Command Reference Guide

Revision 6.2.9

Information in this document is subject to change without notice and does not represent a commitment on the part of Dynamic Concepts, Inc. (DCI). Every attempt was made to present this document in a complete and accurate form. DCI shall not be responsible for any damages (including, but not limited to consequential) caused by the use of or reliance upon the product(s) described herein.

The software described in this document is furnished under a license agreement or nondisclosure agreement. The purchaser may use and/or copy the software only in accordance with the terms of the agreement. No part of this manual may be reproduced in any way, shape or form, for any purpose, without the express written consent of DCI.

Dynamic Concepts Inc.
18-B Journey
Aliso Viejo, CA 92656
www.dynamic.com

Chapter 1 - Introduction

This version (6.2.9) of the dL4 Command Reference Guide is based on Version 6.2.9 of the dL4 product and covers all future releases, except for any new enhancements.

The dL4 command set consists of:

- SCOPE System Command Processor commands
- Business BASIC commands
- Debugger Commands
- loadsave command
- run command
- utilities

These commands are described in this guide.

Intended Audience

The guide is designed to aid dL4 programmers with all levels of Business BASIC experience.

About This Guide

This guide is divided to describe the main components of dL4.

Chapter 2, “SCOPE Commands”, describes the System Command Processor and provides a detailed listing of all its commands.

Chapter 3, “BASIC Commands”, describes dL4 Business BASIC and provides a detailed listing of all BASIC commands.

Chapter 4, “Debugger Commands”, describes the Debugger and provides a detailed listing of all Debugger commands.

Chapter 5, “loadsave”, describes the loadsave command.

Chapter 6, “run” - describes the run command.

Chapter 7, “tools” – describes the utility programs

Appendix A, “Glossary - defines terms in dL4 context.

Appendix B, “dL4 Command Summary” - presents dL4 commands in tabular form.

Appendix C, “Position Parameter” - describes the position parameter.

Related Publications

The planned dL4 document set – now in development and subject to revision -- consists of:

1. *Introduction to dL4 Guide:* The first document for dL4 users. Describes entire product in general terms, defines key terms (e.g., Unicode, class), maps out other dL4 documents.

2. *dL4 Command Reference Guide*: The anchor document of the documentation set. Describes loadsave and run. Includes all SCOPE commands, including the Editor and Debugger.
3. *dL4 Migration Guide*: Compares and contrasts UniBasic and dL4. Designed to assist with migration of UniBasic programs to dL4.
4. *dL4 Files and Devices Reference Guide*: Introduces the concept of driver classes and describes the classes in detail. Designed to help the programmer use and benefit from driver classes.
5. *dL4 Language Reference Guide*: Describes dL4 statements in detail. Also describes language elements such as mnemonics, functions, etc.
6. *dL4 Installation & Configuration Guide Windows*: A platform-specific description of how to configure terminals, printers, etc.
7. *dL4 Installation & Configuration Guide Unix*: A platform-specific description of how to configure terminals, printers, etc.

Conventions

This guide follows these conventions:

EXAMPLE OF CONVENTION	DESCRIPTION
bold type	Literal elements of command
SAVE *filename string*	Metalinguistic variables are shown in *italic* type for clarity and to distinguish them from elements of the language itself.
`LIST`	Mono-spaced type is used to display screen output and example input commands and program examples.
`LIST {-v}`	The right and left brace characters ({*optional items*}) indicate an item that is optional
KILL filename {filename...}	A series of three periods (...) indicates that the preceding item can be repeated as many times as desired. Be careful not to confuse the three periods with the Period or Double Period commands.
WINDOW (**ON** | **OFF**)	Selection of one of a group of items is shown within parenthesis separated by |. Choose only one; WINDOW ON or WINDOW off. The parenthesis are not part of the syntactical form.

Chapter 2 - SCOPE Commands

The System Command Processor, or SCOPE, is a program that allows the system to understand your commands. It provides dL4 developers with a Command Line-oriented Integrated Development Environment (IDE). SCOPE:

- is a Command Interpreter
- acts interactively with user
- provides access to BASIC
- provides access to the Debugger through BASIC

The SCOPE Command Line IDE consists of three (3) command environments: SCOPE, BASIC, and Debugger. The SCOPE, BASIC, and Debugger commands are described in Chapters 2, 3, and 4, respectively.

The order in which SCOPE processes a command is as follows:

1. If the command line begins with an exclamation point (!), SCOPE simply passes the command, without the exclamation point, to the operating system.
2. If the command line does not begin with an exclamation point, SCOPE checks for an internal command. If an internal command is found, SCOPE executes that command.
3. If an internal command is not found, SCOPE looks for a dL4 BASIC program of said name. If a dL4 BASIC program is found, SCOPE loads and executes it.
4. If a dL4 BASIC command is not found, SCOPE passes the command, without the exclamation point, to the operating system.

If SCOPE is started with a "-noshell" command line option, then native operating system commands, with or without an exclamation point, are not executed and are treated as dL4 program names or internal commands.

SCOPE supports a command history feature which allows the user to select, edit, and execute previously typed commands within the current session. Typing an up arrow key or a down arrow key at any SCOPE prompt causes SCOPE to move up or down in the command history and to display the selected command for immediate use or editing.

This chapter describes the SCOPE commands in detail. Below, the table lists and briefly describes the SCOPE commands.

COMMAND	DESCRIPTION
! (Exclamation Point)	Execute an operating system command.
BASIC	Enter BASIC mode.
BYE	Terminate dL4 **SCOPE** session.
CD	Change current working directory.
CLU	Change current working directory.
DRIVERS	Display a list of all available drivers.
EXEC	Execute contents of a text file.
HALT	Terminate BASIC program on another port.
KILL	Delete a data or program file.
LEVEL	Display dL4 revision number and license number
OEM	Lists the currently authorized OSNs (OEM Security Number).
PACK	Change current working directory.
PSAVE	Create an OSN protected program.
RUN	Execute a program in memory or on disk.
SAVE	Save the current program.
TIME	Display current system time and usage.
USERS	Display current number of ports in use.

! (Exclamation Point)

Synopsis

Execute external operating system command

Syntax

!*COMMAND*

Parameters

command is any operating system (or null) command to be executed by a sub-shell.

