreated. Multiple occurrences of "./", *at the beginning of a file name*, are allowed, and will be stripped off. In the second case, the leading "../" is replaced with a "\". Each occurrence of "../" at the beginning of a file name is replaced with a "\". Absolute path names (names that start with a "/"), have another "/" prepended, in the hope that there will be an appropriate directory out in the file system somewhere. File names beginning with a digit have an "_" prepended to them, since the Primos file system will not allow file names to start with a digit. In general, it is best if one's 'tar' tape contains only relative path names. 'Ptar' checks first for leading "./"s, and then for "../", so if you have file name that starts with ".././", 'ptar' will be fooled. Also, one should not have occurrences of either "./" or "../" in the middle of one's file names.

'Ptar' assumes that you are dumping *text* files. Therefore it always sets the parity bit, to conform to Prime's convention of using mark parity. Unix binaries would be of limited use on a prime anyway.

When 'Ptar' finishes, it leaves a file in the current directory, called "_detab.tar.files". This is a command file which will remove tabs from the files just dumped. Tabs are replaced by eight blanks, which is the Unix default. "_detab.tar.files" turns on shell tracing, so that you can make it sure it is transforming the files properly.

The following options are available:

-t Table of contents. This option reads through the 'tar' tape, printing out the file names and their sizes in bytes, but does not dump the files.

-v Verbose. This option will print file names and sizes as they are being dumped. It is wise to use this option.

-x Extract. This is the default behavior. In fact,

putting this option on the command line has absolutely no effect at all. Use the "-t" option if you just wish to see what is on the tape.

-f Use the given file as input. When this option is given, the next argument is used for input. A file name of "-" is taken to mean the standard input. If no files are given, 'ptar' will issue an error diagnostic and die.

Examples

```
x as mt0; mt -r -cb -b512 | ptar -xv; x un mt0
tarfile> ptar -t
ptar -xvf tafile
```

Files

_detab.tar.files to replace tabs with blanks in the newly extracted files.

Messages

Various self-explanatory messages if it can't create directories or files.

Bugs

Will overwrite "_detab.tar.files" each time.

Cannot take more than one file on the command line for the "-f" option.

Cannot dump Prime directory trees into a 'tar' format tape.

See Also

mt(1), tar(1) in the *UNIX Programmer's Manual*.

Usage

pwd

Description

'Pwd' is supplied for Unix users who have not learned to type "cd -p".

Examples

pwd

See Also

cd (1)

Usage

raid [-(a | p)]

Description

'Raid' allows a user to examine bug reports submitted with the 'bug' command. 'Raid' expects to find a variable named 'lastbug' in the global environment (you can create it by entering "set lastbug = 0") containing the number of the last bug report examined. If the "-a" option is present, 'raid' prints all bug reports; otherwise it prints only those reports that have not been seen. If the "-p" option is present, 'raid' spools the bug for printing; otherwise, they are displayed on the terminal with 'pg'.

If you wish to be notified of new bug reports as they occurs, place the following command in your "_hello" variable, or in a shell program that is executed by "_hello" variable:

```
if [eval lastbug < [=ebin=/bugn]]
   echo "You have bugs."
fi
```

Examples

```
raid
raid -ap
```

Files

=bug=/r??? for storage of the bug report
=bug=/s??? for storage of the user's environment at the time the bug was reported

Messages

"Usage: raid ..." for improper command syntax

See Also

bug (3)

Usage

rcl <program> [<ld arguments>]

Description

'Rcl' is a shell file that causes the specified Ratfor program to be preprocessed, compiled and loaded. The source file is expected to be named <program>.r and the output object code is named <program>. Default options are used on all processors. If 'rcl' is invoked with no <program> argument, it automatically processes the last program edited, since it shares the shell variable 'f' with the shell program 'e'.

If <ld arguments> are present, they will be presented to the loader following the main program binary file. This allows the inclusion of subroutine and library files in the object program.

Examples

```
rcl profile
rcl math -l vswtmath
```

Bugs

Will not be supported after Version 7. Use 'rfl' and the extended preprocessor 'rp' instead.

See Also

rf (3), fc (1), ld (1), pl1cl (1), rfl (1), rp (1)

Usage

```
rf [-t | -p | -c]
   [ [-o <output file>] <input file> { <input file> } ]
```

Description

'Rf' is an extended version of the Ratfor preprocessor described in *Software Tools*. For a complete description of 'rf', see the book. For a summary of Ratfor constructs, see *Software Tools* or the *Ratfor Programmer's Guide*.

Note that 'rf' has been superseded by 'rp', and will not be supported beyond Version 7 of the Subsystem.

The following is a summary of the differences between standard Ratfor and the language accepted by 'rf':

1. When supplied with no <file> arguments, 'rf' operates as a filter, taking its input from standard input and writing its output on standard output.

   When any <input file> arguments are given, 'rf' takes its input from the specified files. If "-o <output file>" is specified as the first argument, output is directed to <output file>. Otherwise, ".f" will be appended to (or will replace a ".r" at the end of) the first <input file> name to construct the name of the output file.

   The table of definitions is not cleared between the input files, so definition files can be invoked without being added to the source file(s).

2. Include statements are of the following format:

   ```
   include '<filename>'
   include "<filename>"
   include <filename>
   ```

   where <filename> may be any valid Subsystem pathname. If, however, the <filename> contains any non-alphanumeric characters, one of the quoted forms **must** be used.

3. 'Rf' will accept upper or lower case input, and will map lower case input to upper case, except in quoted strings. Definitions are case sensitive, however; a lower case token appearing in the left-hand side of a 'define' statement will be replaced only if it appears as a lower case token in the text.

4. 'Rf' uses the tilde (~) for "not". Thus, .ne. would be represented as ~=. An exclamation point may be used instead of the tilde for the benefit of Teletype users.

5. 'Rf' will accept and discard an underline appearing within an identifier. For example, 'get_arg' will be interpreted as 'getarg'. The underline may not begin the identifier.

6. 'Rf' allows arbitrarily long identifiers and replaces them with unique six-character names acceptable to the Fortran compiler. The generated names retain the first character so as not to disturb implicit typing.

7. The 'andif' and 'orif' constructs suggested in the exercises in Software Tools have been added. Two syntactic forms are available to invoke these constructs:

   The '||' and '&&' operators may be used in the condition part of an if statement to specify orif and andif, respectively. If this syntax is used, the condition **may not** contain nested parentheses.

   The keywords 'orif' and 'andif' may be used explicitly. With this syntax, an orif or andif statement may be preceded only by an if statement or another orif or andif statement.

8. Multilevel break and next statements have been implemented. Syntax is:

   ```
   break [ <integer> ]
   next  [ <integer> ]
   ```

   The integer, if specified, determines how many surrounding loops will be terminated or continued, respectively. If omitted, 1 is assumed.

9. A case statement has been implemented. Syntax is as follows:

   ```
   case <variable> <compound statement>
   [ else <statement> ]
   ```

   <Variable> is an integer variable whose value selects which of the statements in the <compound statement> is to be executed: if the variable's value is 1, the first statement is selected, etc. At most one of the statements is performed. If there is no statement corresponding to the value of the variable, the <statement> after "else" is executed if the "else" is present; otherwise, execution resumes after the <compound statement>.

   Currently, case statements may be nested to a depth of 10 and may contain as many as 392 alternatives each.

10. The string declaration has been implemented. Syntax is

```
string <variable> <quoted string>
```

String declarations must appear before the first executable statement of a program unit.

11. The "-p" option may be used to permit profiling studies to be performed on Ratfor programs. The option causes insertion of code that, at run time, will record statistical information on program performance. Profiled programs will behave in exactly the same way as they would normally, except that they will run **VERY SLOWLY**. The utility command 'profile' may be used to print up a neat report of a profiled program's performance. The files "timer_dictionary" and "_profile" are generated by 'rf' and the profiled program, respectively.

12. The "-t" option may be used to cause an execution-time trace of the Ratfor program. The trace consists of an indented listing which indicates the point of call and return for each subprogram.

13. The "-c" option may be used to perform statement-level profiling studies on Ratfor programs. The option causes insertion of code that counts the number of times each statement is executed and then writes the totals to the file "_st_profile", where they may be read and summarized by the utility program 'st_profile'. The statement count statistics gathered by this option are inherently more reliable (and probably more useful) than the time measurements gathered by the "-p" option.

Examples

command line format:

```
rf -p prog.r
rf -o prog prog.r
prog.r> rf -t >prog.f
```

orif and andif:

```
if (c1 || c2 && c3 || c4)   statement1
else   statement2

if (c1) andif (c2) orif(c3)   statement1
else    statement2
```

break and next:

```
repeat {
   while (condition)
      if (c1)  break 2    # terminate repeat
      else     next 2     # continue repeat
  } until (c2)
```

```
case:
     case i {                          goto L4
        stmt1       is       L1        stmt1; goto L5
        stmt2   equivalent   L2        stmt2; goto L5
        stmt3       to       L3        stmt3; goto L5
        }                    L4        goto(L1,L2,L3),i
     else                              stmt4
        stmt4                L5        continue

profiling:
     rf -p prog.r; fc prog.f; ld prog.b -o prog
     prog
     profile | sp

     rf -c prog.r; fc prog.f; ld prog.b -o prog
     prog
     st_profile prog.r | sp
```

Files

"timer_dictionary" for programs compiled with "-p" option.

Messages

Extensive. Most are self-explanatory. See *Software Tools* for a complete list.

Bugs

<integer> must be followed by a NEWLINE in the multi-level break and next statements.

'andif' and 'orif' constructs may not appear in 'while', 'for' or 'repeat' statements.

When the statement-level profile option ("-c") is used, the count displayed for each line is the total number of times all statements on that line were executed; thus, for example, the line

```
      repeat c = c + 1; until (prime (c) == YES)
```

will have a displayed count equal to three times the actual number of loops (once for the 'repeat', once for the assignment, and once for the test). In addition, note that only the statements in the uppermost level file are counted; any code compiled as a result of an 'include' will contribute to the total for that 'include'.

See Also

profile (1), st_profile (1), rp (1), *Software Tools*

Usage

rsa (-i | -e <correspondent> | -d)

Description

'Rsa' is a simplified implementation of an RSA (Rivest-Shamir-Adleman) public-key cryptosystem. While interesting as a novelty, it does not provide sufficient security to resist an informed attack.

'Rsa' has three options. The "-i" (initialize) option must be selected by a user before any other users can send him encrypted information. The "-e correspondent" (encipher) option is used to encrypt standard input to standard output using the public key of the named user. (In a practical system, only the named user would then be able to decrypt the result, using his private key.) The "-d" (decipher) option is used to decrypt standard input to standard output using the private key of the current user. Thus, if the current user has login name "BOZO", the network

rsa -e bozo | rsa -d

effects an identity transformation.

Further information on public-key cryptosystems in general and the RSA algorithm in particular can be found in the following references:

Hellman, Martin E. "The Mathematics of Public-Key Cryptography," in *Scientific American*, Vol. 241, No. 2, pp. 146-157, August, 1979

Rivest, R. L., Adi Shamir, and Len Adleman *On Digital Signatures and Public-Key Cryptosystems*, Report MIT/LCS/Tm-82, Laboratory for Computer Science, Massachusetts Institute of Technology, April, 1977

Rivest, R. L., A. Shamir, and A. Adleman "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," in *Communications of the ACM*, Vol. 21, No. 2, pages 120-126, February, 1978.

Examples

```
rsa -i   # initializes public and private key files
plaintext> rsa -e system >ciphertext
ciphertext> rsa -d >plaintext
rsa -e bozo >>=extra=/mail/bozo
```

Files

"=varsdir=/.rsa_encipher" for public-key information;
"=varsdir=/.rsa_decipher" for private-key information.

Messages

"Usage: rsa ..." for invalid argument syntax.
Various self-explanatory messages if key files are not present or unreadable.

Bugs

32 bit arithmetic is insufficient to guarantee security.

Locally supported.

See Also

Subsystem Mathematical Function Library ('vswtml')

Usage

rtime <command>

Description

'Rtime' is a shell program that will determine the CPU time used by the execution of a particular command. The command to be timed should be specified as the set of arguments to 'rtime'; quoting of the command to prevent premature evaluation of any command-language metacharacters is recommended.

The given command is used in a function call, and its output on standard output one redirected to /dev/tty if no other i/o redirection for standard output one is specified. This may cause problems if the command uses all three standard output ports.

Examples

```
rtime rp rp.r
rtime "stuff> filter >nonsense"
```

Messages

"Usage: rtime ..." for improper command syntax.

Bugs

Redirection problem mentioned above.

See Also

hp (1), ctime (1)

Usage

scroll

Description

'Scroll' loads a scrolling terminal program to the GT40. It is presumed that the GT40 is ready for loading gtld format files, which usually implies that FOCAL must not be running.

Examples

scroll

Bugs

Will not load the scrolling terminal on top of FOCAL.

Locally supported.

See Also

focld (3)

Usage

setime [-d mmddyy] [-t hhmm]

Description

The 'setime' command is an interface to the SWT 'ring' process which allows validated users to change the time of day on all systems that are running 'ring'. If the "-d mmddyy" argument is present, the current month, day, and year is set to the given value, otherwise the date remains unchanged. If the "-t hhmm" argument is present, the current time of day is set to that value, otherwise the time of day remains unchanged. At least one of the arguments must be present.

If the current time of day is being reset, the 'setime' command executes immediately. Otherwise, 'setime' pauses until the beginning of the next minute to complete execution.

Examples

setime -d 030184

setime -t 1400

Messages

Cannot transmit SETTIME request
: Something interfered with the transmission of the SETTIME command to the 'ring' process. This should never happen.

Networks are not configured
: The system is not configured to support PRIMENET.

Request to <system> failed
: The attempt to set the time of day/date on system <system> failed.

Request to <system> succeeded
: The attempt to set the time of day/date on system <system> succeeded.

Ring connection has been terminated
: The connection to the 'ring' process has been cleared.

Setime complete
: The SETTIME command has been successfully attempted on all systems in the ring.

SETTIME request initiated
: The SETTIME command has been transmitted to the 'ring'

process.

The first day of the month must be at least 1
0 is not a valid day of the month.

The month must be between 1 and 12 (inclusive)
The only valid months are 1 through 12.

The hour must be between 0 and 23 (inclusive)
The only valid hours are between 0 and 23.

The minute must be between 0 and 59 (inclusive)
The only valid minutes are between 0 and 59.

Usage: setime [-d mmddyy] [-t hhmm]
Some argument was incorrectly specified.

Unable to connect to ring node
The current system is not running a 'ring' process.

You are not validated to SETTIME
Your user number is not allowed to use the SETTIME command.

<month> has only <count> days
The number of days specified is not correct for the given month.

Bugs

Will not work if the current system is not running 'ring'.

Is inherently inaccurate because of the time required for the SETTIME request to go around the ring.

See Also

broadcast (3), execute (3), terminate (3)

Usage

shar <file_spec> { <file_spec> ... }

Description

'Shar' (**Sh**ell **Ar**chiver) is an alternative method of arranging for long term storage of files that need to be kept together for some purpose, or files that will not be frequently accessed.

'Shar' reads the files named on the command line, and writes the "shell archive" on its first standard output port. A shell archive is a file containing shell commands and text, that when executed as a command by the Software Tools Subsystem shell, will reproduce the files used to create the archive.

See the help on 'cat' for a full description of <file_spec>.

'Shar' can even be used to archive other shell archives, as long the component archives do not contain a file by the same name as the archive.

To extract the files, simply execute the archive file as a command. As each file is extracted, its name is written out to the terminal.

Examples

```
shar [files .r$] [files .d$] >prog.shar   # to create an archive
prog.shar       # to extract the files in the archive
```

Messages

"<file>: can't open" when it can't open a file.

"usage: shar ..." if called improperly.

Bugs

May do bizarre things with non-text files.

Can't recursively archive sub-directories.

See Also

ar (1), cto (1), cat (1)

Usage

```
show [-m | -o] { <file spec> }
     <file spec> ::= <pathname> | -[<stdin_number>]
                   | -n(<stdin number> | <pathname>)
```

Description

'Show' concatenates the contents of the files specified in its argument list, replacing any imbedded non-printing characters with printable representations, and writes the result on its first standard output. Normally, the non-printing characters are displayed as digraphs consisting of a caret ("^") followed by an uppercase letter or punctuation character. However, if the "-m" option is specified, non-printing characters are represented as their ASCII mnemonics enclosed in angle brackets (e.g., a NEWLINE would be represented as "<LF>"). If the "-o" option is specified, the characters are displayed as a caret followed by three octal digits.

Input files may be specified in any of several ways:

<pathname>	an ordinary Subsystem pathname.
-<stdin number>	a dash followed by a decimal number, 'n', designates the 'n'th standard input. 'n' must be a legal standard input number.
-	this is the same as specifying "-1" (i.e., standard input 1).
-n<stdin number>	"-n" followed by a decimal number 'n' indicates that the names of the files to be concatenated are to be read from the 'n'th standard input.
-n	this is the same as "-n1".
-n<pathname>	the names of the files to be concatenated are to be read from the named file.

If no arguments appear, input is read from the first standard input port.

Examples

```
show weird_file
files .r$ | show -m -n
```

Messages

"Usage: show ..." for invalid argument syntax.
"<pathname>: can't open" if a specified file can't be opened for reading.

See Also

cat (1), copy (1), print (1), pr (1), tee (1)

Usage

size (-6800 | -8080) <object_file>

Description

'Size' is used to determine the number of bytes of object code in all the code segments of the named object file. The mandatory first argument tells whether the object file was generated by the Motorola 6800 cross-assembler ('as6800') or the Intel 8080 cross assembler ('as8080').

Examples

size -6800 .o
size -8080 mux

Messages

"Usage: size ..." for invalid argument syntax.
Warnings if the object file could not be opened or is not in standard format.

See Also

symbols (3), as6800 (3), as8080 (3), lk (3)

Usage

sol

Description

'Sol' uses the Subsystem virtual terminal handler (VTH) to play a standard or casino version of solitaire on the screen of a video terminal. When 'sol' is executed, it displays the commands used for playing the game.

It is included to demonstrate the use of the VTH package.

Files

=vth=/?*

Examples

sol

Messages

"Terminal type '<term_type>' not supported" when <term_type> is not supported by VTH.

Bugs

Only works on video terminals that are supported by VTH.

Can be mildly addictive.

Usage

```
sprint { <options> } { <file_spec> }
   <options> ::=  -c <copies>       |
                  -h <horiz_space>  |
                  -j                |
                  -l <length>       |
                  -s                |
                  -v <vert_space>   |
                  -x
   <horiz_space>  ::=  0 | 1 | 2 | ... | 14 | 15
   <vert_space>   ::=  1 | 2 | 3 | ... | 15 | 16
```

Description

'Sprint' prints files on the user's terminal, making the assumption that the terminal is a NEC Spin Writer model 5520. Printing is done bi-directionally, optimizing motion of the print head and platen as much as possible.

The following options are available to control 'sprint's behavior:

-c If present, the next argument must be an integer; 'sprint' will produce the specified number of copies of each file that it prints.

-h If present, the next argument must be an integer in the range 0 to 15. This option controls the horizontal spacing between characters measured in 1/120ths of an inch.

-j 'Sprint' will cause a page eject following each file (or copy of a file, when multiple copies are specified). Normally, no extra space is inserted between successive files or copies of the same file.

-l If present, the next argument must be an integer; 'sprint' will use that number as the number of lines per physical page. It is important that this number match the form length selected on the terminal itself, else anomalous behavior may result. 'Sprint' assumes there are 66 lines per page, corresponding to the standard 11 inch form length.

-s This option causes 'sprint' to pause at the top of each page and sound the terminal's audible alarm, allowing the user to insert a new piece of paper. Printing continues when the user types an ACK character (ctrl-f).

-v If present, the next argument must be an integer in the range 1 to 16. This option controls the vertical spacing between lines measured in 1/48ths of an inch. For example, "-v 2" indicates a spacing of 2/48ths of an

inch between lines. To obtain line and a half spacing, text should be triple spaced and "-v 4" specified on the "sprint" command (it is also necessary to set the page length to 132 with the 'fmt' command ".pl" and specify a page length of 132 for 'sprint').

-x This option prevents the initial page eject. This option is useful when printing on special forms (mailing labels, etc.).

The remaining arguments specify files to be printed. Most often, one or more pathnames will be given, indicating that the named files are to be printed, or there will be no other arguments, indicating that input is to be read from standard input. The full syntax of the <file_spec> construct is described in the entry for 'cat' in section 1 of the Reference Manual.

It is assumed that the paper has been mounted so that a form feed will advance to the first line on the next page. This may be done by pressing the 'set tof' switch (in the upper right corner of the keyboard) after the paper has been positioned properly.