Remarks

The Exclamation Point command is used to execute an external operating system command.

All system commands are executed by a separate child process, effectively putting dL4 to sleep until the command terminates. Changes to environmental variables and current working directory within a child processes are effective only during that process. Upon termination of the command, the parent (dL4) resumes execution unaware of the child's activities.

Examples

```
!ls -l
!vi query.bas
!edit query.bas
```

See also

CD, operating system documentation.

BASIC

Synopsis

Enter BASIC Mode.

Syntax

#BASIC {*filename*}

Parameters

filename is any *filename* or *pathname* to a dL4 program file (not text file) to which you have read-permission.

Remarks

If you don't specify a program name, any previous dL4 program remains.

If the *filename* that you enter is a saved dL4 program file, any current program is cleared and memory is loaded with the new program.

Only dL4 saved program files may be read with this command.

An error occurs if the filename cannot be loaded.

Examples

```
#BASIC 23/FILENAME
#BASIC /usr/progs/ar/payments
#BASIC
```

See also

LOAD, SAVE, NEW

BYE

Synopsis

Terminate dL4 SCOPE session and exit to operating system prompt.

Syntax

#BYE

Parameters

None.

Remarks

BYE is used to perform all the following functions:

1. clear any program from memory
2. close all channels
3. delete any remaining signals
4. reset any special terminal settings
5. terminate the current session.

Examples

#BYE

See also

CD

Synopsis

Change current working directory.

Syntax

#CD { *pathname* }

Parameters

pathname is any directory name or full path_name.

Remarks

CD is used to change the current working directory. In the absence of a pathname, then the current working directory is displayed when SCOPE is initialized, i.e., the startup directory is selected.

CD is functionally equivalent to the **CLU** and **PACK** commands.

Examples

```
#CD 23
#CD /usr/progs/ar/checks
#CD data
#CD ″test data″
```

See also

PACK, CLU

CLU

Synopsis

Change current working directory.

Syntax

#CLU { *pathname* }

Parameters

pathname is any directory path to an existing directory.

Remarks

CLU is used to change the current working directory.

In the absence of a pathname, then the current working directory is displayed when SCOPE is initialized, i.e., the startup directory is selected.

CLU is functionally equivalent to the **CD** and **PACK** commands.

Examples

```
#CLU 23
#CLU /usr/progs/ar/cash
```

See also

CD, PACK

DRIVERS

Synopsis

Display a list of available drivers.

Syntax

#**DRIVERS** { *searchtext* }

Parameters

searchtext is optional string to search. The search is case-insensitive. If *searchtext* is specified, then only those drivers whose names or class names contain the string *searchtext* will be displayed. If *searchtext* is omitted, all drivers will be displayed.

Remarks

DRIVERS is used to display a list of available channel drivers.

For information on the supported driver classes, refer to the dL4 Files and Devices reference manual. For platform specific drivers, refer to the dL4 platform guide for the specific operating system.

Examples

```
#DRIVERS
      Class                 Drivers

      Text                  Text
      Formatted             Portable Formatted
      Contiguous            Portable Contiguous
      Indexed-Contiguous    Portable Indexed Contiguous
      Full-ISAM             FoxPro Full-ISAM
```

See also

EXEC

Synopsis

Execute the contents of a text file.

Syntax

#EXEC *filename*

Parameters

filename is any legal filename or pathname to a text file to which you have read permission.

Remarks

EXEC is used to read a file and interpret it one line at a time. The contents of the file can be a list of BASIC and system commands which are interpreted individually. Input is switched to the text file performing all commands within the file until **EOF** (End of File).

EXEC may be used to automatically load and dump BASIC programs, or perform any series of commands as if they were entered at the keyboard.

Examples

```
#EXEC filename
```

```
#EXEC fcw
```

To load and dump a program, first create a text file containing the following lines:

```
basic programname
dump programname.bas!
exit
```

Then enter the following command line at the SCOPE prompt:

```
exec <name of text file>
```

To load and save a group of programs, first create a text file containing the following lines:

```
basic
new
load prog1.bas
```

```
save prog1.run!
new
load prog2.bas
save prog2.run!
exit
```

Then enter the following command line at the SCOPE prompt:

```
exec <name of text file>
```

See also

loadsave

HALT

Synopsis

Terminate BASIC program on another port.

Syntax

#HALT *port number*

Parameters

port number is any integer in the range from 1 to the upper limit defined.

Remarks

HALT is used to terminate a BASIC program on another port by causing an Abort event to occur on that selected port. An Abort event terminates execution of any running BASIC program.

Ports can only be **HALT**ed one port at a time. Trying to **HALT** more than one at a time causes an error. If no program is running, **HALT** is ignored.

Examples

```
#HALT 25
```

See also

KILL

Synopsis

Delete a data or program file.

Syntax

#KILL *filename.expr ...*

Parameters

filename.expr is the data or program file name to be deleted**.**

Remarks

KILL is used to delete a file. **KILL** displays “DELETED!” when you delete a single file, and “ALL DELETED!!” when you delete multiple files.

The *filename..expr* must contain a single *filename* or list of *filenames* to be deleted. Multiple strings may each contain a single filename or a group of filenames separated by spaces. This command deletes both the indexed and the contiguous files if an indexed-contiguous file is killed.

If an error occurs, the statement is aborted and any remaining filenames within the *str.expr* are not deleted. Furthermore, other *filename.exprs* are not processed.

Examples

```
#KILL file1
#KILL file1 file2
```

See also

LEVEL

Synopsis

Display dL4 revision and license numbers.

Syntax

#LEVEL

Parameters

None.

Remarks

LEVEL is used to display the dL4 revision number and/or the license number.

Examples

```
#LEVEL
```

OEM

Synopsis

List the currently authorized OSNs (OEM Security Number).

Syntax

#OEM {TEMP*}*

Parameters

TEMP causes the command to prompt for a temporary OSN (OEM Security Number)**.**

Remarks

The **OEM** command lists the currently authorized OSNs. If the TEMP option is used ("OEM TEMP"), the **OEM** command will prompt for a temporary OSN to be used only by the current SCOPE session.

Examples

```
#OEM
#OEM TEMP
```

See also

LOADSAVE, PSAVE, SAVE

PACK

Synopsis

Change current working Directory.