In addition to optimizing print head motion, 'sprint' provides an extended character set of Greek letters and mathematical symbols to support the special character functions of the Software Tools Subsystem text formatter, 'fmt'. These special graphics are accessed by normal ASCII character codes with their most significant bit turned *off*. (Note that the normal Prime convention is that this bit is always turned on for text characters.) The following table shows the correspondence between ASCII character codes, formatter functions, and special graphics:

Character	*'Fmt' Function*	*Graphic*
a	alpha	lower-case Greek alpha
A	ALPHA	upper-case alpha
b	beta	lower-case Greek beta
B	BETA	upper-case beta
c	chi	lower-case chi
C	CHI	upper-case chi
d	delta	lower-case Greek delta
D	DELTA	upper-case Greek delta
e	epsilon	lower-case Greek epsilon
E	EPSILON	upper-case epsilon
n	eta	lower-case Greek eta
N	ETA	upper-case eta
g	gamma	lower-case Greek gamma
G	GAMMA	upper-case Greek gamma
8	infinity	"infinity" symbol
+	integral	integration symbol
9	INTEGRAL	large integration sign
i	iota	lower-case iota
I	IOTA	upper-case iota
k	kappa	lower-case kappa
K	KAPPA	upper-case kappa
l	lambda	lower-case Greek lambda
L	LAMBDA	upper-case Greek lambda
u	mu	lower-case Greek mu
U	MU	upper-case mu
^	nabla	inverted delta (APL del)
~	not	EBCDIC-style "not" symbol
v	nu	lower-case Greek nu
V	NU	upper-case nu
w	omega	lower-case Greek omega
W	OMEGA	upper-case Greek omega
o	omicron	lower-case omicron
O	OMICRON	upper-case omicron
-	partial	partial differential symbol
p	phi	lower-case Greek phi
P	PHI	upper-case Greek phi
y	psi	lower-case Greek psi
Y	PSI	upper-case Greek psi
3	pi	lower-case Greek pi
4	PI	upper-case Greek pi
r	rho	lower-case Greek rho
R	RHO	upper-case rho
s	sigma	lower-case Greek sigma
S	SIGMA	upper-case Greek sigma
t	tau	lower-case Greek tau
T	TAU	upper-case tau
h	theta	lower-case Greek theta
H	THETA	upper-case Greek theta
q	upsilon	lower-case upsilon
Q	UPSILON	upper-case upsilon
x	xi	lower-case Greek xi
X	XI	upper-case xi
z	zeta	lower-case Greek zeta
Z	ZETA	upper-case zeta

7	downarrow	arrow pointing down
6	uparrow	arrow pointing up
5	backslash	"back slash" symbol
2	tilde	"tilde" symbol
0	largerbrace	large square right brace
1	largelbrace	large square left brace
=	proportional	"proportional" symbol
M	apeq	approximately equal to
]	ge	greater than or equal to
@	imp	implies
[	exist	there exists
_	AND	logical and
\	ne	not equal to
<	psset	proper subset
>	sset	subset
?	le	less than or equal to
}	nexist	there does not exist
	univ	for every
{	OR	logical or
|	iso	congruence
f	lfloor	left floor
j	rfloor	right floor
m	lceil	left ceiling
F	rceil	right ceiling
	small0	a small 0
	small1	a small 1
	small2	a small 2
	small3	a small 3
	small4	a small 4
	small5	a small 5
	small6	a small 6
	small7	a small 7
	small8	a small 8
	small9	a small 9
	scolon	semicolon
	dquote	double quote
	dollar	dollar sign

These extended graphics are produced by fractional motions of the platen and print head and overstriking of standard ASCII graphics. Best results are obtained when the paper is being fed by the platen and pinch roller and not by the pin-feed mechanism.

Most of these characters *require* the special Times-Roman/Mathematics type wheel.

Examples

```
help -p sprint | sprint
sprint junk
sprint -s -l 80 journal_article
sprint -c 5 -j hand_out
```

Messages

"Usage: sprint ..." for incorrect argument syntax.
"<filename>: can't open" if given file could not be opened for reading.

Bugs

If interrupted by the BREAK key while printing, 'sprint' may hang, waiting for the Spinwriter to acknowledge the last group of characters sent. To clear this condition, it is simply necessary to type a ctrl-f at the keyboard. If the ctrl-p key is used instead of BREAK, this condition normally does not occur.

When multiple copies of a file are requested using the "-c" option, 'sprint' obliges by rewinding the input file and re-reading it. If the input is being taken from a standard input port, and that port is not connected to a rewindable device (i.e., a disk file), then only one copy is produced.

Error messages are produced on the standard error output port, which is normally directed to the terminal. If it is undesirable to have these messages interspersed with the contents of the printed files, error output should be redirected to a file.

See Also

cat (1), copy (1), dprint (3), print (1), fmt (1)

Usage

symbols (-6800 | -8080) <object_file>

Description

'Symbols' prints the symbol table placed in an object code file by one of the cross-assemblers 'as6800' or 'as8080' or by the linker 'lk'. The mandatory first argument specifies the assembler used to create the original object code file.

Each symbol is printed along with its (16-bit) value and a "type" designator, which is "ext" for external, "rel" for relocatable, or "abs" for absolute (8080 register mnemonics).

Examples

```
symbols -6800 .o
symbols -8080 mux
```

Messages

"Usage: symbols ..." for invalid argument syntax.
Warnings if the object file could not be opened or if it was improperly structured.

See Also

as6800 (3), as8080 (3), lk (3), size (3)

Usage

terminate [<system>]

Description

The 'terminate' command is an interface to the SWT 'ring' process which allows validated users to terminate the currently executing ring process on a system. The specified 'ring' process clears all of its connections and terminates. However, since a 'ring' process must always be running on each system in the ring to ensure the security of the ring, the shell file executing 'ring' will immediately re-execute it. 'Terminate' allows the 'ring' comoutput file to be reinitialized, and can be used to execute a new version of the 'ring' process. If <system> is specified, 'ring' terminates on the machine with that system name, otherwise all 'ring' processes terminate.

Examples

terminate gt.a

terminate

Messages

Cannot transmit TERMINATE request
: Something interfered with the transmission of the TERMINATE command to the 'ring' process. This should never happen.

Networks are not configured
: The system is not configured to support PRIMENET.

Request to <system> failed
: The attempt to broadcast the message on system <system> failed.

Request to <system> succeeded
: The attempt to broadcast the message on system <system> succeeded.

Requested system is not in the ring
: The system which was to be terminated is not in the ring.

Ring connection has been terminated
: The connection to the 'ring' process has been cleared.

Termination complete
: The TERMINATE command has been successfully attempted on all requested systems.

TERMINATE request initiated
The TERMINATE command has been transmitted to the 'ring' process.

Unable to connect to ring node
The current system is not running a 'ring' process.

You are not validated to TERMINATE
Your user number is not allowed to use the TERMINATE command.

Bugs

Will not work if the current system is not running 'ring'.

See Also

broadcast (3), execute (3), setime (3)

Usage

translang [-b | -l | -bl] <input_file> [-h <hex_file>]

Description

'Translang' is a translator for the Burroughs D-Machine symbolic microprogramming language, described in *Microprogramming Primer*, by Harry Katzan, Jr. (McGraw-Hill Book Company 1977).

The source code to be translated is read from the file <input_file>, which conventionally is named with a ".d" suffix (e.g. "multiply.d"). The hexadecimal microprogram is written to <hex_file> if it is specified; otherwise, it is written to a file whose name is the input file with the suffix changed to ".h" (e.g. "multiply.h"). (If the input file has no suffix, ".h" is simply appended.) This file name should be given to 'dmach' for simulation.

The options control output listing behavior. If "-b" is specified, the binary micro and nano instructions are listed after each line of source text. If "-l" is specified, the hexadecimal micro and nano instructions are listed after the entire source text. If neither option is specified, no listing is produced.

The listing, if generated, is produced on standard output one, and may be redirected to a file or to another program by the usual Subsystem I/O redirection operators. Each line of the source file is listed (double spaced), followed by any error messages that pertain to its translation and by instruction bit patterns (if the "-b" option is used). When no listing is generated, error messages will appear on standard output, preceded by the number of the line causing the error.

The language accepted by 'translang' is a superset of the language defined in Katzan. The following differences are particularly worthy of note:

- The full 96-character ASCII character set may be used. Upper case is not distinguished from lower case.
- Input is totally free-form; spaces are necessary only to separate adjacent keywords or labels.
- The character sequence "->" may be used in addition to "=". Spaces around these assignment operators are not significant.
- The character "%" (from the reference language) may be used in place of "$" to precede a comment.
- There is no need to terminate each source line with

"$".

- The key words "comment" and "commnt" may both be used to precede comments. Furthermore, they may appear anywhere on a line (not just at the beginning).

- Statement labels are not limited to 6 characters in length. (In practice, however, no statement label may be longer than a single input line.)

- The problems with the microcode listing mentioned on page 135 of Katzan have been corrected. The bit patterns listed are now always complete.

- Empty statements are now allowable, and are recommended for improving the readability of microprograms. Specifically, blank lines may be used at will, and labels may be placed on lines by themselves to facilitate insertion and deletion of code following them.

- The character ":" may be used in addition to "." to terminate a statement label.

- Commas are totally ignored; they may be used wherever desired.

- The "end" statement served no purpose and is no longer required (although it will be accepted as a comment without complaint).

There are two major results of these changes: (1) the reference language used throughout Katzan may be translated without change, which was not previously the case; (2) the minor inflexibilities and inconsistencies present in the original translator have been eliminated, thus making its use a little less complex and frustrating.

Examples

translang -b multiply.d
: The source program will be read from the file "multiply.d"; the hexadecimal output will be written to the file "multiply.h". A listing of the source code and the bit patterns produced for each instruction will be sent to the user's terminal.

translang -l emulator -h hex >listing
: The source program will be read from the file "emulator" and the hexadecimal output will be written to the file "hex". A listing of the source code and the hexadecimal microprogram will be placed on the file "listing".

translang -lb stack.d >/dev/lps

The source program will be read from the file "stack.d"; the hexadecimal output will be written to the file "stack.h"; a listing of the source code, the bit patterns it produces, and the hexadecimal microprogram will be printed on the line printer.

Messages

Several syntax and semantics error messages may be produced. These are intended to be self-explanatory.

Bugs

This particular implementation has not been thoroughly tested, so if mystifying results occur, the bit patterns generated by suspect instructions should be reported to someone in the Software Support group.

Since so much of the nano-instruction syntax is optional, it is difficult to detect syntax errors and produce meaningful diagnostics.

See Also

dmach (3), *Microprogramming Primer*

Usage

ts [in | out] [<hh>:<mm> [<mm>/<dd>]]

Description

'Ts' was written to ease the monthly chore of preparing a time sheet. During the month, the worker uses 'ts' like a time clock, entering "ts in" as he begins a work session and "ts out" as he concludes it. His entry and exit times are recorded to the nearest quarter-hour. (Should variations in time be necessary, he may specify a time and, optionally, a date on the command line.) His comings and goings are recorded in a file named ".ts" in his variables directory.

At the end of the month, the worker simply enters the command "ts", which causes a reasonably readable time sheet to be printed on standard output. This timesheet contains daily, weekly, and monthly totals.

After his time has been reported to his superior, the worker should delete his old ".ts" file and begin anew.

Examples

```
ts in
ts out 12:45
ts
```

Files

=varsdir=/.ts for record of work

Messages

"Usage: ts ..." for invalid argument syntax.
"can't open time sheet file" when unable to open "=varsdir=/.ts".

Bugs

This program is incredibly locked in to the pay period used in the Georgia Tech School of Information and Computer Science; e.g., pay periods must begin on the 18th of the month and end on the 17th of the next, and all entries in the timesheet file must have dates between those limits. 'Ts' is also guaranteed to fail on "pathological" timesheet files: those that have entries missing or out of order.

Locally supported.

See Also

log (1)

Usage

unoct [<filename>]

Description

'Unoct' will read the ASCII output of the UNIX program 'od' (octal dump) present on the named file, convert it to binary, and write the result in sixbit code suitable for loading on the GT40 graphics terminal. (If the filename is omitted, standard input is assumed.)

At present, 'unoct' is necessary for loading such programs as FOCAL-GT.

Examples

unoct focal

Messages

"<filename>: can't open" for obvious problems.

Bugs

'Unoct' is kind of an ad hoc solution to the object code porting problem that will hopefully become unnecessary in the near future. It is also somewhat peculiar to the environs of Georgia Tech.

Locally supported.

See Also

scroll (3), information in GT40 directory (//gt40).

Usage

```
wallclock [ <delay> [ <fill_char> ]]
```

Description

'Wallclock' is a program which uses the CRT as a rather large (and expensive) digital display timepiece. It prints out the time that the "clock" was started, in small characters, and then every <delay> seconds (default is one), updates the current time, in large characters.

The characters will be made up of "*"s, unless the user cares to specify an alternate character in <fill_char>.

To stop the clock, use the BREAK key, or type a control-p.

Examples

```
wallclock
wallclock 5
wallclock 10 $
```

See Also

clock (1), vt?* routines (2)

Usage

```
who {-a|-l|-p|-q}
```

Description

'Who' prints a listing on standard output that shows which users are currently logged in. Information provided on each logged-in user includes his login name, his process number, the time at which he logged in, his full name, and either his location or his current login project. If the length of a login name exceeds 8 characters then 'who' prints the name on a line by itself and the other information on the next line.

Available options are:

-a Display information on all active processes, including phantoms; by default, 'who' provides information only on real users.

-l Display the locations of the logged in users. This is the default behavior. This option cannot be specified if the "-p" option is used.

-p Display the current projects of the logged in users. This option cannot be specified if the "-l" option is used.

-q Do not print the header lines.

Examples

```
who
who -a
who -p
```

Files

=userlist= to correlate a login name with the user's full name
=termlist= to correlate process numbers with terminal locations

Messages

"Usage: who ..." for invalid argument syntax.
"can't read user list" when unable to read "=userlist=".
"can't read terminal list" when unable to read "=termlist=".
"can't display both location and project at the same time" when both "-l" and "-p" options are specified.

Bugs

The date of login is not displayed; thus, the time displayed
* for phantom users is probably useless.

See Also

us (1)

Section 4 - Locally-Supported Library Subprograms

A number of library subprograms with highly specialized uses in mathematics and programming are included in a number of * locally-supported Subsystem libraries.

This section is designed to give the user a working knowledge of these functions and subroutines. Each routine has its own entry organized under the following headings. Note that a heading will be omitted if it contains no information.

Header Line

The subprogram's name, a synopsis of its purpose, and the date of last modification to its documentation.

Calling Information

The subprogram declaration and the declarations of its arguments, as well as the name of the library in which it can be found. This should be used as a reference when constructing calls to a given routine.

Function

A description of the purpose of the routine, along with the interpretations of its arguments and the returned value (if any).

Implementation

A short discussion of the strategy used to implement the routine, abstracted from the source code.

Arguments Modified

Names of those arguments modified by the routine.

Calls

Other subprograms called by this routine.

Bugs

Known problems with the use of the routine.

See Also

References to further information or related routines.

Calling Information

```
logical function abq$xs (qu, addition)
shortcall abq$xs (4)

queue_control_block qu
untyped addition

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine adds a 16 bit quantity (the contents of 'addition') to the bottom of a circular queue (deque) structure at 'qu'. The function result is TRUE if the addition was done, FALSE if the queue was full (before the call).

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The hardware ABQ instruction is executed on the arguments.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

qu

Bugs

The routine makes no attempt to validate the argument passed as a queue control block.

Locally supported.

See Also

atq$xs (4), fc (1), mkq$xs (4), rtq$xs (4), rbq$xs (4), tsq$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
logical function atq$xs (qu, addition)
shortcall atq$xs (4)

queue_control_block qu
untyped addition

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine adds a 16 bit quantity (the contents of 'addition') to the top of a circular queue (deque) structure at 'qu'. The function result is TRUE if the addition was done, FALSE if the queue was full (before the call).

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The hardware ATQ instruction is executed on the arguments.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

qu

Bugs

The routine makes no attempt to validate the argument passed as a queue control block.

Locally supported.

See Also

abq$xs (4), fc (1), mkq$xs (4), rtq$xs (4), rbq$xs (4), tsq$xs (4), *System Architecture Reference Guide*, (Prime PDR 3060)

Calling Information

```
long_int function gcd (x0, x1)
long_int x0, x1
```

Library: vswtmath (Subsystem mathematical library)

Function

'Gcd' determines the greatest common divisor of the two long integers specified as arguments. The function return is the GCD (always positive).

Implementation

'Gcd' is a straightforward implementation of Euclid's algorithm.

Bugs

Behavior with nonpositive arguments may be considered irrational by some.

See Also

invmod (4)

Calling Information

```
character function get$xs (array, position)
shortcall get$xs (4)

untyped array (ARB)
integer position

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine extracts a byte quantity from the specified array using highly efficient indexing and byte swapping operations.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Bugs

Does no bounds checking on the array (standard FTN problem), but this may also be seen as a good point.

Locally supported.

See Also

fc (1), put$xs (4)

Calling Information

```
subroutine gky$xs (word)
shortcall gky$xs (2)

integer word

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This subroutine loads the current cpu keys into 'word'.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The subroutine uses the TKA instruction.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

word

Bugs

Locally supported.

See Also

fc (1), sky$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
long_int function invmod (x1, x0)
long_int x1, x0
```

Library: vswtmath (Subsystem mathematical library)

Function

'Invmod' is used to find the inverse of 'x1' in the ring of integers modulo 'x0'. The function return is the inverse if it could be found, or ERR if 'x1' and 'x0' are not relatively prime.

Implementation

'Invmod' uses a variant of Euclid's greatest common divisor algorithm.

Bugs

Rational behavior for nonpositive arguments has not been established.

Locally supported.

See Also

gcd (4)

Calling Information

```
pointer function lsallo (ptr, len)
pointer ptr
integer len
```

Library: vlslb

Function

A string of length 'len' (not counting the EOS) is allocated. The pointer to the string is returned in 'ptr' and as the function value. If all attempts to find sufficient space fail, an error diagnostic ("Too many linked strings") is issued and the program is aborted.

Implementation

First, a test is made to see if there are 'len' characters available between the highest used location and the top of the string space to allocate the string. If not, the available space list is followed to find space. If both fail, storage is reclaimed by calling 'lsfree' to deallocate the available space list, decrementing the highest open pointer to the first allocated location, and rebuilding the available space list. If a second search then fails, 'error' is called to print the diagnostic and abort the program.

Arguments Modified

ptr

Calls

error, lsdump, lsfree, remark

Bugs

There is no way for the user to intercept a 'string space full' condition.

If not enough space is available in either the available space list or highest open list, but enough is available in both, an error is still signalled.

Locally supported.

See Also

lsfree (4)

Calling Information

```
character function lscmpk (ptr, str)
pointer ptr
character str (ARB)
```

Library: vlslb

Function

The linked string specified by 'ptr' and the contiguous string in 'str' are compared on the basis of ASCII collating sequence. Depending upon the relation that the first string has to the second, a function value of '>'c, '='c, or '<'c is returned.

Implementation

Characters are extracted from the linked string using 'lsgetc' and compared with their corresponding elements in 'str' until two unequal characters are seen or an EOS character is encountered. The value returned is then decided from these two characters: if one of the characters is EOS, the longer string is considered greater; if both of the characters are EOS, the strings are considered equal; if neither character is EOS, the string with the largest character is considered greater.

Calls

lsgetc

Bugs

Locally supported.

See Also

lscomp (4)

Calling Information

```
character function lscomp (ptr1, ptr2)
pointer ptr1, ptr2
```

Library: vlslb

Function

The linked strings specified by 'ptr1' and 'ptr2' are compared on the basis of ASCII collating sequence. The value of the function is '>'c, '='c, or '<'c, depending upon the relation that the first string has to the second.

Implementation

Characters are extracted from the strings using 'lsgetc' until two unequal characters are found or an EOS character is seen. The returned value is then decided from these two characters: if one of the characters is EOS, the longer string is considered greater; if both of the characters are EOS, the strings are considered equal; if neither character is EOS, the string with the larger character is considered greater.

Calls

lsgetc

Bugs

Locally supported.

See Also

lscmpk (4)

Calling Information

```
subroutine lscopy (ptr1, pos1, ptr2, pos2)
pointer ptr1, ptr2
integer pos1, pos2
```

Library: vlslb

Function

The string specified by 'ptr1', beginning at position 'pos1' is copied to the string specified by 'ptr2' beginning at position 'pos2'. If 'ptr2' is zero, a string of the proper length is allocated and the pointer to it is returned in 'ptr2' after copying. If in copying, the resultant string would overflow the space allocated for the second string, no new space is allocated, and the copy terminates.

Implementation

The first string is positioned to position 'pos1' with a call to 'lspos'. Then, if 'ptr2' is zero, a string of the proper length is allocated with a call to 'lsallo'. The second string is then positioned to position 'pos2' and characters are copied until the end of one string is reached by using 'lsgetc' and 'lsputc'.

Arguments Modified

ptr2 (if zero)

Calls

lsallo, lsgetc, lslen, lspos, lsputc

Bugs

Locally supported.

Calling Information

```
pointer function lscut (ptr1, pos, ptr2)
pointer ptr1, ptr2
integer pos
```

Library: vlslb

Function

The string specified by 'ptr1' is divided following position 'pos'. The first half of the string is returned in 'ptr1', and the second half is returned in 'ptr2' and as the value of the function.