Syntax

#PACK { *pathname* }

Parameters

pathname is any directory path to an existing directory.

Remarks

PACK is used to display or change a current working directory.

If no *pathname* is specified, the current working *pathname* is displayed.

If a valid *pathname* is specified, the current working directory is changed to that *pathname*.

Examples

```
#PACK 23
#PACK /usr/ub/text
```

See also

CD, CLU

PSAVE

Synopsis

Create an OSN (OEM Security Number) protected program.

Syntax

#PSAVE *n*, *filename.expr* ...

Parameters

filename.expr is the program file name to be created**.**

n is the number of the OSN listed by the OEM command

Remarks

The **PSAVE** command is used to create OSN protected programs. The **PSAVE** command is identical to the **SAVE** command except for an optional OSN number that can precede the **SAVE** filename. For example, the command "PSAVE 2,menu" would save the current program as "menu" after protecting it to require the second OSN listed by the OEM command. Protected programs can be created only if the specified OSN is a master OSN.

Examples

```
#PSAVE 4, "file1"
```

See also

LOADSAVE, OEM, SAVE

RUN

Synopsis

Execute a program in memory or on disk.

Syntax

#**RUN** { *filename* }

Parameters

filename is any legal saved dL4 program file to which you have read-permission. An absolute or relative pathname can be specified with the file name.

Remarks

RUN is used to interpret a previously-loaded program. The program could also have been loaded or entered from text in Basic mode.

If no filename is supplied, the current program in memory, if any, is executed. If a filename is supplied, any current program in memory is erased.

Examples

```
#RUN FILENAME
RUN /tmp/FILENAME
#RUN
```

See also

SAVE

Synopsis

SAVE the current program.

Syntax

SAVE { -l *n* }{ <*attributes*> } { *filename*{!} } }

Parameters

The -l *n* option is used to create an OSN (OEM Security Number) protected program. The value "*n*" is the number of a master OSN as listed by the SCOPE **OEM** command.

<*attributes*> are any optional valid file *attributes, protections*, or *permissions* to apply to the file on creation. Standard IRIS, BITS, or UNIX-style permissions may be supplied. If omitted, file creation is defaulted to Read/Write for all users, subject to any operating system masking in effect. If <*attributes*> are supplied, a *filename* must follow.

filename is any optional *filename* or full *pathname* to a directory to which you have write-permission. A filename is optional if the file has previously been saved. If it has not been saved, then the filename is mandatory in the command. If the filename is omitted, the original *filename* for the program in memory is used.

Remarks

Prior to saving a program, it compiles the program which may result in errors. A compilation error still allows a program to be saved.

The “run-only” option, Save -RO, is used to produce “run-only” program files. Save -RO creates a run-only file which strips symbols so that a file cannot be listed, dumped, debugged, or modified in any way. The creation process is irreversible and secure since the information needed to list the program is actually discarded. A Save -RO file enables the developer to:

- Protect sensitive or trade-secret source code from theft or modification
- Safely embed security checks in a program, allowing it to run only on authorized systems.

When **SAVE** is performed from *program mode*, active channels and variables are undisturbed.

Examples

```
SAVE <22> prog!
SAVE <PWD> dat!
SAVE
```

See also

CHECK, OEM, SAVE

TIME

Synopsis

Display current system time and usage.

Syntax

#**TIME**

Parameters

None.

Remarks

The current system time is displayed in the form:

- *Mon* is a three-letter month name, such as JAN.
- *DD* is the current day of the month
- *Year* is the current year such as 1993.
- *HH* is the current hours in 24-hour format.
- *MM* is the current minute of the hour.
- *SS* is the current second on the minute.
- *NN* is the current hundredth on the second.
- *CPU* is the amount of seconds used by the computer for all of your commands and program execution.
- *Connect* is the number of minutes you have been signed on to the system.

Examples

```
#TIME
```

See also

USERS

Synopsis

Display current number of ports in use.

Syntax

#**USERS**

Parameters

None.

Remarks

USERS is used to determine the current number of dL4 ports in use. The local system is searched for all *port numbers* currently in use. The total number of users is then displayed on the terminal. "Ports in use" is synonymous with "number of users".

In-use *port numbers* include secondary Windows sessions, multi-screens, *phantom ports*, terminals and jobs initiated by **SPAWN**.

Examples

#USERS

See also

Chapter 3 - BASIC Commands

The BASIC commands are used to perform specific dL4 operations, such as **DUMP**, **LOAD**, and **SAVE**. BASIC commands are user-driven and utilize syntax to define these operations.

If a command is not recognized , then BASIC attempts to immediately execute it as a BASIC statement. Even a syntactically correct BASIC statement will not be executed if the current program contains an error as reported by the **CHECK** command. Refer to dL4 Language Reference Guide for information about statements that are executed immediately.

This chapter describes the BASIC commands in detail. The table below lists and briefly describes the commands.

COMMAND	DESCRIPTION
! (Exclamation Point)	Execute external operating system command
. (Dot)	Execute the next n program lines
.. (Double Dot)	Execute next program line and step through function
AUTO	Automatically enter program line numbers
BREAK	Create a breakpoint
CANCEL	Clear all variables and other runtime context
CHECK	Scan program for proper structure and linkage
CONTINUE	Resume execution of stopped program
CONVERT	Convert UniBasic statements from a text file
DELETE	Delete program statements
DISPLAY	Display the values of specified variables of the current running program.
DUMP	List a program to a text data file
EDIT	Edit and change an existing statement.
EXAMINE	Select procedure, function, library, or call stack level.
EXIT	Exit BASIC environment to SCOPE environment.
FILE	Display current program name and open all channels.
FIND	Search and list selected program statements.
GO	Resume execution of stopped program.
HELP	Print text description of an error.
LABEL	Convert statement numbers to labels.
LIST	Decode and list BASIC statements.
LOAD	Load BASIC statements from a text file.
NEW	Clear memory for new program.
NOBREAK	Delete a breakpoint
OEM	Lists the currently authorized OSNs (OEM Security Number).
PDUMP	List current program status, variables, and other information to a text data file.
PSAVE	Create an OSN protected program.
RENUMBER	Renumber statements in a program.
RUN	Execute a program up to specified line, then enter the Debugger.
SAVE	Save the current program.
SHOW	Search and list selected program statements.
SIZE	Display memory usage for current program/data.
STATUS	Print the name of the current program file and execution status.
TRACE	Enable statement trace debugging.
VARIABLE	Display variable names in current procedure
XBREAK	Create an external breakpoint

! (Exclamation Point)

Synopsis

Execute external operating system command

Syntax

! *command*

Parameters

command is any operating system (or null) command to be executed by a sub-shell.