Implementation

The string specified by 'ptr1' is positioned with 'lspos' to position 'pos'. A new string of length 1 is allocated, and the character at position 'pos' is placed in the new string. A pointer is placed in position 'pos' to the new string. 'Ptr2' and the function are given the value of the string position after position 'pos'.

Arguments Modified

ptr1, ptr2

Calls

lsallo, lsgetc, lspos, lsputc

Bugs

Locally supported.

See Also

lsjoin (4), lssubs (4)

Calling Information

```
subroutine lsdel (ptr, pos, len)
pointer ptr
integer pos, len
```

Library: vlslb

Function

Characters are deleted from the string specified by 'ptr' starting from position 'pos' and continuing for 'len' characters. 'Len' may be specified as a huge number to delete all remaining characters in the string. Even if all characters in the string are deleted, the pointer that remains in 'ptr' is still valid and points to a string containing EOS.

Implementation

The string is positioned to position 'pos' with 'lspos'. 'Lsfree' is called to free 'len' characters. If 'lsfree' returns 0 as a pointer value (meaning it ran past the EOS), EOS is placed is position 'pos'; otherwise, the pointer returned by 'lsfree' is placed in position 'pos'.

Calls

lsfree, lspos

Bugs

Locally supported.

See Also

lsdrop (4), lssubs (4), lstake (4)

Calling Information

```
pointer function lsdrop (ptr, len)
pointer ptr
integer len
```

Library: vlslb

Function

The value of the function is a pointer to a string containing all but the first 'len' characters of the string specified by 'ptr'.

Implementation

'Lspos' is called to position the string to position 'len' + 1. 'Lscopy' is then called to copy the remainder into a newly allocated string, a pointer to which is returned as the function value.

Calls

lscopy

Bugs

Locally supported.

See Also

lsdel (4), lssubs (4), lstake (4)

Calling Information

subroutine lsdump

Library: vlslb

Function

The linked string space is dumped in semi-readable format to ERROUT.

Implementation

The string space is printed with various calls to 'print' and 'putch'. Long sequences of 'empty' space are compressed. Unprintable characters are printed as octal values enclosed in angle brackets.

Calls

print, putch

Bugs

Locally supported.

See Also

dump (1)

Calling Information

```
integer function lsextr (ptr, str, max)
pointer ptr
character str (ARB)
integer max
```

Library: vlslb

Function

The linked string specified by 'ptr' is copied into 'str'. No more than 'max' positions of 'str' will be used.

Implementation

Characters from the linked string are extracted using 'lsgetc' and placed in consecutive positions of 'str'.

Arguments Modified

str

Calls

lsgetc

Bugs

Locally supported.

See Also

lsmake (4)

Calling Information

```
subroutine lsfree (ptr, len)
pointer ptr
integer len
```

Library: vlslb

Function

The first 'len' characters of the string specified by 'ptr' are deallocated. 'Ptr' is updated to point to the remaining characters. If no characters remain ('len' is longer than the string) 'ptr' is set to zero.

Implementation

The string is traversed, setting all visited locations to the value UNUSED, until 'len' characters or an EOS has been passed.

Arguments Modified

ptr

Bugs

Space is not available for reuse until after garbage collection. This is done to avoid pointer fragmentation.

'Lsfree' is used for returning strings to the free list. It is not careful with pointers, so it should usually be called only to completely deallocate a string (i.e. "call lsfree (ptr, ALL)").

Locally supported.

See Also

lsallo (4)

Calling Information

```
character function lsgetc (ptr, c)
pointer ptr
character c
```

Library: vlslb

Function

The first character in the string specified by 'ptr' is extracted and returned in 'c' and as the function value. 'Ptr' is updated to point to the next character in the string, but is never advanced beyond the EOS.

Implementation

Any pointers in the string are followed until a character is found. The character becomes the value of the function. If the character was not EOS, 'ptr' is incremented, and any pointers in the string are followed until the next character is found.

Arguments Modified

ptr, c

Bugs

Locally supported.

See Also

lsputc (4)

Calling Information

```
integer function lsgetf (ptr, fd)
pointer ptr
file_des fd

Library:  vlslb
```

Function

'Lsgetf' reads characters from the file specified by 'fd' into a linked string until a NEWLINE character is read. A pointer to the string is returned in 'ptr'. The function value is the number of characters read, or EOF if end-of-file was encountered before a NEWLINE was seen.

Implementation

A new string of zero length is allocated with a call to 'lsallo' and 'ptr' is set to point to it. Subroutine 'getlin' is then called repeatedly until a line whose last character (before the EOS) is a NEWLINE is returned, or end-of-file is encountered. Each line returned is then joined to the end of the linked string with a call to 'lsjoin'. If EOF is encountered before a NEWLINE is seen, the entire string is deallocated with a call to 'lsfree'.

Arguments Modified

ptr

Calls

getlin, lsallo, lsjoin, lsmake, lspos

Bugs

Locally supported.

See Also

lsputf (4)

Calling Information

subroutine lsinit

Library: vlslb

Function

'Lsinit' initializes the string space and associated variables. It *must* be called before using any other linked string routines.

The routines in the linked string library are intended to overcome several disadvantages of the contiguously stored character strings used throughout the Software Tools Subsystem. They facilitate operations such as insertion, deletion and concatenation with a minimum of wasted storage and time. These routines also free the programmer from having to explicitly manage the string storage. However, use of the linked string routines is costly in that for operations such as copying or replacing single characters, they are slower and require more subprogram calls than their equivalent contiguous string routines. Therefore, linked strings are not intended to replace contiguously stored strings, but to provide an extension that facilitates complex string manipulation.

All linked strings are allocated in the named common block 'ls$buf'. Normally, the user does not directly reference this block; rather, references are made through pointers returned from and passed to the linked string routines. Pointers are single-precision integer variables that contain an index of the starting location of the string in the common block. The user has no need to examine the pointers other than to pass them as arguments to the linked string routines.

Linked strings are stored one ASCII character per word, right-justified with zero fill and terminated by an EOS character. Any word having a value greater than 300 (decimal) is interpreted to be a pointer whose value is obtained by subtracting 300 from the word. This allows for the non-contiguity of characters in the string, hence the name "linked".

Space for new strings is obtained either directly or indirectly through the 'lsallo' function. 'Lsallo' attempts to allocate the string contiguously at the end of the common block. If this fails, the available space list is examined; if no space is found here, the garbage collector is invoked and the searches are repeated. Upon a second failure to find sufficient space, an error diagnostic is issued and the program terminated.

Old strings are deallocated using the 'lsfree' subroutine.

Deallocated strings are marked with a special value and are not available for use until after garbage collection.

Implementation

The pointers in the common block 'ls$buf' are set to their proper values. A call to 'lsinit' has the effect of deallocating all strings.

Bugs

'Lsinit' must be called to initialize the string space.

No provision is made for specifying the size of the string space.

Locally supported.

Calling Information

```
subroutine lsins (ptr1, pos1, ptr2, pos2, len)
pointer ptr1, ptr2
integer pos1, pos2, len
```

Library: vlslb

Function

The substring specified by 'ptr2' (from position 'pos2' with length 'len') is inserted into the string specified by 'ptr1' after position 'pos1'. String 2 is *not* destroyed. A pointer to the resulting string is returned in 'ptr1'.

Implementation

If 'pos1' is less than or equal to zero, the string specified by 'ptr2' (string 2) is prepended to the string specified by 'ptr1' (string 1). This is accomplished by copying string 2 into a new string (string 3), pointing 'ptr1' to string 3, and replacing the EOS of string 3 with a pointer to string 1.

If 'pos1' is greater than zero, string 2 is inserted within string 1 (if 'pos1' is greater than the length of string 1, it is assumed to be equal to the length of string 1). String 2 is copied to a new string (string 3) with an extra position at the beginning. String 1 is positioned to 'pos1'. The character at this position is placed at the beginning of string 3, a pointer to string 3 replaces this character, and the EOS of string 3 is replaced with a pointer to 'pos1' + 1 of string 1.

Arguments Modified

ptr1

Calls

lscopy, lsdel, lslen, lspos, lssubs

Bugs

In appending string 2 to string 1, it is slightly less efficient to specify a large number for 'pos1' than to specify the exact length of string 1.

Locally supported.

See Also

lssubs (4)

Calling Information

```
pointer function lsjoin (ptr1, ptr2)
pointer ptr1, ptr2

Library:  vlslb
```

Function

The string specified by 'ptr2' is concatenated to the end of the string specified by 'ptr1'. A pointer to the resulting string is returned in 'ptr1' and as the function value. 'Ptr2' ceases to be a valid pointer.

Implementation

The string specified by 'ptr1' is positioned to its end. Then the EOS character in the first string is replaced by 'ptr2' + 300, thus linking the second string to the first.

Calls

lspos

Bugs

Locally supported.

See Also

lscut (4), lsins (4)

Calling Information

```
integer function lslen (ptr)
pointer ptr
```

Library: vlslb

Function

The length of the string specified by 'ptr' is returned as the function value. 'Ptr' is not modified.

Implementation

The number of characters in the string is counted by calling 'lsgetc' until it returns EOS. The length is computed as the number of calls to 'lsgetc' minus 1.

Calls

lsgetc

Bugs

Locally supported.

Calling Information

```
pointer function lsmake (ptr, str)
pointer ptr
character str (ARB)
```

Library: vlslb

Function

The contiguous string in 'str' is copied into the linked string space and a pointer to the string is returned both in 'ptr' and as the function value.

Implementation

A new string is allocated with the same length as 'str' via a call to 'lsallo'. Characters are then copied into the string using 'lsputc'.

Arguments Modified

ptr

Calls

length, lsallo, lsputc

Bugs

Locally supported.

See Also

lsextr (4)

Calling Information

```
character function lspos (ptr, pos)
pointer ptr
integer pos
```

Library: vlslb

Function

'Ptr' is updated to point to the string starting at position 'pos'. 'Ptr' will not be updated past the EOS. The value returned by the function is the character in position 'pos'.

Implementation

The string is traversed until 'pos' - 1 characters have been skipped. The new pointer is then returned in 'ptr' and as the function value.

Arguments Modified

ptr

Bugs

Locally supported.

Calling Information

```
character function lsputc (ptr, c)
pointer ptr
character c
```

Library: vlslb

Function

The character in 'c' is placed in the next position of the string specified by 'ptr'. 'Ptr' is then updated to point to the next available position. The function value is the value of 'c', unless there is no more room in the string. In this case, EOS is returned and the pointer is not updated. If an EOS is put in the string before the end, the remaining character positions are deallocated.

Implementation

Pointers in the string are followed until a character is found. If the character is not EOS, it is replaced by the value of 'c' and 'ptr' is incremented. If the value of 'c' is EOS, 'lsfree' is called to deallocate the rest of the string.

Arguments Modified

ptr, c

Calls

lsfree

Bugs

'Lsputc' should perhaps allocate more space if the receiving string overflows.

Locally supported.

See Also

lsgetc (4)

Calling Information

```
subroutine lsputf (ptr, fd)
pointer ptr
file_des fd
```

Library: vlslb

Function

The linked string specified by 'ptr' is written to the file described by 'fd'.

Implementation

A section of the string no more than MAXLINE characters in length is extracted using 'lsextr' and written to the file with 'putlin'. The section just extracted is skipped over with a call to 'lspos' and the process is repeated until the EOS is encountered.

Calls

lsextr, lspos, putlin

Bugs

Locally supported.

See Also

lsgetf (4)

Calling Information

```
pointer function lssubs (ptr, pos, len)
pointer ptr
integer pos, len
```

Library: vlslb

Function

The value of the function is a pointer to a string containing 'len' characters from the string specified by 'ptr', starting at position 'pos'.

Implementation

A new string of length 'len' is allocated, the string specified by 'ptr' is positioned to 'pos', and 'len' characters are then copied to the new string with calls to 'lsgetc' and 'lsputc'.

Calls

lsallo, lsgetc, lslen, lspos, lsputc

Bugs

Locally supported.

See Also

lscut (4), lsdel (4), lsdrop (4), lstake (4)

Calling Information

```
pointer function lstake (ptr, len)
pointer ptr
integer len
```

Library: vlslb

Function

The value of the function is a pointer to a string consisting of the first 'len' characters of the string specified by 'ptr'.

Implementation

A string of length 'len' is allocated, and the first 'len' characters of the string specified by 'ptr' are copied into it using 'lsgetc' and 'lsputc'.

Calls

lsallo, lsgetc, lsputc

Bugs

Locally supported.

See Also

lsdrop (4), lssubs (4)

Calling Information

```
integer function mkq$xs (ptr_to_free, room, qu)
shortcall mkq$xs (4)

pointer ptr_to_free
integer room
queue_control_block qu

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This function initializes a queue control block so it can be used by the other queue functions. Queues are a machine-defined data type on higher-level Prime machines, and operations on queues are guaranteed "atomic" (non-interruptable). However, queues require special definition.

Queues must be a fixed size in length, and that length must be 2 ** k words long, with 4 <= k <= 16. Furthermore, the queue must start on a 2 ** k word boundary.

To make things easier for the user, this function simply requires that the user pass a pointer to a free area in memory, and the length of that area ('ptr_to_free' and 'room', respectively). The function then determines the largest queue that can fit into that free area and still meet the queue-related requirements. The function updates the queue control block 'qu' to reflect this placement, and then returns the number of available words in the queue as the function value.

If no queue can be allocated in the space provided, the function returns a zero value. It should be noted that it is possible that the size of the queue created may be only half of the free area due to the address boundary restrictions. Non-zero function returns are always (2 ** k) - 1.

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a PMA routine entered via a JSXB (shortcall).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

qu

Bugs

The function uses 'qu' for some temporary values; 'qu' may be partially initialized even if no queue can be created.

Locally supported.

See Also

abq$xs (4), atq$xs (4), fc (1), rbq$xs (4), rtq$xs (4), tsq$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
subroutine pek$xs (ptr_to_word, contents)
shortcall pek$xs (4)

pointer ptr_to_word
untyped contents

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

The subroutine returns the contents of the word at the address 'ptr_to_word'. Effectively,

```
call pek$xs (loc(word), contents)
```

is equivalent to

```
contents = word
```

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

contents

Bugs

No validity check is done on the pointer.

Locally supported.

See Also

fc (1), pok$xs (4)

Calling Information

```
subroutine pok$xs (ptr_to_word, contents)
shortcall pok$xs (4)

pointer ptr_to_word
untyped contents

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

The subroutine changes the contents of the word at the address 'ptr_to_word'. Effectively,

```
call pok$xs (loc(word), contents)
```

is equivalent to

```
word = contents
```

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Bugs

No validity check is done on the pointer.

The user may do very peculiar things to his/her environment if the call is not used with care.

Locally supported.

See Also

fc (1), pek$xs (4), s1c$xs (4), s2c$xs (4)

Calling Information

```
long_int function prime (i)
long_int i
```

Library: vswtmath (Subsystem mathematical library)

Function

'Prime' is used to retrieve a specified prime number. The argument is the ordinal of the prime number desired. The function return is the specified prime. For example, if 'i' is 1, the function return is 2; if 'i' is 3, the function return is 5, etc.

'Prime' uses the table of prime numbers in the file "=aux=/primes". This file contains the prime numbers up to one million in long-integer binary format. If "=aux=/primes" is unreadable or if 'i' is less than one or greater than 78498, the function return is zero.

Implementation

The file "=aux=/primes" is opened for reading. The read/write pointer for the file is then moved to the desired location and the prime number read. The file is then closed.

Calls

open, close, mapfd, Primos prwf$$

Bugs

Should probably raise cain if the prime numbers file is not available, rather than meekly returning zero.

Locally supported.

Calling Information

```
subroutine put$xs (array, position, char)
shortcall put$xs (4)

untyped array (ARB)
integer position
character char

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine inserts a byte quantity into 'array' at 'position', using highly efficient indexing and byte swapping operations. The byte is assumed to be the least significant byte of 'char'.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

array

Bugs

Does no bounds checking on the array (standard FTN problem), but this may also be seen as a good point.

Locally supported.

See Also

fc (1), get$xs (4)

Calling Information

```
long_int function pwrmod (p, e, n)
long_int p, e, n
```

Library: vswtmath (Subsystem mathematical library)

Function

'Pwrmod' is used to perform an integer exponentiation in the ring of integers modulo a given modulus. The argument 'p' is the base of the expression, 'e' is the exponent, and 'n' the modulus. The function return is p**E (mod n).

Implementation

'Pwrmod' examines the exponent a bit a time, squaring the intermediate result accumulated so far and multiplying it by the base whenever the selected bit is a 1. Each operation is performed modulo 'n', so that intermediate results don't become excessively large.

See Also

invmod (4)

Calling Information

```
logical function rbq$xs (qu, item)
shortcall rbq$xs (4)

queue_control_block qu
untyped item

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine removes a 16 bit quantity (into the variable 'item') from the bottom of a circular queue (deque) structure at 'qu'. The function result is TRUE if the removal was done, FALSE if the queue was empty (before the call).

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The function executes the RBQ machine instruction.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

qu, item

Bugs

The routine makes no attempt to validate the argument passed as a queue control block.

Locally supported.

See Also

abq$xs (4), atq$xs (4), fc (1), mkq$xs (4), rtq$xs (4), tsq$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
logical function rdy$xs (char)
shortcall rdy$xs (4)

character char
```

Library: shortlb
Also declared in =incl=/shortlb.r.i

Function

The function checks to see if a character has been typed at the terminal but not yet input by software. If no character is waiting, the function returns the value FALSE. If a character is waiting, then the function returns TRUE and 'char' gets set to the waiting character.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The function switches to 64R mode to do a "SKS '704" (handled by the Primos restricted instruction FIM). If a value is waiting, it is fetched by a call to the Primos routine T1IN.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Calls

Primos t1in

Arguments Modified

char

Bugs

Locally supported.

See Also

chkinp (2), fc (1)

Calling Information

```
logical function rtq$xs (qu, item)
shortcall rtq$xs (4)

queue_control_block qu
untyped item

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine removes a 16 bit quantity (into the variable 'item') from the top of a circular queue (deque) structure at 'qu'. The function result is TRUE if the removal was done, FALSE if the queue was empty (before the call).

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The function executes the RTQ machine instruction.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

qu, item

Bugs

The routine makes no attempt to validate the argument passed as a queue control block.

Locally supported.

See Also

abq$xs (4), atq$xs (4), fc (1), mkq$xs (4), rbq$xs (4), tsq$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
logical function s1c$xs (ptr_to_variable, old_value, new_value)
shortcall s1c$xs (4)

pointer ptr_to_variable
untyped old_value, new_value

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

The function implements an uninterruptable form of test-and-set operation. The parameter 'ptr_to_variable' is a 2 word virtual memory pointer to a 1 word location in memory to be tested and possibly modified.

If the variable contains the same value as provided in 'old_value' then the variable is updated to 'new_value' and the function returns TRUE. If the variable is not equal to 'old_value' then the function returns FALSE and no change is made to the variable. Effectively,

```
if (variable == old_value) {
   variable = new_value
   return (TRUE)
   }

else
   return (FALSE)
```

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The function uses the STAC instruction which is guaranteed to be atomic (non-interruptable).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Bugs

The pointer supplied is not checked for validity.

Locally supported.

See Also

fc (1), s2c$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
logical function s2c$xs (ptr_to_variable, old_value, new_value)
shortcall s2c$xs (4)

pointer ptr_to_variable
untyped old_value, new_value

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

The function implements an uninterruptable form of test-and-set operation. The parameter 'ptr_to_variable' is a 2 word virtual memory pointer to a 2 word location in memory to be tested and possibly modified.

If the variable contains the same value as provided in 'old_value' then the variable is updated to 'new_value' and the function returns TRUE. If the variable is not equal to 'old_value' then the function returns FALSE and no change is made to the variable. Effectively,

```
if (variable == old_value) {
   variable = new_value
   return (TRUE)
   }

else
   return (FALSE)
```

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The function uses the STLC instruction, which is guaranteed to be atomic (non-interruptable).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Bugs

The pointer supplied is not checked for validity.

Locally supported.

See Also

fc (1), s1c$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
subroutine set_copy (source, destination)
pointer source, destination
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_copy' duplicates one set in another. For proper operation, the source set should be larger than or equivalent in size to the destination set. The source set is not altered by the copy operation.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

'Set_copy' uses the size field encoded in the first word of each set to determine the number of words in the bit vector to be copied. A simple loop implements the copy.

Bugs

Should handle sets of different sizes properly.

See Also

other set operations ('set_?*') (4)

Calling Information

```
pointer function set_create (set, size)
pointer set
integer size
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_create' is used to create a Pascal-style bit vector representation for a set of integers from 1 to 'size'. The function return and the variable 'set' are set to the address in dynamic storage of the newly-created set.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

'Set_create' calls 'dsget' to obtain a contiguous array of 16-bit words that is large enough to represent a bit vector with 'size' elements. The first word of this array is set to 'size' for use by other set manipulation routines. A call to 'set_init' then insures that the new set is empty.

Arguments Modified

set

Calls

dsget, set_init

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_delete (element, set)
integer element
pointer set
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_delete' is used to remove a given element from a set. The first argument is the element (an integer between one and the maximum set size, inclusive), and the second is the set from which it is to be removed.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

The element selected is compared to the size field of the set; if invalid, 'set_delete' prints an error message and terminates the program. Otherwise, the position of the element in the bit vector is calculated, and the bit is reset by straightforward logical operations.

Calls

error

See Also

other set operations ('set_?*') (4)

Calling Information

```
integer function set_element (element, set)
integer element
pointer set
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_element' returns 1 if 'element' is a member of the set 'set', 0 otherwise. The argument 'element' must be an integer from 1 to the maximum size of the set, inclusive. The argument 'set' must have been created beforehand with 'set_create'.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

If 'element' is not in the range of allowable set elements for the given set, the program is terminated by a call to 'error'. Otherwise, the location of the element in the bit vector is calculated, and the function returns the value of the bit at that position.

Calls

error

See Also

other set routines ('set_?*') (4)

Calling Information

```
logical function set_equal (set1, set2)
pointer set1, set2
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_equal' determines if two sets contain the same members. The sets need not be of equal length.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

'Set_equal' makes two calls on 'set_subset'. The function return is true if 'set1' is a subset of 'set2' and 'set2' is a subset of 'set1', false otherwise.

Calls

set_subset

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_init (set)
pointer set
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_init' initializes a set created by 'set_create'. An initialized set is empty, i.e. contains no members.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

'Set_init' simply clears all elements of the bit vector portion of the data structure addressed by its first argument.

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_insert (element, set)
integer element
pointer set
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_insert' is the primary means of placing a given element in a set. 'Element' must be an integer between one and the maximum size of the set, inclusive; 'set' must be a pointer to a set data structure created by 'set_create'. If it is within range, the given element is marked "present" in the bit vector associated with the set.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

If the element is out of range, a call to 'error' is made to inform the user and terminate the program. Otherwise, the location of the element in the bit vector is determined and a few logical operations are employed to set the selected bit.

Calls

error

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_intersect (set1, set2, destination)
pointer set1, set2, destination
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_intersect' determines the intersection of the sets given as its first two arguments and places that intersection in the set specified by the third. For proper operation, all three sets should be equal in size.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

Does a word-by-word logical 'and' of the bit vectors for the first two sets, placing the result in the third.

Bugs

Should be fixed to work with sets of differing lengths.

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_remove (set)
pointer set
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_remove' reclaims the dynamic storage space used by a set data structure. It is the inverse of 'set_create'. To prevent dynamic storage space from becoming irretrievably lost, sets should always be removed by a call to 'set_remove' when they are no longer needed.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

Calls 'dsfree' to throw away the storage space used by the internal data structure.

Calls

dsfree

See Also

other set routines ('set_?*') (4), dsinit (2), dsget (2), dsfree (2)

Calling Information

```
logical function set_subset (set1, set2)
pointer set1, set2
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_subset' returns the logical value '.true.' if and only if its first argument points to a set that is a subset of or equal to the set pointed to by its second argument. The sets need not be of equal length.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

If one set is larger than the other, it is checked to make sure that none of the higher-order elements is present. The subset condition is then true if and only if every element of 'set1' is also an element of 'set2', a statement which can be checked a word at a time with the proper logical operations.

Calls

set_element

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_subtract (set1, set2, destination)
pointer set1, set2, destination
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_subtract' performs the set subtraction operation, i.e. places in the set 'destination' those elements of 'set1' that are not in 'set2'. For proper operation, all three sets should be the same size.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

Since sets are represented as bit vectors, the subtraction operation is performed by logically 'and'ing the elements of the first set with the negation of the elements of the second set.

Bugs

Should work with sets of differing sizes.

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine set_union (set1, set2, destination)
pointer set1, set2, destination
```

Library: vswtmath (Subsystem mathematical library)

Function

'Set_union' computes the union of 'set1' and 'set2', placing the result in 'destination'. For proper operation, all three sets should be the same size.

All set manipulation routines make use of dynamic storage, which must be initialized before use. See 'dsinit' for further information.

Note that all set manipulation routines have long names. To avoid unique name conflicts with other routines, any Ratfor program using the set routines should include the following statement:

```
include "=src=/lib/math/swtmlb_link.r.i"
```

Implementation

The set union is computed by logically 'or'ing the bit vectors associated with 'set1' and 'set2'.

Bugs

Should work with sets of differing sizes.

See Also

other set routines ('set_?*') (4)

Calling Information

```
subroutine sky$xs (word)
shortcall sky$xs (2)

integer word

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This routine loads bits 1 - 14 of the cpu keys with the corresponding bits of 'word'. This can change the processor addressing mode (for the current process), set or clear the carry and link bits and the condition codes, and change the system's response to integer, real and decimal exceptions.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The subroutine uses the TAK instruction.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Bugs

The user can possibly change the current program addressing mode in a manner that cannot be recovered by this routine.

Locally supported.

See Also

fc (1), gky$xs (4), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
subroutine stk$xs (root, ptr_to_ext)
shortcall stk$xs (4)

integer root
pointer ptr_to_ext

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

The Prime machines support the mechanism of a stack that can be extended into additional segments, as needed. This routine allows you to set the extension pointer for any stack, or read the current extension pointer for any stack. The function's actions depend on the value of 'root' (segment of stack root):

If (root == :100000) then the function returns the current extension pointer in 'ptr_to_ext'. (:100000 == ints(-32768))

If (root > -32768 & root < 0) then the function returns in 'ptr_to_ext' the current extension pointer for the stack whose root is in segment abs(root).

If (root == 0) then the function sets the extension pointer of the current stack to the value of 'ptr_to_ext'.

If (root > 0) then the function sets the extension pointer of the stack whose root is in segment 'root' to the value in 'ptr_to_ext'.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). If the operation is specified as relative to the current stack root then the stack segment number is taken from SB% + 1 (the current stack frame root pointer).

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

ptr_to_ext

Bugs

There is no validity checking done on either the 'root' parameter or the 'ptr_to_ext' parameter.

No validation is done to make sure that 'ptr_to_ext' points to a valid stack.

Locally supported.

See Also

fc (1), *System Architecture Reference Guide* (Prime PDR 3060)

Calling Information

```
logical function tsq$xs (qu, count)
shortcall tsq$xs (4)

queue_control_block qu
integer count

Library: shortlb
Also declared in =incl=/shortlb.r.i
```

Function

This function sets the variable 'count' to the number of entries in the queue at 'qu'. The function value is TRUE if the queue is non-empty, FALSE if the queue has no entries.

The declaration 'queue_control_block' is defined in =incl=/shortlb.r.i; this file should be included if this routine is used.

Implementation

Implemented as a simple PMA routine entered via a JSXB (shortcall). The hardware TSTQ instruction is executed on the arguments.

Note that any routine using this call must be compiled using the "-q" option of 'fc'.

Arguments Modified

count

Bugs

The routine makes no attempt to validate the argument passed as a queue control block.

Locally supported.

See Also

abq$xs (4), atq$xs (4), fc (1), mkq$xs (4), rtq$xs (4), rbq$xs (4), *System Architecture Reference Guide*, (Prime PDR 3060)

Section 5 - Low Level Support Commands

This section is devoted to the description of low level support Subsystem commands. These commands should not be invoked from command level, as they are intended to be used as support for higher level commands. Georgia Tech's low level support commands reside in the directory "=ebin=" and are supplied on the Software Tools release tape. The commands described in section 1 should be used to invoke the commands described here.

Documentation for each command is organized under the following headings. Note that a heading will be omitted if it contains no additional information.

Header Line

The command's name, function, and the date of last modification to the documentation.

Usage

A description of the syntax permitted on the command line. The notation used in this description is identical to that used in Section 1 of this manual.

Description

A detailed coverage of the capabilities and operation of the command.

Examples

A few short examples of the command.

Files

A list of the names of special files used by the command.

Messages

A listing of important error messages or diagnostic information issued by the command.

Bugs

Known bugs in the operation of the command.

See Also

References to further information or related commands.

Usage

bmerge {<object file>}

Description

'Bmerge' is a program which will take the object code files given as arguments, if any, and create a new object code file that is written to standard output. If you build your programs as separately compiled modules, with each module containing many subroutines/functions, you can use 'bmerge' to combine those modules into one object code file for building a library.

'Bmerge' accepts directives from standard input to indicate the order and type of subprograms to be included in the resulting object code file; by default, no subprograms will be included if there is no input.

The following may be included in the input stream to direct the creation of the object code file :

input item	*meaning*
<name>	include the named subprogram at the current point in the object code
.rfl	reset the "forced load" flag at this point
.sfl	set the "forced load" flag at this point

A sample input stream would be :

```
ave
.sfl
add
sub
mul
.rfl
div
```

If the files specified in the argument list contain more than one occurrence of an entry point name (i.e., possibly different versions of the same subprogram), then the version which gets included depends on the order in which the files were specified in the command invocation. Multiple occurrences of an entry point name in the input to 'bmerge' causes inclusion of more than one version of the named subprogram, with the inclusion order being the reverse of the order of occurrence (last-in, first-out basis).

Examples

```
entry_names> bmerge ave.b ave_lib.b >new_ave.b
files .b$ | change .b$ | bmerge [files .b$] >all_object.b
```

Messages

"<name>: too many object files" when trying to merge too many object code files at the same time.
"<name>: not found in object files" when trying to include a nonexistant routine.
"bad object file..." for an ill-formatted object code file.
"<name>: error copying object module" if the length of the routine in the resulting object code file is not of the same length as in the source file.
"block size (<size>) exceeds buffer space" if the next block to be read from the input object code file is larger than the program's file buffer.
"<name>: extraneous END block" for object code files which have too many END blocks.

various error messages from the dynamic storage routines

Bugs

Binary output is used to generate the resulting object code file. If standard output is the terminal, unpredictable results may be obtained because of the irrational behavior of binary I/O to the terminal.

Internal procedures (procedures/functions within procedures/functions) in PL/1, Pascal, or PL/P modules if specified by name, will not be merged correctly.

If a module has multiple entry points, only the first one is recognized.

See Also

bnames (5), brefs (5), ld (1), lorder (1)

Usage

bnames {<object file>}

Description

'Bnames' is a program which will open each of the files, if any, named as its arguments and print the names of the routines encountered on standard output. This program is useful for examining library files to see if the correct subroutines have been included.

The following are the possible types of output :

type	*meaning*
<name>	name of a subprogram entry point
.main	main program entry point
.data	Fortran block data module
.rfl	a reset "force load" loader group
.sfl	a set "force load" loader group

Examples

```
bnames ave.b ave_lib.b
bnames [files .b$] | find "%." -x >routine_names
```

Messages

"<name>: bad object file..." for an ill-formatted object code file.
"block size (<integer>) exceeds buffer space" for files whose block size exceeds the program input buffer size.
"<name>: extraneous END block" for superfluous END blocks in the object code file.

Bugs

If a module has multiple entry points, only the first one is recognized.

See Also

bmerge (5), brefs (5), ld (1), lorder (1)

Usage

```
brefs { <object file> | -n }
```

Description

'Brefs' prints the precedence relationships between the entry points that are defined and/or referenced within the named object files. Each output line contains two entry point names; the first is the name of the calling routine ($MAIN for Fortran main programs or unnamed assembly language routines), and the second is the name of an external object referenced by that routine. The output from 'brefs' is suitable for piping into 'tsort' to determine the proper ordering for routines in a library.

If the "-n" argument appears in the place of an object file name, 'brefs' will obtain names of object files from its standard input. For more information on this syntax, see the entry for 'cat' (1).

Examples

```
brefs ave.b ave_lib.b
brefs lib.b | tsort | bmerge lib.b >lib
```

Messages

"<object file>: can't open" if a non-existent or inaccessible file is specified.
"<object file>: bad object file" if something other than an object file is specified.
"block size exceeds buffer space" if the object file is badly formatted.

Bugs

If a module has multiple entry points, only the last entry point is recognized.

See Also

bmerge (5), bnames (5), ld (1), lorder (1), tsort (1)

Usage

bs

Description

'Bs' is a shell file that executes the program 'guess' when a command is not found in a user's search rule. 'Guess' is a program that tries to find a command "close" to one that was mistyped. This shell file may be added to the end of a user's search rule so that it can aid a fumble-fingered typist.

Examples

<<'bs' should not normally be run from command level>>

Bugs

Because of search rule problems, 'bs' will fail if a user does not have the current directory in his search rule.

Locally supported.

See Also

bs1 (5), guess (5), mkclist (3)

Usage

bs1

Description

'Bs1' is a shell file that executes the program 'guess' when a command is not found in a user's search rule. This program is identical to 'bs' except that it calls 'guess' with an argument of "1" for <maxcost>. This significantly reduces the search time, but restricts the set of commands that 'guess' will consider.

Examples

<<'bs1' should not normally be run from command level>>

Bugs

Because of search rule problems, 'bs1' will fail if a user does not have the current directory in his search rule.

Locally supported.

See Also

bs (5), guess (5), mkclist (3)

Usage

bugfm

Description

'Bugfm' is not meant to be directly user-invoked; rather, it is a utility used by the 'bug' command to solicit for bug report information such as the name of the reporter, the name of the command or subroutine which is suspected of having a bug, the name of an example file which generates an error with the named program, and a description of the error.

The time, date, and login name of the bug reporter are inserted in the resulting bug report to allow report verification. The resulting bug report is sent to standard output.

Examples

<< should not be invoked by the user >>

See Also

bug (3), raid (3)

Usage

bugn [-i]

Description

'Bugn' is not intended to be user-invoked; rather, it is a utility used by the 'bug' command to aid in bug report generation. It determines what the highest bug number is so far; if the optional argument "-i" is specified, it increments the bug number and replaces the old highest bug number with the new one. In either case, it prints the resulting bug number on standard output.

Examples

<< not to be invoked by the user >>

Files

=bug=/$ to store the current highest bug number; use of the "-i" option causes this file to modified.

Messages

"Usage: bugn ..." for improper calling sequence

Bugs

Will not handle more than 999 concurrent bug reports.

See Also

bug (3), raid (3)

Usage

```
c1 [-afuy] { -D<name>[=<value>] } { -I<dir> } { <file> }
```

Description

'C1' is a classical recursive-descent compiler for the C programming language, performing lexical analysis, preprocessing and parsing. 'C1' produces an "intermediate form" which can be used by the virtual code generator ('vcg') to produce 64V-mode relocatable object code, or PMA. 'C1' produces three files: "<file>.ct1" contains entry points, "<file>.ct2" contains external definitions, and "<file>.ct3" contains the intermediate form.

The following options are available:

-a Abort all active shell programs if any errors were encountered during processing. This option is useful in shell programs like 'ccl' that wish to inhibit compilation and loading if processing failed. By default, this option is not selected; that is, errors in processing do not terminate active shell programs.

-f Suppress automatic inclusion of the standard definitions file. Macro and common block definitions for the C Standard I/O Library and for interfacing with the Subsystem reside in the file "=cdefs=". 'C1' will process these definitions automatically, unless the "-f" option is specified.

-u Reserved.

-y Check for potential problems, e.g. type mismatches. If this option is selected, messages are output in "<file>.ck".

-D Defines the identifier <name> with optional <value> for program internal use (maximum of 10).

-I Specifies a directory where include files reside (maximum of 10). All "-I" directories are searched after the current directory and before "=incl=".

NOTE: This command is not meant to be directly invoked by the user. Use one of the compiler interludes, 'cc', 'ccl', 'ucc', or 'compile'.

Examples

```
c1 file.c
```

```
c1 prog.c -af
```

Messages

Numerous and self-explanatory.

Bugs

Several known compiler bugs exist. See the *User's Guide for the Georgia Tech C Compiler*.

See Also

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler. cc (1), ccl (1), compile (1), ucc (1), vcg (1), *User's Guide for the Georgia Tech C Compiler*

Usage

cck1

Description

'Cck1' is the first phase of the UNIX (tm) 'lint' like facility provided by the Georgia Tech Software Tools C compiler. This program is normally not called directly by the user, but instead via one of the C compiler interludes, with the '-y' option.

'Cck1' reads a '.ck' file on its first standard input, and produces output which should be sorted and passed on to the second phase, 'cck2'.

The '.ck' file is produced automatically by 'c1' when the '-y' option is passed on to it from one of the compiler interludes.

Examples

```
prog.ck> cck1 | sort | cck2
```

Files

?*.ck file output by 'c1' with the '-y' option, used as input to 'cck1'.

Messages

"Too many nesting levels"

"Nesting stack overflow"

Bugs

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

cc (1), ccl (1), ucc (1), c1 (5), cck2 (5), *User's Guide to the Georgia Tech C Compiler*

Usage

cck2

Description

'Cck2' is the second phase of the UNIX (tm) 'lint' like facility provided by the Georgia Tech Software Tools C compiler. This program is normally not called directly by the user, but instead via one of the C compiler interludes, with the '-y' option.

'Cck2' reads the (hopefully) sorted output of 'cck1', the first phase of the C program checker. It then prints messages detailing possible syntactic and semantic errors in the given C program.

Examples

```
prog.ck> cck1 | sort | cck2
```

Files

?*.ck file output by 'c1' with the '-y' option, used as input to 'cck1'.

Messages

Numerous and self explanatory.

Bugs

This program is only available to licensees of Version 2.0 of the Georgia Tech C Compiler.

See Also

cc (1), ccl (1), ucc (1), c1 (5), cck1 (5), *User's Guide to the Georgia Tech C Compiler*

Usage

csv

Description

'Csv' is a command for system administrators to ease the change when bringing up revision 9 of the Software Tools Subsystem. The shell variables save file format has changed at this revision and this command attempts to convert the variables files mechanically. 'Csv' accepts a list of user names on its first standard input port, attempts to open the corresponding variables file (hopefully "=vars=/<user_name>/.vars"), and changes the special Subsystem character mnemonics for the variables "_eof", "_erase", "_escape", "_kill", "_newline", and "_retype". The new shell variables are copied into a temporary file, which is then used to overwrite the user's permanent variables file.

This command is best run on an empty system because any users who are logged in during this execution will have their variables changed, but when they log out they will overwrite any changes that have been made. Also, the system administrator will have to change his variables manually because when he logs out he will overwrite any changes already made. The easiest way to execute this command is probably to list the files under "=vars=", remove any non-user files, and pipe the resulting list into 'csv'.

Examples

```
valid_users> csv
lf -c =vars= | =ebin=/csv
```

Messages

Self explanatory.

Files

=temp=/?* =vars=/<user_name>/.vars

Bugs

Could probably be a little more intelligent.

See Also

User's Guide for the Software Tools Subsystem Command

Interpreter, Software Tools Subsystem Manager's Guide

Usage

```
cvusr <old_userlist> <new_userlist>
```

Description

'Cvusr' is a simple shell script that takes two file names as arguments, the old, pre-Version 9 user list, and the new user list to be created. It simply pads six-character login names with blanks to be 32 characters long. It should be run once, by the system administrator, when the new Subsystem is installed. It is not needed at sites whose first edition of the Subsystem was Version 9.

Examples

```
=ebin=/cvusr //old_extra/users //extra/users
```

Messages

"Usage: ..." if called improperly.
"old user list is new user list!!" if both arguments are identical.

Bugs

Will create a strange user list if any of the old login names are longer than six characters.

See Also

Software Tools Subsystem Manager's Guide

Usage

guess <command> [<maxlevels>]

Description

'Guess' is a program which tries to discern the correct command when a misspelled command is entered. The program works by computing a "distance" between a misspelled command and commands in a predefined list. If any commands are found with a distance less than a predefined tolerance, 'guess' will present for selection all commands in the group that have the lowest distance. If this list contains only one command, it will ask for verification that it selected the right command. If this list contains more than one command, it prefaces each command by a number, and asks for the correct command to be selected by number. In either case, a response of a single carriage return means "don't execute anything." If the list has more than 10 commands in the group with lowest distance, 'guess' responds as the Subsystem normally does: "<command>: not found".

'Guess' searches through the file "=extra=/clist" which contains the system internal commands, commands from "=lbin=" and "=bin=". The user can define his own list to include his personal command directory by running the program 'mkclist', and this will create a file in the user's "bin" directory "=ubin=/clist".

Files

=ubin=/clist
=extra=/clist

Bugs

'Guess' will not consider commands that are accessible from the user's search rule, but not in one of the "clist" files.

Examples

<<'guess' should not normally be run from command level>>

Bugs

Locally supported.

See Also

bs (5), bs1 (5), mkclist (3)

Usage

```
mkcl [-s]
```

Description

'Mkcl' is not intended to be user-invoked; rather, it is a utility used by the 'mkclist' command to build a list of commands in a compressed binary format for the use of the 'guess' command. 'Mkcl' reads a list of command names from standard input, one name per line, and builds a binary output file. This binary output file contains the command names ordered by name length first and then alphabetical order; i.e., all the one-character commands come first in alphabetical order, then the two-character commands in alphabetical order, etc. File marks are used to allow fast locates of a command within the file.

If the optional argument "-s" is specified, then 'mkcl' generates a new system command file; otherwise, it generates a new user command file. Because binary output is used, the output of 'mkcl' should never be sent to the terminal, but to a file or to a pipe for further processing.