Remarks

The Exclamation Point command is used to execute an external operating system command.

All system commands are executed by a separate child process, effectively putting dL4 to sleep until the command terminates. Changes to environmental variables and current working directory within a child processes are effective only during that process. Upon termination of the command, the parent (dL4) resumes execution unaware of the child's activities.

Examples

```
!ls -l
!vi query.bas
!edit query.bas
```

See also

CD, operating system documentation.

. (Dot)

Synopsis

Continue execution for one or more program lines.

Syntax

. {*n*}

Parameters

n is an optional integer used to specify the number of lines to step through the program. If n is omitted, 1 is assumed.

Remarks

Single statement execution is performed by entering a . and pressing Return. The current statement is executed and the next statement to execute is displayed. Subsequent dots are used to step through the program. To resume normal execution of a program, issue the command **CONTINUE**. For applications relying on **CALL**ed subprograms, single statement execution can be performed for both *stepping through* a subprogram by entering a . and pressing return.

The "." form executes subprograms. If "." encounters a function, it steps into it. This means it enters the function and afterwards remains at the first line.

There may be as many spaces as desired between "." and the integer.

Examples

```
.100
.  3
.       7
```

See also

.. (Double Dot), RETURN

.. (Double Dot)

Synopsis

Continue execution for one or more program lines without showing procedure calls.

Syntax

.. {*n*}

Parameters

n is an optional integer used to specify the number of program lines to be executed. If n is omitted, 1 is assumed.

Remarks

Double dot is used to execute the next program line(s) of a program.

Double dot executes the next n program lines where n is the command argument or one (1). Single-Step Program Execution

If Double dot encounters a procedure or function call, it steps through it. This means the procedure is executed and control advances to the next statement of the calling program without showing or stopping at the lines in the called procedure.

There may be as many spaces as desired between Double dot and the integer.

Examples

```
. . 3
. .         5
```

See also

. (Dot), RETURN

AUTO

Synopsis

Automatic entry of program line numbers.

Syntax

{ *starting line-no* } **AUTO** { *increment* }

Parameters

starting line-no is an optional first line number, or an existing label, in the current program at which to begin entering new statements. If omitted, 10 is the default. If an existing statement label is supplied, entry begins at its current line number.

increment is the optional line number increment for automatic entry. If omitted, 10 is the default. A label may not be supplied as the increment.

Remarks

AUTO is used to enable automatic entry of program line numbers.

AUTO displays the *line-no*, allowing entry of the new statement. If the *line-no* already exists, it is replaced by the new entry if the statement is accepted without error.

If an error is detected in the statement entered, a message is displayed, any original line is unchanged and the same *line-no* is requested.

AUTO is terminated by pressing the key associated with the **ESC** function.

Examples

```
AUTO 100
100 AUTO 1
```

Errors

Various syntax and encoding errors.

See also

BREAK

Synopsis

Create a breakpoint to the Debugger at a specified position or event, where processing or reception of data is to be interrupted.

Syntax

BREAK *position*
BREAK IF ERROR

Parameters

position is the point in the program at which processing or reception of data is to be interrupted. A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of a position parameter, refer to Appendix C.

If error sets a breakpoint for the Debugger. The breakpoint occurs on error detection, before executing any program error trapping (e.g., if err 0).

Remarks

BREAK is used to create a breakpoint for any purpose, such as examining data to allow for a display of variables and statuses.

The number of breakpoints that can be created within a program is limited only by memory available.

Examples

```
BREAK 60
BREAK if error
```

See also

NOBREAK, STATUS, XBREAK

CANCEL

Synopsis

Clear all variables and other runtime context.

Syntax

CANCEL

Parameters

None.

Remarks

CANCEL clears variables and other runtime context of the current running program to allow programming modifications. When you exit from a program or from the Debugger, you are left in a stopped state. **CANCEL** cancels this stopped state.

CANCEL is automatically performed before any program change to a stopped program that requires recompilation of the program. Changes to defined *structure variables* and **OPTION** statements are examples of program changes requiring recompiling of a program. Whenever an automatic **CANCEL** occurs due to a coding change that would invalidate the stopped program, the following message is displayed:

```
Notice: program edit required recompilation, variables cleared
```

CANCEL clears all stacks, variables, channels and runtime information. Once a running program is canceled, **CONTINUE**, **GO**, single-step and *line-no* **RUN** commands are disallowed. An initial **RUN** must be performed to re-initialize the program and open any required channels.

Examples

```
CANCEL
```

See also

CHECK

Synopsis

Scan program for proper structure and linkage.

Syntax

CHECK { -s } { -u }

Parameters

The optional -s switch is used to invoke a limited check operation, in which only program structures are checked.

The optional –u switch is used to perform a check with additional warning messages for all variables that have not been DIMmed or otherwise declared. This check is automatically performed for any program unit containing an **OPTION AUTO DIM OFF** statement.

Remarks

CHECK is used to verify the current program for proper structures and external linkage. The command performs a trial compilation and link of the program in order to provoke errors from the compiler and linker. (Check -S omits the linker check.) Thus, **CHECK** is similar to a **SAVE** without the write-to-file part.

During the compilation phase, **CHECK** detects any structural errors such as an extra **ENDIF** statement or a missing structure definition.

During the linking phase, **CHECK** combines a set of one or more programs capable of running, based on their various procedural references to each other. Linking occurs only if the “main” program makes references to procedures which do not reside within that program.

Errors are reported as they are encountered during the process

If no errors are detected, the message 'No errors detected' is displayed.

Whereas program statements are syntactically checked during entry, **CHECK** essentially performs an compilation of the program to ensure that it is structurally correct.

Examples

```
Check
Check -s
Check -u
```

See also

SAVE, RUN

CONTINUE

Synopsis

Resume execution of stopped program.

Syntax

CONTIN
CONTINUE

Parameters

None.