Examples

```
lf -c =bin= =lbin= =ebin= | sort | uniq | mkcl -s
lf -c =bin= =lbin= =ebin= =ubin= | sort | uniq | mkcl
```

Files

creates =extra=/clist if a system command list is desired.
creates =ubin=/clist if a per-user command list is desired.

Messages

"Usage: mkcl ..." if invalid arguments are specified.
"Can't create clist file" if trying to create a system command file from a nonowner account to the =extra= directory, or if trying to create a private user command list without having the directory =ubin= defined.
"Overflow!!!!!!!! argggggg..." if there are more commands than can be handled in the program's data area.
"writef returned an error" if the program could not write the file header of file marks.
"writef died in loop" if the program did not finish writing the command names to the command file.
"writef died on last writef" if the program could not update the file header with new information after writing out the commands.
"seekf returned error" if the program could not rewind the command file.

Bugs

If there are more than 600 commands or more than 4096 characters total in all the command names, table overflow occurs.

It could be hazardous to your terminal's health to copy the resulting command list file, since there may be some terminal control sequences embedded within the binary file.

Locally supported.

See Also

bs (5), bs1 (5), guess (5), mkclist (3)

Usage

ring

Description

'Ring' is a network communication server for Prime computers. When run as a privileged process on a node of the ringnet, it figures out who it is and who all the other nodes are, and then procedes to connect itself in a ring with its predecessor and successor using virtual circuits. Once connected, it will (currently) accept requests from users to execute commands on a legal remote node and pass the status back to the user. It also ensures that the system time (time of day, and date) is consistent among all the machines in the network.

'Ring' is unfinished but has many possibilities. The plans before the SWT project ended and its creator found another job were to set up a method for load sharing among computers in a network under the Software Tools Subsystem. The idea was to make a "/dev/net" device that would have its port number returned by a port server ('ring', or course) on the remote system. A shell would be cranked up on the remote system who's standard input would be a NET device. The source of the NET device would be the system where the user actually resided. This would allow the user (only under the Subsystem) to communicate with his process remotely.

Messages

Numerous. Sorry, but see the source code.

Bugs

Simply unfinished. Has tremendous possibilities.

See Also

broadcast (3), execute (3), setime (3), *Ring -- The Software Tools Subsystem Network Utility*

Usage

x snplnk 1/ [2/1]

Description

'Snplnk' is a Primos-executable routine that scans a given segment looking for dynamic library links (DYNT's) and forces the Primos ring 3 pointer fault handler to resolve the address. After 'snplnk' has been run on a segment, the segment may be shared non-writable if it contains pure code. It is actually meant to be run from Primos during library initialization rather than from SWT. is the desired segment to snap as an octal number, and the "2/1" causes 'snplnk' to print the name of the routine it is about to attempt to resolve along with the location of the pointer in the segment.

To use 'snplnk', the information should be installed in the segment and the segment should then be left writable. 'Snplnk' is then run on the segment and the segment can then be made non-writable. As an example, Georgia Tech has a file that is run at boot time called "swt.share.comi" that installs the SWT subsystem. The file contains:

```
/* SWT.SHARE.COMI, Share Software Tools Subsystem
/*    Last modified: 06/11/84

OPR 1
SHARE SYSTEM>SW2035 2035 700  /* Library
SHARE SYSTEM>SH2030 2030 700  /* Shell library
SHARE SYSTEM>ST2030 2030 700  /* SWT Initialization program
SHARE SYSTEM>SE2031 2031 700  /* Screen Editor
R SYSTEM>SW4000 1/1           /* Install the Library
R SYSTEM>SH4000 1/5           /* Shell library
R SYSTEM>INITSWT

SNPLNK 1/2030; SHARE 2030 600 /* Snap links and make the
SNPLNK 1/2031; SHARE 2031 600 /* segments not writeable
SNPLNK 1/2035; SHARE 2035 600
OPR 0

CO -CONTINUE 6
CO -END
```

The command "SNPLNK 1/2035; SHARE 2035 600" first snaps all the dynamic links in segment 2035 (the shared standard library) and then makes the segment non-writable. This prevents a user from altering the segment whether malicious or otherwise.

Messages

"A register setting missing" for missing

"Segment range is 2030-2037" for a segment out of that range

Bugs

Should be written to use reasonable argument handling.

Gets a pointer fault when trying to snap a link to a routine that does not exist.

The program attempts to be as intelligent as possible about what a link is and is not but mistakes can theoretically happen.

Usage

```
x sph <primos tree> [-u <user>] [-p <project>] [-g {<groups>}]
                    [-v <privilege>]
```

Description

'Sph' is a SWT supplied command that enables the system administrator to create phantoms with certain attributes (name, project, groups, or special phantom privileges) specified. It can be run only from the system console or, at Georgia Tech, by a user that has the .GURU group associated with his job. 'Sph' is a primos command, so if it is to be run from the subsystem, the 'x' command must be used to pass it directly to primos, or it must be called through the 'sys$$' subroutine.

The required argument <primos tree> is the primos treename of the file to phantom. This file is a *Primos* command file, not a SWT shell file or executable binary. All the remaining arguments are optional and, except for the '-v' option, default to the attributes that the caller currently has. The '-u' specifies the user name of the process to create and the '-p' option specifies the project. The '-g' option is followed by zero or more groups that the phantom is to have. The group names should not be preceded by a '.' (which is the Primos standard) because the 'sph' command will include them automatically. The '-v' option allows the caller to set the privilege word (prvl) in the Primos internal databases for the process privilege. Currently, the only useful values for this option are zero and one. Zero prevents the phantomed process from being able to execute 'sph' and one allows the programs to use 'sph'.

Messages

"Can't attach to <primos tree> (SPH)" for a non-attachable directory.

"Phantom is user <pid> on <date> at <time>" for a successful phantom.

Any Primos standard error message for any other exception conditions (by calling Primos ERRPR$).

Examples

```
x sph "system>cron.comi" -u cron -p lab -g guru -v 1
x sph "jeff>blerf" -u jeff -p blivnoxx -g
```

Bugs

Locally supported until Prime supports EPF's and the SPAWN$ subroutine call.

Should probably be written for SWT, also.

See Also

cron (3)

Section 6 - Low Level Library Subprograms

This section is designed to give the user a working knowledge of the low level functions and subroutines. This information is supplied for informative purposes only, since the user should not invoke these routines directly under normal circumstances; appropriate routines in sections two or four should be invoked instead. Each routine has its own entry organized under the following headings. Note that empty entries are omitted entirely.

Header Line

The subprogram's name, a synopsis of its purpose, and the date of last modification to its documentation.

Calling Information

The subprogram declaration and the declarations of its arguments, as well as the name of the library in which it can be found.

Function

A description of the purpose of the routine, along with the interpretations of its arguments and the returned value (if any).

Implementation

A short discussion of the strategy used to implement the routine, abstracted from the source code.

Arguments Modified

Names of those arguments modified by the routine.

Calls

Other subprograms called by this routine.

Bugs

Known problems with the use of the routine.

See Also

References to further information or related routines.

Calling Information

```
subroutine at$swt (name, namel, ldisk, passwd, key, code)
character name (MAXPATH)
packed_char passwd (3)
integer namel, ldisk, key, code
```

Library: vswtlb (standard Subsystem library)

Function

'At$swt' is the Subsystem interlude to the Primos ATCH$$ subroutine. It allows the program to attach to another directory, and takes the same arguments as ATCH$$. If there is an error in trying to reach the directory, 'at$swt' returns E$BPAS in 'code', instead of leaving the user in Primos.

'Name' is the name of the directory to attach to, 'namel' is the length of 'name', 'ldisk' is the number of the logical disk to be searched to find the given directory, 'passwd' is the password of the directory (the characters are packed two per word), 'key' is the composition of the 'REFERENCE' and 'SETHOME' subkeys (see the *Primos Subroutines Reference Guide*, PDR3621), and 'code' is an integer variable which contains the return code.

Implementation

'At$swt' first sets up an on-unit for the "BAD_PASSWORD$" condition before it tries to call the Primos ATCH$$ routine. It calls that Primos routine with all of its arguments (which are not processed in any way), and returns normally if there was no error in attaching to the directory. Any errors cause an error message to be issued and control is returned to the calling procedure.

Arguments Modified

code

Calls

Primos atch$$, Primos mkonu$, Primos pl1$nl

Bugs

Should be converted to use the new Primos 'at$?*' routines.

See Also

follow (2), getto (2), tscan$ (6), Primos atch$$

Calling Information

```
subroutine bponu$ (cp)
longint cp
```

Library: vswtlb (standard Subsystem library)

Function

'Bponu$' is an on-unit handler for the "BAD_PASSWORD$" condition. It is used by 'getto' to catch directory attaches which fail because of a bad password.

'Bponu$' should never be called by the user as such; it is invoked when the on-unit mechanism detects the "BAD_PASSWORD$" condition.

Implementation

'Bponu$' calls the Primos PL1$NL routine with the "bad password label" (i.e., address of the password error return location) to execute a "nonlocal goto" to that routine.

Calls

Primos pl1$nl

See Also

getto (2), Primos mkonu$

Calling Information

subroutine c$end

Library: vswtlb (standard Subsystem library)

Function

'C$end' is called from Ratfor programs that have been processed with the "-c" (statement count) option. Calls to 'c$end' are planted before each 'stop' statement in the program.

'C$end' simply writes out the statement count array to the file "_st_count" for later processing.

Implementation

The statement count array in common block 'c$stc' is written (by repeated calls to 'print') to the file "_st_count".

Calls

create, cant, print, close

See Also

c$incr (6), rp (1)

Calling Information

```
subroutine c$incr (stmt)
integer stmt
```

Library: vswtlb (standard Subsystem library)

Function

'C$incr' is called from Ratfor programs that have been processed with the "-c" (statement count) option. Calls to 'c$incr' are planted before each executable statement in the program to keep track of the number of times the corresponding statement was executed.

The sole argument is the line number (in the Ratfor source code) of the line containing the statement being executed. Each call to 'c$incr' with a given line number as argument causes the count for that line to be incremented by one.

Implementation

A common block ('c$stc'), created by Ratfor, contains an array of statement counts indexed by line number. 'C$incr' simply increments the appropriate element of the array.

See Also

c$end (6), rp (1)

Calling Information

subroutine c$init

Function

'C$init' is called at the beginning of the main program in Ratfor programs that have been processed with the "-c" (statement count) option of 'rp'. It initializes the statement count array for statement count processing.

'C$init' is inserted into the Fortran output as inline code, rather than being referenced from the standard Subsystem library. As such, it can never be accessed by the user unless the "-c" option is specified (even then, it should not be called by the user, since the statement counts will be erroneously modified).

Implementation

A Fortran **do loop** is used to initialize all of the elements in the statement count array to zero.

See Also

c$end (6), c$incr (6), rp (1)

Calling Information

```
integer function call$$ (name, length[, onunit])
integer name (16), length
external onunit
```

Library: vswtlb (standard Subsystem library)

Function

'Call$$' takes the packed name and name length of a P300 format run file, a SEG segment directory, or EPF run file. First 'call$$' attempts to restore the file with a call to the Primos routine REST$$. If REST$$ returns unsuccessfully, 'call$$' attempts to load the file as a SEG run file through a call to 'ldseg$'. If 'ldseg$' returns because the file was not a segment directory, R$RUN is called with the restore option to attempt to load the file as an EPF. If all attempts to load the file fail, 'call$$' returns ERR. 'Onunit', if specified, indicates that the shell's ANY$ onunit is to be created.

'Call$$' returns (with value OK) if and only if the program it calls exits by calling 'swt' or does a procedure return from its main procedure. Otherwise, control goes wherever the called program sends it.

Before executing the run file, 'call$$' zeroes out the P300 fault vector in segment 4000, zeroes the program error return code, calls 'iofl$' to mark which Subsystem file units are open, and saves the stack base register in the Subsystem common block for use by 'rtn$$'.

Implementation

'Call$$' first zeroes the program error return code and the P300 fault vector. It then tries to load the run file in memory with a call to REST$$. If there is an error on the restore, 'call$$' calls 'ldseg$' to load the file as a SEG run file. If 'ldseg$' fails because the file is not a segment directory, R$RUN is called to restore the file in memory as an EPF. If R$RUN returns an error, 'call$$' returns ERR.

For P300 run files and SEG segment directories, if the program just loaded begins in 64V mode, 'call$$' executes a PCL instruction to the address of its main entry control block. Otherwise, 'call$$' builds an R or S mode entry control block for the program in the stack. After setting up an onunit for ANY$ via a call to the Primos routine MKONU$ (if the user specified a third argument), 'call$$' executes a PCL instruction to the correct entry control block. If the file is an EPF run file, the onunit for ANY$

is still set (if requested), but then R$INVK is called to start execution of the file.

When the called program returns directly to 'call$$' from 'rtn$$', 'call$$' calls 'cof$' to close all files opened by the program, restores the user's terminal configuration word (saving the output suppressed bit) via calls to Primos DUPLX$, restores the previous saved stack base register and returns with the value OK.

Calls

cof$, iofl$, ldseg$, move$, Primos break$, Primos duplx$, Primos mkonu$, Primos rest$$, Primos rvonu$, Primos r$run, Primos r$invk

Bugs

Will destroy the current executing memory image if the object must be loaded at the same addresses.

The ability to execute EPF's is not really supported until Prime decides to support EPF's.

See Also

rtn$$ (6), swt (2)

Calling Information

```
integer function chunk$ (bp, seg, fd)
longint bp
integer seg, fd
```

Library: vswtlb (standard Subsystem library)

Function

'Chunk$' expects the segment directory to be open on 'fd' (a Primos file descriptor). It opens the file in the segment directory at position 'seg + 2' and reads 2048 words into memory at position pointed to by 'bp'. The function return is OK if the read was successful, and ERR if any errors occur.

Implementation

Straightforward through calls to the Primos routines SGDR$$, SRCH$$, and PRWF$$.

Calls

Primos sgdr$$, Primos srch$$, Primos prwf$$

See Also

ldseg$ (6), zmem$ (6)

Calling Information

```
subroutine cof$ (state)
integer state (MAXFILESTATE)
```

Library: vswtlb (standard Subsystem library)

Function

When called, 'cof$' closes all open files that were opened after the last call to 'iofl$'. 'Cof$' also resets the terminal input buffer pointer and character count in the Subsystem common block.

Implementation

'Cof$' checks the flag word of each of the file descriptors in 'state' up to the ERR marker. If the file is currently open, 'cof$' calls 'close' to close it.

Next, 'cof$' skips the ERR marker, and calls the Primos routine SRCH$$ to close all of the Primos files indicated by the second list in 'state' (up to the next ERR marker).

Lastly, 'cof$' resets the terminal input buffer pointer to 1 and the terminal buffer character count to 0.

Calls

close, Primos srch$$

See Also

iofl$ (6), close (2), open (2)

Calling Information

```
subroutine cpfil$ (ifd, ofd, rc)
integer ifd, ofd, rc
```

Library: vswtlb (standard Subsystem library)

Function

'Cpfil$' expects 'ifd' to contain the Primos file unit number of a file open for reading, and 'ofd' to contain the Primos file unit number of a file open for writing. 'Cpfil$' attempts to copy the contents of the input file to the output file. If any condition arises that prevents completion of the copy, 'cpfil$' sets 'rc' to ERR; otherwise, it sets it to OK. On return, both files are left open and positioned to the end.

Implementation

'Cpfil$' makes repeated calls to Primos PRWF$$ with a large buffer to quickly move the data between the files.

Arguments Modified

rc

Calls

Primos prwf$$

See Also

cpseg$ (6), filcpy (2), fcopy (2)

Calling Information

```
subroutine cpseg$ (ifd, ofd, rc)
integer ifd, ofd, rc
```

Library: vswtlb (standard Subsystem library)

Function

'Cpseg$' expects 'ifd' to contain the Primos file unit of a segment directory open for reading and 'ofd' to contain the Primos file unit of an empty segment directory open for writing. 'Cpseg$' attempts to make an exact copy of the input segment directory in the output segment directory. If it is successful, it sets 'rc' to OK; otherwise, it sets 'rc' to ERR.

Implementation

'Cpseg$' scans the open segment directory with the Primos routine SGDR$$, calling 'cpfil$' to copy files, and calling itself recursively to copy nested segment directories.

Arguments Modified

rc

Calls

cpfil$, cpseg$, Primos sgdr$$, Primos srch$$

See Also

cpfil$ (6), filcpy (2)

Calling Information

```
integer function dgetl$ (line, length, fd)
integer length
character line (length)
file_descriptor_struct fd
```

Library: vswtlb (standard Subsystem library)

Function

'Dgetl$' is an internal Subsystem routine that performs the function of 'getlin' for disk files only. The first argument specifies a string to receive the line read; the second argument is the length of the longest string that will fit in the receiving buffer; the third is a pointer to the appropriate file descriptor structure in the Subsystem common area. 'Dgetl$' returns the number of characters placed in the receiving buffer (excluding EOS) if the read was successful; EOF otherwise. 'Dgetl$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'getlin'.

Implementation

'Dgetl$' (which is written in PMA, incidentally) shortcalls an internal routine that calls the Primos routine PRWF$$ to read a buffer full of data from the disk file selected by the file descriptor. This buffer is then unpacked into the user's receiving string. During the unpack and copy operation, compressed blanks (encoded as an RHT (relative horizontal tab) followed by a blank count) are converted into the proper number of ordinary blanks. The copy operation ends when a NEWLINE is encountered or when the user's buffer is full.

Arguments Modified

line

Calls

Primos prwf$$

See Also

getlin (2), tgetl$ (6), putlin (2), dputl$ (6), tputl$ (6)

Calling Information

```
file_mark function dmark$ (f)
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Dmark$' performs the function of 'markf' for disk files. The single argument is the file descriptor of a disk file; the function return is the current file pointer value for the selected file. ERR is returned if the position of the file could not be determined.

As with all Subsystem routines whose names contain the dollar sign ($), 'dmark$' is not intended for general use. 'Markf' is normally used to provide the required functionality.

Implementation

The Primos routine PRWF$$ is used to return the current file position, which is in units of words past the beginning of file. If for any reason PRWF$$ cannot determine the position, 'dmark$' returns ERR.

Calls

Primos prwf$$

See Also

markf (2), tmark$ (6), seekf (2)

Calling Information

```
subroutine dmpcm$ (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Dmpcm$' outputs the contents of the Subsystem's common blocks in a printable format. Unprintable characters are mapped into a mnemonic format, and output is appropriately titled.

'Fd' is the file descriptor of the file unit which should receive the information.

Implementation

The common area values which may be unprintable are mapped into mnemonic strings by calls to the routine 'ctomn'. Then, the value of each variable in the common area is printed, with the appropriate headings.

Calls

ctomn, print

See Also

dump (1)

Calling Information

```
subroutine dmpfd$ (fd, ofd)
file_des fd, ofd
```

Library: vswtlb (standard Subsystem library)

Function

'Dmpfd$' prints all information that is of importance to the user about file descriptor 'fd' on file unit 'ofd'. Among the items of information produced are the file name (if obtainable), file position (if a disk file), file buffer information, the most recent file system return code, and the contents of the file buffer (if a disk file). Each of these pieces of information is displayed with the appropriate heading.

Implementation

'Gfnam$' is called to obtain the name of the file associated with the descriptor. If the name could be obtained, it is printed out. Other file unit information is printed, in the proper format, from information stored in the Subsystem common areas. The current position in a disk file is obtained by calling the Primos routine PRWF$$.

Calls

gfnam$, mapsu, print, putch, putlin, Primos prwf$$

See Also

dump (1)

Calling Information

```
integer function dopen$ (path, fd, mode[, typ[, limit]])
character path (ARB)
integer mode, typ, limit
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Dopen$' is an internal Subsystem routine that performs the function of 'open' for disk files only. The first argument is the pathname of the file to be opened; it must be an EOS terminated string (i.e. dopen$('/dir/file1's....). The second argument is the file descriptor assigned to the file in 'open'. The third argument is the mode, READ, WRITE or READWRITE. The fourth argument is optional. It specifies the type of the file. The fifth argument is optional; it specifies the number of times to retry the open if the file is in use. 'Dopen$' returns the value of 'fd' if the attempt to open was successful; ERR if the attempt failed. The user is always left in the home directory after an attempt to open.

By default, 'dopen$' returns a file descriptor to a sequential access method (SAM) file. If creating a direct access method file (DAM) is desired, the 'mode' argument may be ORed with the KNDAM file key (i.e., 'mode' can be "READ-WRITE+KNDAM" to create a DAM file opened for reading or writing). The constant KNDAM is contained in the "PRIMOS_KEYS" include file.

Implementation

'Dopen$' calls 'getto' to reach the UFD containing the desired file and pack the filename into an array suitable for use with Primos routines. If 'getto' is successful, the Primos subroutine SRCH$$ is called to open the file. If either 'getto' or SRCH$$ fails, Primos AT$HOM is called to return the user to the home directory, and ERR is returned as the function value of 'dopen$'.

Arguments Modified

typ

Calls

getto, Primos at$hom, Primos missin, Primos srch$$, Primos sleep$

See Also

open (2)

Calling Information

```
integer function dputl$ (line, fd)
character line (ARB)
file_descriptor_struct fd
```

Library: vswtlb (standard Subsystem library)

Function

'Dputl$' is called by 'putlin' to write a line on a disk file. The first argument is an EOS-terminated string to be placed on the disk file; the second argument is the file descriptor of the file on which the string is to be written. The function return is OK for a successful call, ERR otherwise. 'Dputl$' is not protected from user error, and so should not be used except as it is called by 'putlin'.

Implementation

'Dputl$' maintains a count of blanks to be used for file compression. When a non-blank character is encountered in the string, any blanks accumulated are translated to a relative horizontal tab (RHT) and a blank count, and the non-blank character is output. Characters placed in the disk buffer are output by a shortcalled routine internal to 'dputl$'; this routine calls the Primos routine PRWF$$ to do the actual data transfer.

Calls

Primos prwf$$

See Also

putlin (2), dgetl$ (6), tputl$ (6)

Calling Information

```
integer function dread$ (buf, nw, f)
integer buf (ARB), nw
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Dread$' is an internal Subsystem routine that performs the function of 'readf' for disk files only. The first argument specifies a string to receive the words read; the second argument is the number of words to be read; and, the third argument is the file descriptor of the file from which data will be read. 'Dread$' returns the number of words placed in the receiving buffer if the read was successful; EOF otherwise. 'Dread$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'readf'.

Implementation

'Dread$' calls the Primos subroutine PRWF$$ to fill a buffer with words from disk file 'f'.

Arguments Modified

buf

Calls

move$, Primos prwf$$

Bugs

EOF is returned if any error occurs; the user is not informed of the actual error that occurs.

See Also

readf (2)

Calling Information

```
subroutine dsdbiu (block, form)
pointer block
character form
```

Library: vswtlb (standard Subsystem library)

Function

'Dsdbiu' is called by 'dsdump' to dump the contents of a block of storage that has been allocated by 'dsget'. The first argument is a pointer to the control words of the block; the second is LETTER for a character dump, DIGIT for a numeric dump.

This routine is technically not available for direct call by the user, since the format and location of block control words is subject to change.

Implementation

The SIZE control word of the block is read to obtain the size of the block, and that many words are written to ERROUT via 'print' in the particular format specified. The first argument is incremented to point to the end of the block.

Arguments Modified

block

Calls

print

Bugs

None that can be helped.

See Also

dsget (2), dsfree (2), dsinit (2), dsdump (2)

Calling Information

```
integer function dseek$ (pos, f, ra)
file_mark pos
file_des f
integer ra
```

Library: vswtlb (standard Subsystem library)

Function

'Dseek$' is an internal Subsystem routine that performs the function of 'seekf' for disk files only. The first argument is a long integer value which specifies the amount of relative or absolute positioning, depending on the value of the third argument, 'ra'. If 'ra' equals ABS then positioning is from the beginning of the file; if 'ra' equals REL then positioning is from the current position. The second argument is the file descriptor of the file whose file pointer is being manipulated. The function return is OK if the positioning was successful, ERR if 'ra' is ABS and 'pos' is negative, ERR if 'ra' is neither ABS nor REL, and EOF otherwise. 'Dseek$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'seekf'.

Implementation

'Dseek$' calls the Primos subroutine PRWF$$ to set the file pointer of a disk file.

Calls

Primos prwf$$

Bugs

EOF is returned if any error occurs during disk read; the user is not informed of the actual error that occurs.

See Also

seekf (2)

Calling Information

```
integer function dwrit$ (buf, nwx, f)
integer buf (ARB), nwx, f
```

Library: vswtlb (standard Subsystem library)

Function

'Dwrit$' is an internal Subsystem routine that performs the function of 'writef' for disk files only. The first argument is the array of words to be written to the file; the second argument is the number of words to be written; the third argument is the file descriptor of the file to which data will be written. 'Dwrit$' returns the number of words written (which should always equal 'nwx'), or EOF. 'Dwrit$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'writef'.

Implementation

'Dwrit$' calls the Primos subroutine PRWF$$ to write words to disk.

Calls

Primos prwf$$, move$

Bugs

EOF is returned if any error occurs; the user is not informed of the actual error that occurs.

See Also

writef (2)

Calling Information

```
integer function findf$ (file)
packedchar file (16)
```

Library: vswtlb (standard Subsystem library)

Function

'Findf$' is an internal routine used to verify the existence of a file. The argument is a packed, blank-padded character string (such as that returned by 'getto') that is 32 characters in length. 'Findf$' returns YES if the file exists in the current directory, NO otherwise.

Implementation

'Findf$' calls the Primos routine SRCH$$ with the key KEXST to determine if the named file exists.

Calls

Primos srch$$

See Also

getto (2), file (1)

Calling Information

```
integer function finfo$ (path, entry, attach)
character path (ARB)
integer entry (MAXDIRENTRY), attach
```

Library: vswtlb (standard Subsystem library)

Function

'Finfo$' is an internal Subsystem routine used to return the Primos directory entry associated with a named file. The 'path' argument is the pathname of the file whose entry is desired; 'entry' is a buffer to receive the entry itself; 'attach' is set to YES if the user's attach point changed as a side effect of obtaining the directory entry, NO otherwise. The function return is OK if the directory entry was obtained, ERR otherwise.

See Prime's File Management System guide for information on the structure of directory entries as returned by the Primos routines DIR$RD and ENT$RD.

Implementation

'Getto' is called to attach to the parent directory of the named file. The 'attach' parameter is set as a side effect of this action. The Primos routine SRCH$$ is then used to open the current directory for reading, and the Primos routine ENT$RD to fetch the entry for the named file. The current directory is then closed by SRCH$$ and 'finfo$' returns.

Arguments Modified

entry, attach

Calls

getto, Primos srch$$, Primos ent$rd

See Also

filtst (2), file (1), lf (1)

Calling Information

```
integer function first$ (flag)
integer flag
```

Library: vswtlb (standard Subsystem library)

Function

'First$' checks to see if this is the first call to itself. If it is being called for the first time, then it returns YES; otherwise, it returns NO. 'Flag' is set to the return value, in either case.

'First$' is used by the 'swt' command to prevent further calls to itself when a previous invocation is still active.

Implementation

'First$' checks the Subsystem common area variable 'first_use' to see if it contains a special value; if it doesn't, then a YES is returned and this special value is set. If it finds the special value, then it returns NO.

Arguments Modified

flag

Calling Information

```
integer function flush$ (fd)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Flush$' is used to clean up the state of the internal Subsystem buffers associated with an open file. In general, this is necessary before changing access mode on a disk file (e.g., from read to write or from character to block) and when closing a file (to insure that all data is transferred from the buffer to disk).

The single argument to 'flush$' is the file descriptor (returned by 'open', 'create', or 'mktemp') of the file whose buffer is to be flushed. The function return is OK if the flush succeeded and ERR if it failed.

Although it sees a great deal of use internally, 'flush$' is practically useless to the general user. The only circumstance in which its use might be appropriate is when a log file or audit trail must be written to disk as frequently as transactions occur; in such a case, the disk I/O must not be buffered.

Implementation

The action of 'flush$' varies according to the device assigned to the file and the last operation performed. In all cases, buffer pointers and character counts must be reinitialized. For terminal devices, no other action is required. If the last operation performed on a disk file was a 'putlin' or 'putch', then any pending compressed blanks must be forced out and the buffer must be written to disk (via the Primos routine PRWF$$). If the last operation was a 'getlin' or 'getch', then it is necessary to back up the file's current position to the point at which the unused portion of the buffer begins; a call to PRWF$$ does the actual repositioning. If the last operation was a 'writef', any words remaining in the buffer are simply written out with PRWF$$. Finally, if the last operation was a 'readf', the file's current position is simply backed up by the number of unused words still in the buffer.

Calls

dputl$, Primos prwf$$, Primos break$

Calling Information

```
integer function gcdir$ (path)
character path (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Gcdir$' is used to determine the full pathname of the current working directory. The sole argument is a character array to receive the pathname. The function return is OK if the name could be found, ERR otherwise.

Implementation

'Gcdir$' first obtains the treename of the current directory using the Primos routine GPATH$. This treename is then unpacked by 'ptoc' and passed to 'mkpa$', which converts it into a SWT pathname.

Arguments Modified

path

Calls

mkpa$, ptoc, Primos gpath$

Bugs

Be warned that because of Prime's password protection scheme, it is not always possible to obtain a pathname that can later be used to attach to the home directory with the same access rights.

See Also

mktr$ (6), mkpa$ (6), follow (2), getto (2)

Calling Information

```
integer function gcifu$ (funit)
integer funit
```

Library: vswtlb (standard Subsystem library)

Function

'Gcifu$' returns the file unit which is providing command input for the shell both in the argument 'funit' and as the value of the function.

Implementation

'Gcifu$' returns the value contained in 'Comunit' in the Subsystem common block.

Arguments Modified

funit

See Also

sh (1)

Calling Information

```
file_des function getfd$ (fd)
file_des fd
```

Library: vswtlb (standard Subystem library)

Function

'Getfd$' is used by 'open' and 'mkfd$' to find an unused file descriptor with which to set up a file unit. If it could find one, it returns that file descriptor; otherwise, it returns ERR.

Implementation

The file descriptor list is searched to find one that is available. The search is attempted first on file descriptors that lie within the current page of memory. If one is not found, the search is then performed on any remaining file descriptors (possibly requiring paging to bring in the required data); if a free descriptor is found, then it is returned to the caller. If none are found this time, ERR is returned.

Arguments Modified

fd

See Also

mkfd$ (6), open (2)

Calling Information

```
integer function gfnam$ (fd, path, size)
file_des fd
character path (MAXPATH)
integer size
```

Library: vswtlb (standard Subsystem library)

Function

'Gfnam$' tries to find the name of the open file unit 'fd'. If the unit is a terminal, it returns the name of the terminal device. If the unit is a null device, then it returns the null device name; otherwise, it obtains the pathname and returns it in 'path'. If the pathname can be obtained, the length of 'path' is the returned value; otherwise ERR is returned.

'Size' is the number of characters that can be held in 'path' (including EOS).

Implementation

'Gfnam$' first tries to verify that the file unit for which a name is desired is open and a legal file unit; if it isn't both, then ERR is returned. Otherwise, it checks to see what type of file is associated with the given file descriptor. For terminal and null devices, the appropriate device name is returned (device names are of the form '/dev/?*'). For disk files, the Primos GPATH$ subroutine is called to obtain the Primos treename for the file. If a treename could be obtained, then 'mkpa$' is called to generate a Subsystem pathname for the file; otherwise, ERR is returned.

Arguments Modified

path

Calls

ctoc, Primos gpath$, mapsu, mkpa$, ptoc

Calling Information

```
integer function gtacl$ (path, key, at)
character path(ARB)
integer key, at
```

Library: vswtlb (standard Subsystem library)

Function

If 'key' is 1, 'gtacl$' retrieves the standard ACL protection for the file 'path' into the ACL common block, or if 'key' is 2, it returns the priority ACL protection into the ACL common block. 'At' is set to YES if the current attach was moved to get to the specified file. The function return is OK if the information was was retrieved and ERR otherwise.

Implementation

'Gtacl$' attempts to attach to the directory containing the file and then procedes to retrieve the acl information. It then scans through the returned information and formats it for further use in the common blocks. If any error is encountered it attaches home if the attach point has changed and returns an error, otherwise it returns OK.

Calls

ctov (2), equal (2), follow (2), getto (2), mkpa$ (2), mktr$ (2), mapstr (2), vtoc (2), Primos pa$lst, Primos ac$lst

See Also

gfdata (2), sfdata (2)

Calling Information

subroutine icomn$

Library: vswtlb (standard Subsystem library)

Function

'Icomn$' is used to completely reinitialize all Subsystem common areas. At present, it is used only by the program 'swt' on Subsystem entry.

Implementation

'Icomn$' initializes the Subsystem password, argument count, command interpreter status flag, shell error code, command input unit, user template storage area, Subsystem return label, and linked string control words, then calls 'ioinit' to set up the input/output common blocks.

Calls

ioinit

See Also

ioinit (6)

Calling Information

```
subroutine iofl$ (state)
integer state (MAXFILESTATE)
```

Library: vswtlb (standard Subsystem library)

Function

'Iofl$' saves the Subsystem and Primos file descriptors which correspond to closed files in 'state', so that subsequently opened files can be closed by 'cof$'.

Implementation

For each Subsystem file descriptor, 'iofl$' examines the its flag word to determine if it is closed. For each closed file, its descriptor is saved in 'state'. After the last closed Subsystem file descriptor entered into 'state', ERR is entered into 'state' to mark the end of the list.

Next, for each Primos file descriptor, 'iofl$' calls the Primos routine PRWF$$ to test whether or not the file is open and valid. For each valid closed file, its file descriptor is entered into 'state'. After the last valid closed Primos file descriptor has been entered, ERR is entered into 'state' to mark the end of the list.

Calls

Primos prwf$$

See Also

cof$ (6)

Calling Information

subroutine ioinit

Library: vswtlb (standard Subsystem library)

Function

'Ioinit' reinitializes certain control words in the Subsystem's input/output common block. At present, it is used solely for starting the Subsystem from scratch.

Implementation

'Ioinit' sets the erase character, kill character, repeat character, escape character, terminal character buffer pointer, Subsystem newline character, kill response (the string that gets printed when a kill character is encountered), terminal attributes, and terminal character count. In addition, it "opens" the user's terminal on the file descriptor designated by the macro TTY and sets all other file descriptors to "closed", and sets the Subsystem printer form and printer destination. Finally, it sets all entries in the standard port table to TTY.

Calls

ctoc

See Also

icomn$ (6)

Calling Information

```
subroutine ldseg$ (rvec, name, len, code)
integer rvec (9), name (ARB), len, code
```

Library: vswtlb (standard Subsystem library)

Function

'Ldseg$' first attempts to open the file 'name' in the current directory, using 'len' as the length of the name. If the open is successful, and the file is a SEG run file of recent (Primos revision 17 or later) origin, 'ldseg$' loads the private segments of the file into memory and sets 'rvec' to the initial save vector of the program. If the load is successful, 'ldseg$' returns a code of 0; otherwise, the Primos error code E$BPAR is returned.

Implementation

'Ldseg$' first opens the segment directory and file 0 in the directory. Using calls to the Primos routine PRWF$$, 'ldseg$' reads and checks the revision flag, segment map, segment bit map, save vector, time vector, and symbol table. Using this information, 'ldseg$' traverses the symbol table, loading initialized chunks of segments with calls to 'chunk$' and zeroing uninitialized segments with calls to 'zmem$'. Completely uninitialized segments remain unmodified. After loading is complete, 'ldseg$' sets the values in 'rvec' and returns with a code of 0.

Arguments Modified

rvec, code

Calls

chunk$, print, zmem$, Primos errpr$, Primos srch$$, Primos prwf$$

See Also

call$$ (6)

Calling Information

subroutine ldtmp$

Library: vswtlb (standard Subsystem library)

Function

Using the public template "=utemplate=" to locate the user's private template file, 'ldtmp$' reloads the hash table in the Subsystem common area that contains the private templates.

Implementation

'Ldtmp$' first zeroes the private template area, so the reference to "=utemplate=" will be to the public template. It then opens "=utemplate=", parses the lines with 'gtemp' and fills in the hash table.

Calls

close, getlin, gtemp, open, Primos break$, length, print, seterr, scopy

See Also

expand (2)

Calling Information

```
integer function lookac (name)
character name (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Lookac' returns the index of 'name' in the ACL common block or ERR if 'name' is not located.

Implementation

A linear search is used to scan the common block for the name. If the name is found, its index is returned, otherwise the routine returns ERR.

Calls

equal (2)

See Also

lacl (1), sacl (1), gfdata (2), sfdata (2)

Calling Information

```
file_des function lopen$ (path, fd, mode)
character path (ARB)
file_des fd
integer mode
```

Library: vswtlb (standard Subsystem library)

Function

'Lopen$' is an internal Subsystem routine that performs the function of 'open' for disk files in the spool queue only. The first argument is the pathname of the file to be opened; it must be an EOS terminated string specifying a spooler device file (e.g. "/dev/lps"s). The second argument is the file descriptor assigned to the file in 'open'. The third argument is the mode, READ, WRITE or READWRITE. 'Lopen$' returns the value of 'fd' if the attempt to open was successful; ERR if the attempt failed. The user is always left in the home directory.

Implementation

'Lopen$' examines the pathname for line printer spooler options (see 'open'). The Subsystem printer form and printer destination shell variables are used as the spool file's form type and destination, respectively, if they are defined; otherwise, the default installation form and printer destination are used. The Primos routine SPOOL$ is then called to open a spool file on disk, which may be written by the standard Subsystem disk I/O routines.

Calls

ctop, mapdn, mapup, ctoi, Primos srch$$, Primos spool$, putch, parstm, mapstr, ctoc

See Also

open (2), dopen$ (6), sp (1), pr (1)

Calling Information

```
integer function lutemp (temp, str, strlen)
integer strlen
character temp (ARB), str (strlen)
```

Library: vswtlb (standard Subsystem library)

Function

'Lutemp' converts a single template into its equivalent string representation. The argument 'temp' is the template to be expanded; 'str' is the string to receive the expansion of at most 'strlen' characters. The function returns the length of the expanded string contained in 'str' if the template was found, EOF otherwise.

Normally, the routine 'expand' would be called to expand a template, since it rescans the text returned by 'lutemp' to evaluate any nested templates.

The following dynamic templates are available:

date	the current date, in form MMDDYY
time	the current time, in form HHMMSS
user	the login name of the user calling 'expand'
pid	the process id of the process calling 'expand'
passwd	the Subsystem password of the user calling 'expand'
day	the name of the current day of the week (e.g. tuesday)
home	the login directory of the user calling 'expand'

The statically-defined templates reside in the file "=template=", and may be changed at the discretion of the Subsystem manager. For a complete list of templates, see the *User's Guide to the Primos File System*.

Users may create their own personal templates by placing template names and replacement text in the file "=utemplate=" (nominally "=varsdir=/.template"). The template file is a standard text file which may be manipulated by any of the usual text processing tools. Each template appears on a line by itself, followed by blanks and its replacement text. Blank lines and comment lines (beginning with "#") may be added to the template file to improve readability. For an example of a template file, see

=template=.

Implementation

'Lutemp' first looks up the requested template in a hashed symbol table, which contains the values of all "static" (determined at Subsystem initialization time) templates and resides in the shared Subsystem data area, checking the user's personal templates and then the system-defined templates. If the search succeeds, 'lutemp' returns the string value associated with the template. Otherwise, 'lutemp' assumes that the template is dynamic and searches a second shared hash table containing the values of dynamic template. If it finds the template in this table, 'lutemp' uses an associated function code value to direct appropriate calls to 'date' (for time, date, day, pid, user) or to file system routines (for home), or to read values from Subsystem common areas (for passwd).

Arguments Modified

str

Calls

equal, date, gcdir$, length, mapstr, scopy, Primos at$hom, Primos at$or

Bugs

There is no protection against setting static values for the dynamic templates. The user can possibly cause problems for both himself and other Subsystem users by setting his own values for the dynamic template names.

See Also

expand (2), open (2), getto (2), follow (2), *User's Guide to the Primos File System*

Calling Information

```
integer function mkdir$ (name, owner, non_owner)
character name (ARB), owner (ARB), non_owner (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Mkdir$' is used to create a new directory. The argument 'name' is the pathname of the directory to be created; the arguments 'owner' and 'non_owner' specify the owner and non-owner passwords, respectively, of the new directory. The function return is OK if the directory was successfully created, ERR otherwise.

Implementation

'Getto' is called to attach to the directory which will become the parent of the new directory. A call to 'findf$' insures that the directory does not already exist. The password strings are converted to packed format via calls to 'ctop'. The Primos routine CREA$$ actually creates the directory and sets the passwords. Then the Primos routine SATR$$ is called to set the protection so that the owner has all rights and non-owner has read access.

Calls

ctop, findf$, getto, Primos at$hom, Primos crea$$, Primos satr$$

See Also

follow (2), getto (2), mkdir (1)

Calling Information

```
file_des function mkfd$ (funit, mode)
integer funit, mode
```

Library: vswtlb (standard Subsystem library)

Function

'Mkfd$' allocates a Subsystem file descriptor for a disk file and initializes it so that it refers to the file open on the Primos funit number given as the argument 'funit'. 'Mode' must be READ, WRITE, or READWRITE. The function return is a file descriptor if the allocation succeeds, ERR otherwise.

'Mkfd$' is normally used to enable Subsystem I/O on a file that for some reason has already been opened by a Primos routine.

Implementation

A Subsystem file descriptor is allocated from the available pool (by a call to 'getfd$') and initialized as per 'open'. The given I/O mode, file unit, and disk device status are associated with the descriptor.