Remarks

CONTINUE is used to resume the execution of a program stopped by Breakpoint, **STOP**, non-trapped error, or by the user (usually **ESCAPE** or **CTRL D**).

Execution resumes at the first instruction not yet executed in sequence.

Examples

```
CONTINUE
CONTIN
```

See also

BREAK, NOBREAK, XBREAK

CONVERT

Synopsis

Convert UniBasic statements from a text file.

Syntax

CONVERT *textfilename* {,*alternate profile*}

Parameters

textfilename is the name of any ASCII text file that contains UniBasic program statements.

The optional *alternate profile* directs **CONVERT** to that file for conversion information. If this profile is not supplied, dL4 assumes that your conversion profile is stored within the file *convert.prf*. If the alternate profile is not supplied, you should obtain it from your installation file or tape.

Remarks

CONVERT is used when you convert your UniBasic statements from text files.

CONVERT is similar to **LOAD**, except that certain syntax conversions are automatically performed by **CONVERT** to assist in migrating programs from UniBasic, IRIS, or BITS to dL4. The **CONVERT** command converts a whole file at a time, statement by statement.

In addition to handling syntactical changes, **CONVERT** utilizes the file named *convert.prf*, or any alternate profile selected to assist in the migration of User Calls.

CONVERT performs the following functions automatically:

- **INDEX** #c is changed to **SEARCH** #c
- % operator is changed to **MOD**
- Semicolons are converted to commas as required in **READ** and **WRITE** statements
- **CREATE** is changed to **BUILD**
- UniBasic Multi-LET with ‘,’ separators is converted to ‘;’ separator
- Inserts spaces for missing space separators in mnemonic strings – ‘**CSBU’** is converted to ‘**CS BU’**
- Keyword collisions are corrected by appending ‘_’ to a symbol
- Characters in quoted strings are converted to Unicode characters
- **CHN** is converted to **CHF**
- Missing parentheses around function arguments are automatically added
- When using a conversion profile, parenthesized subscript expressions are converted bracketed subscripts (“A(5)” becomes “A[5]”)
- **ERM** is converted to **ERM$** or the intrinsic function **ERRMSG$** if specified in the conversion profile
- **MSF** is converted to **MSF$**
- **STR** is converted to **STR$**
- **PEEK** statements are converted to **PAUSE NOT**(“orig text”) statements.
- **POKE** statements are converted to **PAUSE NOT**(“orig text”) statements.
- **SECTOR** statements are converted to **PAUSE NOT**(“orig text”) statements.
- **TAPE** statements are converted to **PAUSE NOT**(“orig text”) statements.
- **REM** is not required to be followed by a space
- **RESTORE** is changed to **RESTOR**

By utilizing the conversion profile, User Calls are remapped from the pre-dL4 forms:

CALL NN, parameters or **CALL** $NAME, parameters to the form:

Call procedure-name (arguments)

CONVERT inserts the appropriate **DECLARE** statements.

Examples

```
CONVERT ar.text arprofile
```

See also

User Calls

DELETE

Synopsis

Delete program statements.

Syntax

{ *starting line-no* } **DELETE** { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to delete. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

ending line-no is an optional last *line-no* in the current program to delete. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Remarks

DELETE with starting and ending statement numbers is used to delete a range of statements, and **DELETE** without a *starting line-no* or *ending line-no* is used to remove all statements in the current program

DELETE does not clear variable names and values.

Examples

```
9900 DELETE
100 DELETE 200
DELETE
```

See also

NEW

DISPLAY

Synopsis

Display the values of specified variables of the current running program.

Syntax

DISPLAY <*var*>[,<*var*>...]

Parameters

Var indicates the names of specified variables of the current program. Each variable structure member, or array element, appears on a separate line. You can specify an unlimited number of variables, and their display extends onto new pages as necessary. **DISPLAY** prints all members of a structure variable, and all elements of an array variable. A subscript of the form "i;j" can be used to display array elements i through j.

Remarks

DISPLAY is restricted solely to the values of variables.

Examples

```
DISPLAY rec.
DISPLAY rec. salary
DISPLAY a$
DISPLAY a$[1,10]
DISPLAY a.b[5,2].c.d[3;5]
```

See also

DUMP

Synopsis

List a program to a text data file

Syntax

{ *starting line-no* } **DUMP** { *switches* } *filename***{!}**{/*text*/}{ *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the output. Each parameter is a single letter preceded by a hyphen (**-**)**.** There is only one switch available with **DUMP**:

- u Force line number mode. The -u option causes the command to display lines with line numbers whether or not the program uses or needs line numbers.

filename is a relative or absolute filename to which you have write permission. If the *filename* already exists, it must be terminated by an exclamation point (**!**) to replace its contents. The file is built as a standard Text File. If *filename* begins with a dollar sign (such as $LPT), the pipe driver will be opened instead of a Text File.

/*text/* is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Examples

```
100 DUMP FILENAME 200
INPUT: DUMP  FILENAME! END_INPUT:
```

See also

LIST, FIND, Starting and Ending Line numbers

EDIT

Synopsis

Edit and change an existing statement.

Syntax

EDIT *stn*

Parameters

stn is the statement number of an existing statement to be edited within the program.

Remarks

EDIT is used to display the statement and to request input from the user. All of the built-in terminal input actions for editing are supported:

Back	Move cursor one space to left without erasing a character.
Backspace	Delete character before cursor.
Cancel	Clear input buffer.
Delete	Delete current input character identified by cursor.
End	Move cursor to last space of input.
Enter	Terminate input and transmit data to system for processing.
Forward	Move cursor to right one character.
Home	Move cursor to first character of input, without affecting characters.
Insert	Toggle between Overwrite and Insert mode.
NextWord	Advance to first character of next word.
PrevWord	Move cursor to first character of last word.
ToggleEcho	Toggle between Echo mode on and off.
Abort	Discard contents of input buffer and return to appropriate prompt.
Escape	Discard contents of input buffer and return to appropriate prompt.

The **EDIT** command treats the line to be edited as typeahead and then allows the use of normal input editing keys to support editing. In addition, the up and down arrow keys can be used to edit the previous or next lines of the program.

Any time an error is detected during program entry, the line is redisplayed and **EDIT** is entered automatically.

Examples

```
EDIT 10
```

See also

EXAMINE

Synopsis

Select procedure, function, library, or call stack level.