Calls

getfd$, Primos break$

See Also

getfd$ (6), mapfd (2), open (2)

Calling Information

```
integer function mkpa$ (tree, path, default)
character path (MAXPATH), tree (ARB)
integer default
```

Library: vswtlb (standard Subsystem library)

Function

'Mkpa$' is used to convert a Primos treename into an equivalent Subsystem pathname. The first argument is the treename to be converted. The second argument is a string to receive the equivalent pathname. The last argument is used to resolve an ambiguity in Primos treenames; if it equals YES, then simple names are interpreted as top-level user file directories, otherwise simple names are interpreted as files in the current directory.

The function return is the length of the pathname returned in 'path'.

The following conversions are performed:

```
<name>dir>subdir>file -> /name/dir/subdir/file
dir>subdir>file       -> //dir/subdir/file
*>subdir>file         -> subdir/file
simplename            -> simplename
                            (if 'default' == NO)
                      -> //simplename
                            (if 'default' == YES)
```

Implementation

Simple checks determine which of the above cases applies, then translation is straightforward.

Arguments Modified

path

Calls

scopy, mapdn, index

See Also

mktr$ (6)

Calling Information

subroutine mkpacl

Library: vswtlb (standard Subsystem library)

Function

'Mkpacl' converts ACL information like that returned by 'gtacl$' into Primos ACL information in the ACL common block.

Implementation

The ACL common block is scanned for information and encoded into an EOS-terminated string a character at a time. When finished, a call to 'ctov' converts the information into a form that Primos can use.

Calls

ctoc (2), ctov (2), encode (2)

See Also

lacl (1), sacl (1), gfdata (2), sfdata (2)

Calling Information

```
integer function mksacl (path, pairs, type, sep)
character path (ARB), pairs (ARB), sep (ARB)
integer type
```

Library: vswtlb (standard Subsystem library)

Function

'Mksacl' converts ACL information like that returned by 'gtacl$' into SWT ACL information in the ACL common block. The name of the pathname is returned in 'path', the string containing the access pairs is returned in 'pairs', and the type is returned in 'type'. 'Sep' is a string that is to be placed between each of the access pairs. The function return is the number of characters in 'pairs'.

Implementation

The ACL common block is scanned for information and encoded into 'pairs'. After each pair is entered, 'sep' is encoded into the string. The number of characters is returned as the function return.

Calls

ctoc (2), encode (2)

See Also

lacl (1), sacl (1), gfdata (2), sfdata (2)

Calling Information

```
integer function mktr$ (path, tree)
character path (ARB), tree (MAXPATH)
```

Library: vswtlb (standard Subsystem library)

Function

'Mktr$' is used to convert a Subsystem pathname into an equivalent Primos treename. The argument 'path' is an EOS-terminated string containing the pathname to be converted. The argument 'tree' is a string that will receive the equivalent Primos treename. The function return is the length of the treename returned in 'tree'.

The pathname may begin with a series of backslashes ("\"), each of which indicates a one-level ascension in the directory hierarchy. For example, the pathname "\" means the directory which is the parent of the current directory, and "\\file2" means the file named "file2" in the grandparent of the current directory.

Slashes in the input pathname that are preceded by an at-sign ("@") are passed through to the treename unchanged; they are not interpreted as separator characters.

Multiple slashes (except at the beginning of the path) are ignored.

Implementation

The first characters in the pathname determine the initial portion of the treename. If there are two leading slashes, then the treename begins with "mfd". If there is only one leading slash, then a packname was specified and the treename begins with "<packname>mfd". If there are leading backslashes, then the Primos routine GPATH$ is called to get the name of the current directory, and the appropriate portion becomes the start of the treename. The remainder of the conversion consists mostly of substituting slashes for greater-than signs and handling escape sequences.

Arguments Modified

tree

Calls

scopy, Primos gpath$, ptoc, mapstr

See Also

mkpa$ (6), follow (2), getto (2)

Calling Information

```
integer function parsa$ (str)
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Parsa$' compares the protections given in 'str' with those already in the common block and modifies the common block to reflect the changes. If the changes are made, the function return is OK, otherwise the function returns ERR.

Implementation

'Parsa$' goes through 'str' one pair at a time, calling 'lookac' to locate a corresponding name and then comparing the differences. It then changes the common block to reflect any modifications that have been made and goes through and removes any deleted entries. If there are any parse errors or erroneous attributes in 'str' the function returns ERR.

Calls

equal (2), lookac (6), scopy (2)

See Also

lacl (1), sacl (1), gfdata (2), sfdata (2)

Calling Information

```
subroutine pg$brk (cp)
long_int cp
```

Function

'Pg$brk' is used by the 'page' subroutine to catch the QUIT$ condition and return to a set place within it.

The user should not call this routine directly.

Implementation

'Pg$brk' simply calls 'pl1$nl' with the 'Rtlabel' array from the Software Tools common block. This was previously set to the proper label to return to.

Calls

Primos pl1$nl

See Also

page (2)

Calling Information

```
subroutine reonu$ (ptr)
pointer ptr
```

Library: vswtlb (standard Subsystem library)

Function

'Reonu$' is called from the Primos condition mechanism when the Primos internal command REN is given. 'Reonu$' returns to the caller who most recently declared 'reonu$' as its on-unit for the "REENTER$" condition.

Implementation

Using information passed by the condition mechanism, 'reonu$' finds the stack frame of the declarer of its on-unit and unwinds the stack with a call to the Primos routine PL1$NL.

Calls

Primos pl1$nl

See Also

sys$$ (2)

Calling Information

```
integer function rmfil$ (name)
packed_char name (MAXPACKEDFNAME)
```

Library: vswtlb (standard Subsystem library)

Function

'Rmfil$' is used to remove files, segment directories, and access categories by name. The sole argument is the name of a file (of either type, in the current directory) to be deleted. Note that the name is *not* an EOS-terminated string; it is a packed, blank-filled array of characters. The function return is OK if the deletion occurred, ERR otherwise.

Implementation

'Rmfil$' uses the Primos routine SRCH$$ to delete the file if possible. If this attempt fails because the file is a non-empty segment directory, it is opened and cleaned out by a call to 'rmseg$', then deleted by SRCH$$. If it fails because the file is an access category, it calls CAT$DL to remove it.

Calls

ptov, Primos cat$dl, Primos srch$$, rmseg$

See Also

remove (2), del (1), rmseg$ (6)

Calling Information

```
subroutine rmseg$ (funit)
integer funit
```

Library: vswtlb (standard Subsystem library)

Function

'Rmseg$' cleans out the segment directory open on the Primos file unit given as its sole argument.

Implementation

'Rmseg$' deletes successive entries in the directory using the Primos routine SRCH$$. When 'rmseg$' comes across an entry that is itself a non-empty segment directory, it opens that directory then calls itself recursively to clean it out. When all entries have been removed, the directory itself is set to zero length.

Calls

Primos sgdr$$, Primos srch$$, rmseg$

See Also

del (1), remove (2), rmfil$ (6)

Calling Information

subroutine rtn$$

Library: vswtlb (standard Subsystem library)

Function

'Rtn$$' unwinds the stack and returns to the routine (usually 'call$$') indicated by 'rtlabel' in the Subsystem common block.

Implementation

'Rtn$$' first checks the Primos command level data flags to see if the calling routine was DBG; if so, it immediately exits. If it was not called by DBG, 'rtn$$' returns to the routine indicated by 'rtlabel' via the Primos routine PL1$NL.

Calls

Primos pl1$nl

See Also

call$$ (6)

Calling Information

```
integer function sprot$ (name, attr)
character name (ARB), attr (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Sprot$' is used to set the protection attributes (read, write, truncate) for a given file. Both owner and nonowner attributes, specified in convenient notation, may be set on a single call.

The first argument is the full pathname of the file whose protection attributes are to be changed. The name must be in the usual Subsystem format in an EOS-terminated string.

The second argument is a protection attribute specification string. Each attribute (read, write, truncate) is represented by a single letter ("r", "w", and "t", respectively). In addition, the letter "a" indicates all attributes; it is equivalent to "twr". In the specification string, owner protection attributes appear first, followed by a slash and nonowner protection attributes. If no permissions are to be confered to nonowners, the slash may be omitted. If all attributes are omitted, neither owners nor nonowners may access the named file in any way. Examples: "a/r" confers all permissions for owners, read-only for nonowners; "rw/rw" confers read/write permission for everyone, "/w" confers write permission for nonowners and no permission for owners.

The function return is OK if the attempt to set protection attributes succeeded, ERR otherwise. The error condition is returned if (1) the 'attr' string contains an illegal protection key; (2) the named file could not be reached; (3) Primos returns an error code during the attempt to set the file's attributes.

Implementation

Protection keys in the form of two three-bit strings in the UFD entry for a file are maintained by the Primos routine SATR$$. 'Sprot$' first scans the protection attributes string, building a bit-string representation of the attributes as it goes. A call to 'getto' then sets the current directory to the parent of the named file. SATR$$ is then invoked to set the file's protection attributes. Finally, the Primos routine AT$HOM is used to attach back to the home directory.

Calls

getto, index, Primos at$hom, Primos satr$$

See Also

follow (2), getto (2), chat (1), lf (1)

Calling Information

```
integer function st$lu (symbol, node, pred, table)
character symbol (ARB)
pointer node, pred, table
```

Library: vswtlb (standard Subsystem library)

Function

'St$lu' attempts to find the character-string symbol given as its first argument in the symbol table given as its fourth argument. If the symbol is present, a pointer to its node is returned in 'node' and 'st$lu' returns YES; otherwise, 'st$lu' returns NO. In both cases, 'pred' is set to the address of the link field of the previous node, i.e. the one that points to the desired node (if it is present) or is at the end of the appropriate hash chain (if it is not present).

'St$lu' is not intended for general use; the symbol table interface routines 'enter', 'lookup', and 'delete' are provided for that purpose.

Implementation

'St$lu' hashes the character string by summing the internal representations of its characters and then reducing this number modulo the hash table size (a prime number). This hash is used as an index into a hash table containing heads of linked lists of symbol table nodes. The appropriate list is then searched linearly for a node containing the desired symbol text. The utility routine 'equal' is used for performing string comparisons.

If the symbol is found, then its index and its predecessor's link field index are returned, along with the function value YES. Otherwise, the address of the last link field in the appropriate chain is returned, along with the function value NO. The combination of node address/predecessor address is designed to make insertion, deletion, and updating of symbol table nodes relatively easy.

Arguments Modified

pred, node

Calls

equal

See Also

enter (2), lookup (2), delete (2), mktabl (2), rmtabl (2), dsget (2), dsfree (2), dsinit (2), sctabl (2)

Calling Information

```
longint function szfil$ (fd)
integer fd
```

Library: vswtlb (standard Subsystem library)

Function

'Szfil$' returns the size of the file (in words) associated with the open Primos file unit 'fd'. If there is some error on the file, the function return is ERR. The file is left positioned at the end of file.

Implementation

Primos PRWF$$ is called to relatively position the file to a large position until the file reaches end of file. Then PRWF$$ is called to read the current position of the file. If any error occurs, the function returns ERR, otherwise the size of the file is returned as the function return.

Arguments Modified

none

See Also

gfdata (2), sfdata (2), szseg$ (6)

Calling Information

```
subroutine szseg$ (size, fd)
longint size
integer fd
```

Library: vswtlb (standard Subsystem library)

Function

'Szseg$' returns the size of the segment directory open on the Primos file descriptor 'fd'. If an error occurs while sizing the directory, 'size' will contain ERR, otherwise it will contain the size in words of the directory.

Implementation

The directory is read and each of the file entries is checked. If an entry is a normal file, 'szfil$' is called to size it. If the entry is another segment directory, 'szseg$' is called recursively to size the subdirectory.

Arguments Modified

size

Calls

Primos sgdr$$, Primos srch$$, szfil$ (6)

See Also

gfdata (2), sfdata (2), szfil$ (6)

Calling Information

subroutine t$clup

Library: vswtlb (standard Subsystem library)

Function

A call to 't$clup' is inserted before each "call swt" in a Subsystem program that is to be profiled. When used with the "-p" option, Ratfor inserts this call automatically before a "stop" statement, and converts the "stop" to a "call swt".

The purpose of 't$clup' is to write to the file "_profile" a summary of the amount of real, cpu, and paging time spent in each subroutine of the profiled program. This summary is then read by the program 'profile' and formatted into a report.

Since no profiling information is written by any of the other profiling routines, 't$clup' must be called if a profile is to be made.

'T$clup' should be called explicitly only by those users wishing to profile Fortran programs by hand; Ratfor users should always profile with the "-p" option of the preprocessor.

Implementation

'T$clup' repeatedly calls 't$exit' until all subprogram calls have been cleaned up from the internal call stack. The file "_profile" is opened via a call to 'create' and filled by repeated calls to 'writef'. A final call to 'close' closes the file, leaving it ready for analysis by 'profile'.

Calls

cant, close, create, t$exit, writef, Primos at$hom

Bugs

This code should be invoked by 'swt', if necessary and possible.

See Also

t$entr (6), t$exit (6), t$time (6), t$trac (6), rp (1)

Calling Information

```
subroutine t$entr (routine)
integer routine
```

Library: vswtlb (standard Subsystem library)

Function

'T$entr' records the real, cpu, and paging times of the current process upon subprogram entry. This information is later modified by 't$exit' to reflect only the time spent in the particular subprogram, which is then added to the total for the subprogram.

'Routine' is the number of the subprogram being entered; subprograms are numbered consecutively beginning with 1 for the main program.

'T$entr' should be called explicitly only by those users profiling Fortran programs with hand-inserted code, in which case a call to 't$entr' should be the first executable statement of any profiled routine.

Implementation

A call to 't$time' gathers the necessary information, which is then stacked in a stack provided by the user (automatically, in the case of Ratfor programs).

Calls

Primos tnou, swt, t$time

Bugs

Stack overflow terminates the program.

See Also

t$exit (6), t$clup (6), t$time (6), t$trac (6), rp (1)

Calling Information

subroutine t$exit

Library: vswtlb (standard Subsystem library)

Function

'T$exit' is called from profiled programs just before a "return" statement is executed. It records the current amount of real, cpu, and paging time used, and determines from these the amount of time spent in the current subprogram. This information is added to the total time figures maintained for each subprogram.

Implementation

'T$time' is called to fetch the pertinent information, which is then subtracted from the values on the stack to obtain the time spent in the current routine. Adjustments are made to items remaining on the stack so that they do not reflect time spent in subordinate subprograms.

Calls

t$time

See Also

t$entr (6), t$clup (6), t$time (6), t$trac (6), rp (1)

Calling Information

```
subroutine t$init
```

Function

'T$init' is called at the beginning of the main program in Ratfor programs that have been processed with the "-p" (profiling) option of 'rp'. It initializes the profiling common blocks with the number of routines in the program.

'T$init' is inserted into the Fortran output as inline code, rather than being referenced from the standard Subsystem library. As such, it can never be accessed by the user unless the "-p" option is specified (even then, it should not be called by the user, since it has no effect on the profiling information).

Implementation

A simple assignment statement initializes a variable in the common blocks produced by 'rp' and used by the profiling subroutines.

See Also

rp (1) t$clup (6), t$entr (6), t$exit (6), t$time (6), t$trac (6)

Calling Information

```
subroutine t$time (reel, cpu, diskio)
long_int reel, cpu, diskio
```

Library: vswtlb (standard Subsystem library)

Function

'T$time' is called by 't$entr' and 't$exit' to fetch the amounts of real time, cpu time, and disk I/O time accumulated so far during this run.

Implementation

The Primos routine TIMDAT is called to fetch the information, which is converted uniformly to timer ticks.

Arguments Modified

reel, cpu, diskio

Calls

Primos timdat

Bugs

Timer resolution is not good.

See Also

t$entr (6), t$exit (6), t$clup (6), rp (1)

Calling Information

```
subroutine t$trac (mode, name)
integer mode
integer name (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'T$trac' is called from traced Ratfor programs (those processed with the "-t" option). Calls to 't$trac' are planted in the Fortran output of Ratfor as the first executable statement of each routine and before each "return" and "stop".

'Mode' is 1 for subprogram entry, 2 for subprogram exit, and 3 for initialization of the indentation level. 'Name' is a period-terminated packed string containing the name of the routine being traced. It need be supplied only when 'mode' has a value of 1 (subprogram entry).

'T$trac' produces an indented listing with vertical lines to help connect subprogram entry and exit. The trace is produced on ERROUT.

Implementation

A level counter is maintained to determine the amount of indentation; simple output statements produce the trace.

Calls

putch, print

See Also

t$entr (6), t$exit (6), t$clup (6), t$time (6), rp (1)

Calling Information

```
integer function tcook$ (ubuf, size, tbuf, tptr)
character ubuf (ARB), tbuf (MAXTERMBUF)
integer size, tptr
```

Library: vswtlb (standard Subsystem library)

Function

'Tcook$' reads and processes a line from the terminal. It handles the processing of escape sequences, case and character set mapping, line kills, character deletes, EOF's, and NEWLINE's. 'Ubuf' is the user's buffer that is to receive no more than 'size' characters (including the EOS). 'Tbuf' is a scratch buffer that is used to process the input before moving it to 'ubuf' and 'tptr' is the index of the current character being processed. Before the first call to 'tcook$', 'tptr' should contain a 1 and the first element of 'tbuf' should contain an EOS. If these variables are used in subsequent calls, they will be updated automatically to contain the correct values. The function return value is the number of characters returned in 'ubuf', not including the EOS.

Implementation

'Tcook$' reads input from the terminal one character at a time, interpreting each character as it is read. Special characters (the Subsystem escape character, kill character, retype character, newline character, and EOF character) are removed and the appropriate action taken while other characters are echoed and passed on directly. When the NEWLINE character is read or when the end-of-file is generated, reading terminates and the resulting line is returned after any required case and character set mapping are performed.

The several special characters used by 'tcook$' to provide extra functionality are explained below. Look at the description of the 'escape' character to see how to enter the special characters without their special properties being interpreted.

eof
: The eof character causes the generation of an end-of-file when read from the terminal. If there is information already entered on the current line, the user's kill response is printed, the information on that line is forgotten, and the end-of-file is generated.

erase
: The erase character causes the removal of one character of previously read input. The cursor is

backed up one character position. If there happened to be no characters on the line, nothing happens.

escape
The escape character is normally used to enter the Subsystem special characters and other characters with special attributes. If any character is preceded by the escape character (including the escape character) it will be passed without interpretation into the receiving buffer. If an escape character is followed by a '/' character and then another character, that character will have its parity bit (normally on) turned off. The final case is an escape followed by three octal (0 through 7) digits. The character formed by those three digits is the character that is entered.

kill
The kill character is used to discard all text on the current input line. When entered, the user's kill response is printed, any information on the current line is forgotten, and the user is allowed to retype the line.

newline
The newline character is a signal to the input routines that the user is finished with the current line of text and is ready for the machine to accept it. If the character is not a linefeed character (the standard newline character) then 'tcook$' will echo a carriage return linefeed combination and return a NEWLINE character at the end of the line.

retype
Occasionally, the user will have a message forced to his terminal, or will have typed input ahead of the system, while the system is generating output. In such a case, the representation of the current input line on the user's terminal will become disrupted. To see what has actually been entered, the user may enter the retype character, and 'tcook$' will echo the current input text. The user can then finish entering his commands.

Arguments Modified

ubuf, tbuf, tptr

Calls

Primos c1in, Primos duplx$, Primos tonl, Primos t1ou

See Also

User's Guide for the Software Tools Subsystem Command Interpreter, term (1), *Primos Subroutines Reference Guide* (*DOC 3621*)

Calling Information

```
integer function tgetl$ (buf, size, f)
character buf (ARB)
integer size
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Tgetl$' is the device-dependent driver for terminal line-image i/o. The first argument is a string to receive the line read; the second argument is the maximum number of characters to be placed in the string; the third argument is the file descriptor of a terminal file. The function return is either zero (when EOF is detected) or the length of the string read in.

Implementation

'Tgetl$' first checks to see if the terminal buffer is empty. If it is, then 'tcook$' is called to refill the buffer. The characters in the terminal buffer are copied to the user buffer 'buf' until either 'size' characters have been copied or the terminal buffer has been exhausted. The return value is the number of characters that were copied into 'buf'.

Arguments Modified

buf

Calls

tcook$

See Also

getlin (2), tputl$ (6), tcook$ (6)

Calling Information

```
file_mark function tmark$ (f)
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Tmark$' is used to obtain the current position of a terminal file. Since this concept doesn't have much meaning at the present, 'tmark$' always returns zero.

See Also

markf (2), seekf (2)

Calling Information

```
integer function tputl$ (line, fd)
character line (ARB)
integer fd
```

Library: vswtlb (standard Subsystem library)

Function

'Tputl$' is the device-dependent driver called by 'putlin' to write a string on a terminal file. The first argument is the string to be written; the second is the file descriptor of a terminal file. The function return is OK if the write was successful, ERR otherwise.

Implementation

'Tputl$' buffers each character in the string into a local buffer, and outputs the string in chunks (via calls to the Primos routine TNOUA). If case mapping is in effect, characters not on a KSR 33 keyboard are converted to escaped representations which are printable on a KSR 33.

Calls

Primos tnoua

See Also

tgetl$ (6), dputl$ (6), putlin (2)

Calling Information

```
integer function tread$ (buf, nw, f)
integer buf(ARB), nw
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Tread$' is the device-dependent driver for terminal line-image i/o. The first argument is an array to receive the words read; the second argument is the number of words to be read; the third argument is the file descriptor of the file from which data will be read. 'Tread$' returns the number of words placed in the receiving buffer if the read was successful, EOF otherwise. 'Tread$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'readf'.