Syntax

EXAMINE *position*

Parameters

position is the name of a program or procedure, library, or line number.

Remarks

EXAMINE is used to find procedures/functions and to select the library or call stack level for examination with other commands. For example, to list lines in a library used by the current program, enter the command "**EXAMINE** libraryname" followed by **LIST** commands. To re-select the main program, enter the command "**EXAMINE** programname". **EXAMINE** also changes the current debugging stack level. The command selects the first stack level applying to the given procedure or line, and informs you which level is the current stack level and which line is the current program line on that level.

EXAMINE and **LEVEL** perform much the same function, although in different ways. The **LEVEL** command is used to move the current Debugger view between levels in the **CALL**/**SWAP** stack, as displayed in the **STATUS** command.

Examples

```
EXAMINE 1000
Examining [0];1
EXAMINE 1030
Examining 1020;1
```

See also

EXAMINE

EXIT

Synopsis

Exit BASIC environment to SCOPE environment.

Syntax

EXIT

Parameters

None.

Remarks

EXIT is used to terminate *BASIC mode.* Before terminating BASIC:

- All user channels are closed
- Program tracing is ended.
- The alternate escape is enabled as an abort event.
- Command mode is entered.

EXIT is identical to pressing the interrupt key (**CTRL C**). However, it may be included in a text file executed using **EXEC**.

Examples

```
EXIT
```

See also

EXEC

FILE

Synopsis

Display current program and all open channels.

Syntax

FILE {*options*}

Parameters

{options} is:

-h Display all open hidden channels

#no Display all channel function information for channel #

Remarks

FILE is used to display the name of the program loaded into memory and information about all opened channels.

chnl#	Driver Class	Driver Title	Filename
11	Window	Terminal Window	" Keyword "
12	Raw	Raw File	"/(fd0)/(fd2)

chnl# is the number of a dL4 channel.

Driver Class is the class of dL4 driver associated with the opened channel.

Driver Title is the name of the actual driver servicing the opened channel.

Filename is the name of the opened file or device.

If a channel number is used, the file displays **Chf**(0) to **Chf**(11) and **Chf**$(0) to **Chf**$(11) values.

Examples

```
FILE -h
FILE #7
```

See also

Chf() and Chf$ in dL4 Language Reference Guide

FIND

Synopsis

Search and list selected program statements.

Syntax

{ *starting line-no* } **FIND** { *switches* } ***/*** *text* ***/*** { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to search and decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (**-**) :

V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is the default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to search and decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Remarks

FIND is used to search for desired program statement by reading BASIC object code. When a statement is found, **FIND** converts the BASIC object code back into text and lists it. The command looks for every full and partial match of the desired program statement. For example, if you specify Swap, both Swap and Swapf are located.

Examples

```
FIND -V /open #/
100 FIND -V /variable =/ 500
100 FIND /chain/ INPUT:
```

See also

LIST, SHOW, Starting and Ending Line numbers

GO

Synopsis

Resume execution of stopped program.

Syntax

GO {*procedure/line*}
{*line*|*label*} **GO** {*position*}

Parameters

position indicates where execution is to stop. A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of a position parameter, refer to Appendix C.

Remarks

GO is used to resume the execution of a program stopped by:

- Breakpoint
- **STOP**
- a non-trapped error

If debugging options such as Breakpoint or Single Step are used, execution resumes at the first instruction in sequence not yet executed. Entry into Debugger mode using **STOP**, Breakpoint, or a non-trapped error leaves all channels open.

Examples

```
GO 150
GO 780
```

See also

BREAK, CONTINUE, NOBREAK, XBREAK

HELP

Synopsis

Print text description of an error.

Syntax

HELP *error number*

Parameters

error number is any positive integer representing a dL4 error number as returned by the **SPC(8)** function. Enter only one number at a time: **HELP** 25 26 is a format error.

Remarks

HELP is used to investigate the causes of error situations. If no *error number* is specified, the text description of the last error is displayed. If no error exists, the string "No such error" is displayed. This string is also displayed if you enter **HELP** with no error number.

error number is assumed to be an error number returned by **SPC(8)**. Alternate error numbers, such as those returned by **ERR**(0) may not be supplied to the **HELP** command as the *error number*.

Examples

```
HELP 23
HELP 255
```

See also

LABEL

Synopsis

Convert statement numbers to labels.

Syntax

LABEL

Parameters

None.

Remarks

LABEL is used to remove all line number references within a program. For example, the **GOTO**, ESCSET, and **GOSUB** statements, applying to a statement number in UniBasic code, are converted from the form:

```
GOTO  NNNN
```

to the form:

```
GOTO LNNNN
```

where NNNN is the old statement number, and LNNNN is the new label for that line number.

Once statement numbers are removed from a program, they are omitted during a **DUMP** operation, and supplied automatically during a **LOAD**. Programs without line numbers are more easily maintained and allow the use of modern development tools, such as screen editors, cut and paste, source-code control systems, etc.

Examples

```
LABEL
LNNN:
GOTO LNNN
```

See also

LIST

Synopsis

Decode and list dL4 program statements.

Syntax

{ *starting line-no* } **LIST** { *switches* } { / *text* / } { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is assumed.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (**-**)**.** There is only one switch available with **LIST**:

- V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is enabled by default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded and listed. All other statements are not listed.

ending line-no is an optional last *line-no* in the current program to decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is assumed.

Remarks

LIST is used to decode BASIC object code which has been **LOAD**ed, convert it back into text, and list it in statement number sequence onscreen.

To decode statements to a *file*, *device* or *pipe*, use **DUMP**.

LIST always uses line numbers to display statements.

Examples

```
LIST -V
LIST -V /WRITE #0/
100 LIST -V 500
INPUT: LIST END_INPUT:
```

See also

FIND, DUMP, SHOW

LOAD

Synopsis

Load dL4 statements from a text or program file.

Syntax

LOAD *filename* { *starting line-no* } **LOAD** { *filename* **|** *-filename* } { *increment line-no* }

Parameters

starting line-no is an optional first *line-no* to use for numbering incoming text program lines. If omitted, 10 is assumed. If the incoming text program has line numbers, they are used and any supplied *starting line-no* is effectively ignored.

filename is any Text File or Program file to which you have read-permission.

The optional '-' preceding the *filename* may be used to remove comments from the incoming program.

increment line-no is an optional increment value to use for numbering incoming text program lines. If omitted, 10 is assumed. If the incoming text program has line numbers, they are used and the supplied *increment line-no* is effectively ignored.

Remarks

LOAD is used to merge the program lines contained in the specified text file or to load a new program file into memory. Note that loading a program file replaces the current program. As each line of text is loaded, it is added to the current program (which may have been empty) in memory. The statements in the text file need not be in any particular order if the program uses line numbers. If any statement already exists, it is replaced. For example, assume the following program is currently in memory:

```
10 A=A+1
20 B=SQR(A)
```

and a **LOAD** is performed from a text file containing:

```
20  C=SQR(A)+B
26 If A=30 Then Stop
30 Goto 100
```

The resulting program would be:

```
10 A=A+1
20 C=SQR(A) +B
26 If A=30 Then End
30 Goto 100
```

If the incoming program does not use line numbers, the statements in the text file must be in the exact order required by the program. The optional *starting line-no* and *increment line-no*, or the defaults, is used to add or replace existing lines within the program.

LOAD-filename strips all comment text, not merely trailing ! comments.

LOAD merges lines without line numbers, rather than executing them.

To load a new program source, you must execute **NEW** prior to **LOAD**.

Examples

```
LOAD  sys/program
9000  load sys/inputsub  10
```

See also

NEW

NEW

Synopsis

Clear memory for a new program.

Syntax

NEW

Parameters

None.

Remarks

NEW is used to:

- clear (close) all open channels
- clear workspace
- release (erases) all breakpoints
- release memory, memory-resident programs, and variables
- reset autoline number
- reset auto increment

Examples

```
NEW
```

See also

LOAD

NOBREAK

Synopsis

Delete a breakpoint at the specified position or event.

Syntax

NOBREAK *position*
NOBREAK IF ERROR

Parameters

A *position* parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of a position parameter, refer to Appendix C.

Remarks

NOBREAK is used to delete or clear a breakpoint at a specified position or positions. It is not a toggle for **BREAK**, although you may use **NOBREAK** and then **BREAK** together to reassign breakpoints.

Entering **NOBREAK** without a procedure or line causes a deletion of all breakpoints.

Examples

```
NOBREAK 600
NOBREAK IF ERROR
NOBREAK
```

See also

BREAK, XBREAK

OEM

Synopsis

List the currently authorized OSNs (OEM Security Number).

Syntax

OEM {TEMP*}*

Parameters

TEMP causes the command to prompt for a temporary OSN (OEM Security Number)**.**

Remarks

The **OEM** command lists the currently authorized OSNs. If the TEMP option is used ("OEM TEMP"), the **OEM** command will prompt for a temporary OSN to be used only by the current SCOPE session.

Examples

```
#OEM
#OEM TEMP
```

See also

LOADSAVE, PSAVE, SAVE

PDUMP

Synopsis

List program status, variables, and other information to a text data file

Syntax

PDUMP *filename*{**!**}

Parameters

filename is a relative or absolute filename to which you have write permission. If the *filename* already exists, it must be terminated by an exclamation point (**!**) to replace its contents. The file is built as a standard Text File. If *filename* begins with a dollar sign (such as $LPT), the pipe driver will be opened instead of a Text File.

Examples

```
PDUMP FILENAME
```

See also

DUMP

PSAVE

Synopsis

Create an OSN (OEM Security Number) protected program.

Syntax

PSAVE *n*, *filename.expr* ...

Parameters

filename.expr is the program file name to be created**.**

n is the number of the OSN listed by the OEM command

Remarks

The **PSAVE** command is used to create OSN protected programs. The **PSAVE** command is identical to the **SAVE** command except for an optional OSN number that can precede the **SAVE** filename. For example, the command "PSAVE 2,menu" would save the current program as "menu" after protecting it to require the second OSN listed by the OEM command. Protected programs can be created only if the specified OSN is a master OSN.

Examples

```
#PSAVE 4, "file1"
```

See also

LOADSAVE, OEM, SAVE

RENUMBER

Synopsis

Renumber statements in a program.

Syntax

{*begin stn*} **RENUMB** {*step*}
{*begin stn*} **RENUMBER** {*step*}

Parameters

begin stn is the optional first statement number to use for the renumbered program. If omitted, 10 is assumed. If begin *stn* is a label, its current *stn* is used as the first statement number.

Step is the optional increment to use between the renumbered lines. If omitted, 10 is assumed.

Remarks

RENUMBER is used to make room for new statements between sequentially-numbered lines.

If there's an error, no renumbering takes place.

Examples

```
1000 RENUMB 30
2000 RENUMBER 20
```

See also

Statement Numbers, Starting and Ending Statement Numbers, LABEL

RUN

Synopsis

Execute a program and optionally enter the Debugger at a specified line.

Syntax

{*line-no*} **RUN** {*position*}

Parameters

line no is the line number where the program is to be restarted.

position is the breakpoint where the program is to stop and enter the Debugger. For a full definition of a position parameter, refer to Appendix C.

line-no **RUN** is used to start execution of a program. A line number is specified to restart execution of a program using any existing variable values or open channels. Using the **RUN** command without a current program results in an empty program being executed. A READY message is printed and the user is left in BASIC mode.

Remarks

The **GO** command resumes execution of a program stopped by a breakpoint.

Examples

```
RUN 1600
100 RUN
```

See also

SAVE

Synopsis

SAVE the current program.

Syntax

SAVE { -l *n* }{-ro} {{ <*attributes*> } { (*options*) } { *filename*{!} } }

Parameters

The -l *n* option is used to create an OSN (OEM Security Number) protected program. The value "*n*" is the number of a master OSN as listed by the SCOPE **OEM** command.

<*attributes*> are any optional valid file *attributes, protections*, or *permissions* to apply to the file on creation. Standard IRIS, BITS, or UNIX-style permissions may be supplied. If omitted, file creation is defaulted to Read/Write for all users, subject to any operating system masking in effect. If <*attributes*> are supplied, a *filename* must follow.