Implementation

'Tread$' calls the Primos subroutine C1IN 'nw' times, or until a NEWLINE or EOF is encountered. C1IN gets the next character from the terminal or command input stream.

Arguments Modified

buf

Calls

Primos c1in

Bugs

The semantics of 'tread$' are a little shaky; since one character per word is stored in a terminal buffer, 'tread$' actually reads characters instead of raw words.

See Also

readf (2), dread$ (6)

Calling Information

```
integer function tscan$ (path, buf, clev, nlev, action)
character path (MAXPATH)
integer buf (MAXDIRENTRY), clev, nlev, action
```

Library: vswtlb (standard Subsystem library)

Function

'Tscan$' is used to traverse a subtree of the file system rooted at 'path'. Each time 'tscan$' is called, it returns the entry of the next file (or directory) in 'buf', as controlled by 'action' (discussed below). 'Clev' is the current level of descent into the subtree; it must be initialized to zero before calling 'tscan$'. 'Nlev' is the maximum level of descent into the subtree; 'clev' will never be greater than 'nlev'. The function return is OK if an entry was successfully fetched, ERR if the fetch was unsuccessful, EOF when the entire subtree has been scanned, or EOD if a directory has been completely scanned and the EODPAUSE action has been specified (see below).

The argument 'action' is a bit mask composed of the sum of several action codes. The codes POSTORDER and PREORDER control the visitation of directories. If POSTORDER is in effect, each directory entry will be returned after all the files in the directory have been visited. If PREORDER is in effect, each directory entry will be returned before any of the files in the directory have been visited. The first element of 'buf' is set to 0 or 1 to indicate a preorder or postorder encounter, respectively. Note that both PREORDER and POSTORDER may be specified; in this case, each directory in the subtree is visited twice. The code EODPAUSE causes 'tscan$' to return the code EOD when it completes the scan of a directory. Finally, the code REATTACH causes 'tscan$' to insure that the user is always attached to the directory currently being scanned. (Maintaining the attach point can be expensive, so it has been made optional.)

Implementation

Various pieces of state information are retained in the common block 'c$tscn'. 'Tscan$' changes state as it scans the subtree, deciding when to ascend, descend, pause, get the next entry, etc. The algorithm is a simple tree traversal, complicated mainly by the difficulties of error handling and maintaining the attach point.

Arguments Modified

buf, clev, path

Calls

at$swt, ctoc, error, expand, equal, follow, move$, upkfn$, Primos at$hom, Primos dir$rd, Primos gpas$$, Primos srch$$, Primos texto$

Bugs

No more than one instance of 'tscan$' may be active at a given time.

See Also

Primos dir$rd lf (1), del (1), chat (1), cp (1), follow (2)

Calling Information

```
integer function tseek$ (pos, f, ra)
longint pos
file_des f
integer ra
```

Library: vswtlb (standard Subsystem library)

Function

'Tseek$' is an internal Subsystem routine that performs the function of 'seekf' for terminal files only. The first argument is a long integer value which specifies the amount of positioning relative to the current position (negative and absolute positioning are not allowed on terminal devices). The second argument is the file descriptor of the file whose file pointer is being manipulated. The third argument must equal REL. The function return is OK if the positioning was successful, ERR if 'ra' is ABS or if 'pos' is negative. 'Tseek$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'seekf'.

Implementation

'Tseek$' calls the Primos subroutine C1IN 'pos' times to "skip over" 'pos' characters.

Calls

Primos c1in

See Also

seekf (2)

Calling Information

```
integer function ttyp$f (ttype)
character ttype (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Ttyp$f' looks in the file "=termlist=" to see if a terminal type is defined for the user's terminal line. 'Ttyp$f' returns a character string which is the name of the user's terminal type in 'ttype'. It returns YES if it could determine the terminal type, and NO otherwise.

Implementation

'Ttyp$f' tries to find the terminal name from the terminal list file in "=termlist=" (nominally "//extra/terms"); if it can find it, it returns the name found, sets the associated terminal attribute values in the Subsystem common area, and returns YES. If the terminal name could not be determined from the "=termlist=" file, it returns NO.

Arguments Modified

ttype

Calls

ctoi, close, date, getlin, open, ttyp$v

See Also

term_type (1), other ttyp$?* routines (6)

Calling Information

subroutine ttyp$l

Library: vswtlb (standard Subsystem library)

Function

'Ttyp$l' will list all of the terminal types that are supported by the Subsystem and its associated software packages (such as VTH).

Implementation

'Ttyp$l' opens the "=ttypes=" (nominally "//extra/terms") file, if it can, and lists the terminal types available in a readable format to the terminal.

Calls

close, input, length, open, print

Bugs

Some might consider it a bug that the output is always written to the terminal.

See Also

se (1), term (1), term_type (1), other ttyp$?* routines (6)

Calling Information

```
integer function ttyp$q (ttype, blankok)
character ttype (MAXTERMTYPE)
integer blankok
```

Library: vswtlb (standard Subsystem library)

Function

'Ttyp$q' asks the user for the name of his terminal. If an unknown terminal type is specified, the user is given the option of having the known terminal types listed by entering either a "?" or "help" or entering a valid terminal type. If a valid terminal type was given by the user, the function returns YES; otherwise, the function return value is NO. For valid user responses, 'ttype' contains the terminal type name.

Implementation

After a user response is entered, it is mapped to lower case (for consistency). If a null response is entered and is permitted by the caller (i.e., 'blankok' is YES), then all terminal type information in the Subsystem common area is erased; otherwise, the terminal type entered is validated. If it is a valid terminal type, the values of its attributes are set; otherwise, the user is asked to enter a correct response or a help request.

Arguments Modified

ttype

Calls

ctoc, equal, input, mapstr, print, ttyp$l, ttyp$v

See Also

se (1), term (1), term_type (1), other ttyp$?* routines (6)

Calling Information

```
integer function ttyp$r (ttype)
character ttype (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Ttyp$r' retrieves the name of the terminal from the Subsystem common area, if it has been previously set. If a valid name is found, the function returns YES; otherwise, it returns NO.

Implementation

'Ttyp$r' calls 'chkstr' to see if a valid terminal name has been set in the Subsystem common area. If an invalid terminal name has been set, it clears the name and returns NO; otherwise, it copies the name to 'ttype' and returns YES.

Arguments Modified

ttype

Calls

chkstr, ctoc

See Also

term (1), term_type (1), other ttyp$?* routines (6)

Calling Information

```
integer function ttyp$v (ttype)
character ttype (MAXTERMTYPE)
```

Library: vswtlb (standard Subsystem library)

Function

'Ttyp$v' takes the terminal type name given in 'ttype' and tries to set the values of that terminal's attributes in the Subsystem common area. If the terminal type is not a legal one, then 'ttyp$v' returns NO and leaves the values of the terminal attributes undisturbed; otherwise, it sets up the appropriate values and returns YES.

Implementation

'Ttyp$v' takes the terminal type given in 'ttype' and tries to find it in the file "=ttypes=" (nominally "//extra/terms"). If the file could not be opened or if the terminal type wasn't found in the file, 'ttyp$v' returns NO; otherwise, it sets the value of the terminal's attributes from values given in the file and returns a value of YES.

Calls

close, ctoc, equal, input, open, Primos break$

See Also

other ttyp$?* routines (2)

Calling Information

```
integer function twrit$ (buf, nw, f)
integer buf (ARB), nw
file_des f
```

Library: vswtlb (standard Subsystem library)

Function

'Twrit$' is the device-dependent driver for terminal line-image i/o. The first argument is a string of words to be written; the second argument is the number of words to be written; and, the third argument is the file descriptor of the file to which data will be written. 'Twrit$' returns the number of words written (which should always equal 'nw'). 'Twrit$' is not intended for general use; it is not protected from user error, and may cause termination of the user's program if used incorrectly. It should always be referenced through 'writef'.

Implementation

'Twrit$' calls the Primos subroutine T1OU 'nw' times, once per word in 'buf'. T1OU outputs one character to the user terminal.

Calls

Primos t1ou

See Also

writef (2), tread$ (6), readf (2)

Calling Information

```
integer function upkfn$ (name, len, str, max)
packed_char name (ARB)
integer len, max
character str (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Upkfn$' operates on the packed Primos file name or password of length 'len' in 'name'. It converts it to an EOS-terminated string in 'str' by unpacking it and placing the escape character ("@") in front of all slashes. Thus, the name in 'str' is acceptable to all Subsystem routines expecting a file or path name.

The function value is the number of characters placed in 'str'. In no case will more than 'max' elements of 'str' be disturbed.

Implementation

'Upkfn$' uses the Subsystem macro 'fpchar' to take successive characters from the packed name. Each character is copied into the receiving string (preceded by an escape character, if it is a slash) after being mapped to lower case until the string is full or the end of the name is reached.

Arguments Modified

str

Calls

mapdn

See Also

follow (2), getto (2), open (2), ptoc (2)

Calling Information

```
integer function vt$alc (tbl, c)
integer tbl
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$alc' is used to allocate another VTH definition table for the key sequence definitions.

Implementation

'Vt$alc' verifies that there is room for another definition table and then initializes the new table if there is room. The "next table" pointer in the table 'tbl' is set to indicate the index to the new table. If no room is found, then ERR is returned.

Arguments Modified

tbl

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$cel (dummy)
integer dummy
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$cel' is used to clear the line from where the cursor is currently positioned to the end of the line. The return value is OK if the line was cleared, and ERR otherwise.

Implementation

The VTH common block is checked for the existence of a clear to eol sequence. If one exists, it is written out and the function return is OK. If there is no sequence, the function return is ERR.

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$clr (dummy)
integer dummy
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$clr' is used to clear the screen on the user's terminal. The return value is OK if the screen was cleared, and ERR otherwise.

Implementation

The VTH common block is checked for the existence of a clear screen sequence. If one exists, it is written out, 'vt$del' is called to print out a small delay sequence, and the function return is OK. If no sequence exists, the return is ERR.

Calls

vt$del

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

subroutine vt$db

Library: vswtlb (standard Subsystem library)

Function

'Vt$db' is used to print out the values of terminal characteristics in the VTH common block. 'Vtinit' should have been called beforehand to set up these terminal characteristics.

Implementation

'Vt$db1' is called to print the mnemonics for the cursor movement control sequences. Then the numerical terminal characteristics (such as cursor movement delay time and screen dimensions) are output by calls to 'print'. All output is to ERROUT.

Calls

ctomn, print, vt$db1

Bugs

Not meant to be called by the normal user.

Used mainly for debugging of the VTH package.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$db1 (title, seq)
character title (ARB), seq (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$db1' is used to print out a label in 'title', along with the mnemonics for the special (unprintable) character sequence in 'seq'.

Implementation

'Print' is called to output 'title'; then each character in 'seq', up to the first EOS, is converted to its associated mnemonic and printed out. All output is written to ERROUT.

Calls

ctomn, print

Bugs

Not to be called by the normal user.

Mainly used for debugging of the VTH package.

See Also

other vt?* routines (2) and (6)

Calling Information

subroutine vt$db2

Library: vswtlb (standard Subsystem library)

Function

'Vt$db2' dumps out the key sequence definitions from the terminal input tables in semi-formatted form.

Implementation

For each key sequence definition table, 'vt$db2' first checks to see if the table is being used. If so, it prints out all the entries in the table, with the format being based on what the information type is. The types of information stored include the pointer to the next definition table, a pointer to a definition sequence, or character values. Unprintable character values are converted to mnemonics before being output.

Calls

ctomn, print

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

subroutine vt$db3

Library: vswtlb (standard Subsystem library)

Function

'Vt$db3' prints the actual definition sequences for the keyboard macros.

Implementation

After printing out a heading, 'vt$db3' prints out each character in the macro definition sequence, mapping unprintable characters to their corresponding mnemonic.

Calls

ctomn, print

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$def (dummy)
integer dummy
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$def' is used to accept a macro definition from the user. If a status line is enabled, the user is prompted, otherwise he must remember the entry format himself. The format is

<del><seq1><del><seq2><del>

where <del> is a delimiter not used in either sequence, <seq1> is the macro, and <seq2> is the substitution string. When the macro sequence is entered, it is immediately replaced by the substitution string. If there is no room for another definition, no room for another substitution sequence, or an illegal sequence is entered, the function return is ERR and 'vt$err' is called to print an an appropriate message, otherwise the function return is OK.

Calls

vtalc, vterr, vtgsq, vtrdf, vtmsg, vtupd, and Primos c1in

Bugs

Not meant to be called by the normal user.

See Also

Primos t1in and other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$del (delay)
integer delay
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$del' is used to delay the terminal for 'delay' milliseconds after certain operations.

Implementation

The VTH common block is checked for the current baud rate of the terminal. 'Delay' is then used to calculate the number of nulls to write to the terminal.

Calls

Primos t1ou

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$dln (dummy)
integer dummy
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$dln' is used to delete a line at the current cursor position on the user's screen. The return value is OK if the line was deleted and ERR otherwise.

Implementation

The subsystem common block is checked for the existence of a delete line sequence. If one exists, it is written out and 'vt$del' is called to print out a small delay sequence. If the sequence existed, the function returns OK and if it did not exist, the function returns ERR.

Calls

vt$del

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

subroutine vt$dsw

Library: vswtlb (standard Subsystem library)

Function

'Vt$dsw' reclaims the space occupied by unused definition tables for use in storing other definitions.

Implementation

'Vt$dsw' looks for tables whose entries have all been undefined; their "in use" indicators are reset, and all references to them by other tables are removed.

Calls

vtprt

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$err (msg)
character msg (ARB)
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$err' prints the specified message, 'msg', in the status line (if one exists), and resets the VTH pushback buffer to 0.

Implementation

A call to 'vtmsg' is made to print the message on the status line (if one is enabled). Then the pushback pointer in the VTH common block is set to 0 and a BEL character is printed.

Calls

vtmsg, vtupd, Primos t1ou

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$get (row, col, start, len)
integer row, col, start, len
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$get' is responsible for reading characters from the terminal and interpreting the special characters. The first two arguments are the 'row' and 'column' at which to start accepting input. The third argument is the start of the input area on the current row, and the fourth argument is the length of the input area. 'Vt$get' will continue reading from the terminal until a line termination character is input (RETURN, KILL_RIGHT_AND_RETURN, MOVE_UP, or MOVE_DOWN). The function return is the termination code. Any macros are expanded by a call to 'vt$idf'.

Calls

vtdef, vterr, vtidf, vtndf, vtout, vtput, vtmove, vtmsg, vtupd, Primos c1in

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6), se (1), and the *Introduction to the Software Tools Text Editor* (Se section) for a list of available special characters.

Calling Information

```
integer function vt$gsq (msg, delim, seq, max)
character msg (ARB), delim, seq (ARB)
integer max
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$gsq' is used to read a sequence of characters from the users terminal. The first argument is a message which will be displayed in the status line, if one is enabled. The second argument is the delimiter used to terminate the sequence. The third argument is the returned sequence, and the last argument is the maximum length of the sequence. 'Vt$err' is called to print error messages for empty sequences, or for sequences which are too long. The function return is ERR if an illegal sequence is entered, or the length of the returned sequence.

Arguments Modified

seq

Calls

ctomn, encode, vt$err, vtmsg, vtupd, Primos c1in

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$idf (c)
character c
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$idf' is invoked to expand the definition of a keyboard macro which is encountered in user input; the definition is pushed back into the input stream.

Implementation

'Vt$idf' first checks for infinite recursive definition expansion. If it detects too high a nesting level, it returns ERR; otherwise, it locates the definition sequence, and copies it into the input pushback buffer. If the definition exceeds the capacity of the pushback buffer, ERR is returned; otherwise, OK is returned.

Calls

vt$err

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$ier (msg, name, line, fd)
character msg (ARB), name (ARB), line (ARB)
file_des fd
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$ier' is used to report an error in the contents of the terminal characteristics file (=vth=/?*). The file name 'name', a message 'msg' explaining the error, and the line 'line' from the file which caused the error are printed to ERROUT.

Implementation

'Vt$ier' calls 'print' to output the file name, the error message, and the erroneous line from the file to ERROUT. The VTH initialization file indicated by 'fd' is closed, and the function returns ERR.

Calls

close, print

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$iln (dummy)
integer dummy
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$iln' is used to insert a line at the current cursor position on the user's screen. The return value is OK if the line was inserted and ERR otherwise.

Implementation

The subsystem common block is checked for the existence of an insert line sequence. If one exists, it is written out and 'vt$del' is called to print out a small delay sequence. If the sequence existed, the function returns OK and if it did not exist, the function returns ERR.

Calls

vt$del

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$ndf (ch)
character ch
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$ndf' removes keyboard macro definitions from the appropriate tables.

Implementation

'Vt$ndf' prompts the user for the sequence which is to be removed from the definition tables. If the sequence is found, then its definition is removed and its associated definition tables are garbage collected; otherwise, ERR is returned.

Calls

vtmsg, vtupd, vtdsw, vterr, vtgsq, vtrdf, Primos c1in

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$out (ch)
character ch
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$out' is the very low level routine which does the actual character output to the terminal screen; the character contained in 'ch' is printed on the screen after certain considerations are evaluated.

Implementation

First, 'vt$out' checks to see if the character would be output on the last character position of the last line of the screen; if so, it returns without doing the output operations (thus preventing the screen from scrolling). Next, the character is checked to see if it is printable; if not, then a printed representation is output (if a "shifted" sequence for the unprintable character is defined, i.e. a transparent mode indicator for the terminal, then that sequence is output before the character itself). The internal screen cursor position indicators are updated to reflect that a character was printed.

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
integer function vt$pos (row, col, crow, ccol)
integer row, col, crow, ccol
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$pos' positions the cursor on the terminal screen to 'row', 'col' from 'Crow', 'ccol'. If the positioning can be done faster relatively, a relative position is output, otherwise the positioning is done absolutely. 'Vtinit' or 'vtterm' should have been called beforehand to set up the terminal characteristics in the virtual terminal handler. If the positioning can be done, 'vt$pos' returns OK. If the positioning can't be done, or the row and column are out of the terminal's screen boundary, ERR is returned.

Implementation

'Vt$pos' after checking to make sure the coordinates given are actually on the terminal's screen, computes a 'row-coordinate' and a 'column-coordinate' that are output after the lead-in absolute cursor positioning sequence for the terminal. There are only a few different standard ways to compute this character. Based on how the terminal does absolute addressing, 'vt$pos' then outputs the characters in the correct sequence to do the positioning. A small delay (usually nulls) is output for terminals that need it. Interested users should look at the code for more information.

Calls

print, vt$del

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$put (str, row, col, len)
character str (ARB)
integer row, col, len
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$put' takes the string given in 'str' and copies it to the screen buffers so that when the screen is next updated, the string appears starting at row 'row' and column 'col'. 'Len' indicates how long the string is.

Implementation

'Vt$put' first verifies that a legal location on the screen is given by the coordinates ('row', 'col'); if they are off the screen, then internal buffer variables are set to defaults which will prevent strange updating of the screen. Otherwise, the line is "fitted" to the screen; as much of it as possible will be displayed without overstepping the screen boundaries. The string in 'str' is then packed into the screen buffer, ready for the next screen update to occur.

Calls

print

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$rdf (c, tbl)
character c
integer tbl
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$rdf' removes a keyboard macro definition sequence from the definition tables.

Implementation

'Vt$rdf' locates the definition sequence in the definition tables and "packs" the table to remove the definition.

Calls

length

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine vt$rel (row, col, crow, ccol)
integer row, col, crow, ccol
```

Library: vswtlb (standard Subsystem library)

Function

'Vt$rel' positions the cursor on the terminal screen to 'row', 'col' from position 'crow', 'ccol' using only relative cursor positioning. 'Vtinit' or 'vtterm' should have been called previously to set up the terminal characteristics. 'Vt$rel' is called as a last resort to position the cursor by 'vt$pos'. If it is impossible to position the cursor with what knowlege it has, 'vtterm' will have already returned an error.

Implementation

'Vt$rel' uses whatever capabilities are available to position the cursor. If the terminal lacks a cursor_up sequence, the cursor is homed to the upper left hand side of the screen and moved down using the cursor_down sequence, or issuing LF characters (which is relatively universal). It moves the cursor to the right my issuing the cursor_right sequence and to the left by issuing the cursor_left sequence, or by issuing a CR character and issuing the cursor_right sequence.

Bugs

Not meant to be called by the normal user.

See Also

other vt?* routines (2) and (6)

Calling Information

```
subroutine zmem$ (node)
integer node (5)
```

Library: vswtlb (standard Subsystem library)

Function

"Zmem$" zeroes a block of memory in segment "node(2)" starting at word "node(4)" through "node(5)".

Implementation

Trivial.

See Also

ldseg$ (6)