(*options*) are any optional program file options such as "exec=command", "stdexec", or "netexec".

filename is any optional *filename* or full *pathname* to a directory to which you have write-permission. A filename is optional if the file has previously been saved. If it has not been saved, then the filename is mandatory in the command. If the filename is omitted, the original *filename* for the program in memory is used.

Remarks

Prior to saving a program, it compiles the program which may result in errors. A compilation error still allows a program to be saved.

The "run-only" option, Save -RO, is used to produce "run-only" program files. Save -RO creates a run-only file which strips symbols so that a file cannot be listed, dumped, debugged, or modified in any way. The creation process is irreversible and secure since the information needed to list the program is actually discarded. A Save -RO file enables the developer to:

- Protect sensitive or trade-secret source code from theft or modification
- Safely embed security checks in a program, allowing it to run only on authorized systems.

The "exec=command" option can be used under Unix with execute permissions to create a program that can be executed directly from a Unix shell prompt. The command value is the path of run along with any run command line options. The option "stdexec" is equivalent to "exec=!#/usr/bin/run". The option "netexec" is equivalent to 'exec=!#/usr/bin/run –NB'. On Windows systems, file extensions can be used and associated to create equivalent functionality.

Active channels and variables are undisturbed.

Examples

```
SAVE <22> prog!
SAVE <755> (stdexec) prog!
SAVE <W> dat!
SAVE
```

See also

CHECK, OEM, SAVE

SHOW

Synopsis

Search and list selected program statements.

Syntax

{ *starting line-no* } **SHOW** { *switches* } ***/*** *text* ***/*** { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to search and decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (**-**) :

V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is enabled by default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to search and decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Remarks

SHOW is used to search for desired program statement by reading BASIC object code. When a statement is found, **SHOW** converts the BASIC object code back into text and lists it. The command looks for every full and partial match of the desired program statement. For example, if you specify Swap, both Swap and Swapf are located.

Examples

```
SHOW -V /open #/
100 SHOW -V /variable =/ 500
100 SHOW /chain/ INPUT:
```

See also

LIST, SHOW, Starting and Ending Line numbers

SIZE

Synopsis

Display memory usage for current program/data.

Syntax

SIZE { -l }

Parameters

The -l option causes the sizes and names of all linked libraries to be displayed.

Remarks

SIZE is used to display the amount of memory allocated for the storage of a current program and variables. For example:

```
 Code    Data  By
 4376       0  menudispatch
 4376       0  Total
```

Code is the number of bytes used to store the *BASIC object code* encoded program.

Data is the number of bytes used to store data for variables.

Code and Data space is displayed separately for each nested subprogram.

Examples

```
SIZE
SIZE -l
```

See also

NEW

STATUS

Synopsis

Prints the name of current program file and execution status.

Syntax

STATUS {BREAKPOINT | MACHINE |SYSTEM | UNIT}

Parameters

{BREAKPOINT} displays each of the existing breakpoints.

{MACHINE} displays current characteristics of the current program as a whole.

{SYSTEM}displays the current directory and other system information.

{UNIT} displays current characteristics of the program unit being examined.

Remarks

STATUS is used to determine the current execution status and program filename, breakpoints, and other status information, which it prints. The current execution status is displayed after a breakpoint. One line is printed for each function call, procedure call, subprogram call, program **SWAP**, or **GOSUB** that is still in progress.

Most of the displayed lines take the form "xxx [n] location", where "xxx" is either "-→" if this is the current examination level, as controlled by the **LEVEL** and **EXAMINE** commands, or three (3) spaces before the status indication.

The value of "n" is the stack level, starting at zero in the main program and increasing with each function call, procedure call, subprogram call, or program **SWAP**. The string "location" has one of the following forms (l = line number; s = statement number within that line, starting at 1):

l;s
local: l;s
external: l;s
pgm: l;s
external: local: l;s
lib: external: local: l;s
lib: external: l;s
pgm: l;s
pgm: local: l;s
pgm: external: l;s
pgm: external: local: l;s
pgm: lib: external: l;s

pgm is the name of the program containing the line.

lib is the name of the library containing the line.

external is an external procedure or function name.

local is a local procedure or function name.

For example:

```
--> [1] PrintHello:20;1
    [0] 40;1
```

In the above example, the program has stopped at the first statement of line 20 within a **CALL** to the procedure "PrintHello".

If a subprogram call (**CALL** "filename") or **SWAP** is in progress, then either:

" filename - **CALL**ed"

or:

" filename - **SWAP**ed"

is printed at the point in the stack at which the call or **SWAP** occurred:

```
--> [1] SUBPROGRAM:10;1
  SUBPROGRAM - CALLed
    [0] 50;1
--> [1] SWAPPROGRAM:10;1
  SWAPPROGRAM - SWAPed
    [0] 60;1
```

The **STATUS** command considers each program, external function, or external procedure to be a separate program unit with its own **GOSUB** stack. If any entries are present in a **GOSUB** stack to indicate that a **GOSUB** has occurred without a **RETURN** or other action to pop the stack, then the stack is displayed as in the following example:

```
--> [1] PRINTHELLO:20;1
        GOSUB Stack:
        [0] 50;1
    [0] 40;1
```

Each line of the **GOSUB** display shows the line number and the statement number at which the **GOSUB** was executed.

Status Breakpoint

The *status breakpoint* command displays each of the existing breakpoints, using the same "pgm:lib:external:local:line;stmt" format as the *current execution status*. If error breakpoints (**BREAK IF ERROR**) are enabled, the line "Break If Error" is also printed:

```
dbg>status b
PrintHello:20;1
```

Status Machine

The *status machine* command, which displays current characteristics of the current program as a whole, is described in the **EXAMPLES** subsection.

Status System

The *status system* command, which displays current directory and other system information, is described in the EXAMPLES subsection.

Status Unit

The *status unit* command displays current characteristics of the program unit being examined. Each program, external function, or external procedure is a separate program unit. These characteristics are described in the **EXAMPLES** subsection.

Entering **STATUS** * produces a Format error.

Examples

If you enter **STATUS** without a parameter, the screen displays:

```
status
-->[0] 90;1
```

where [0] indicates level 0, 90 indicates line number, and 1 indicates statement number within the line number.

If you enter **STATUS** with the {MACHINE} parameter, the screen displays:

```
status machine
Default input channel: 102
Default output channel: 102
Trace channel: None
```

```
Command line string: ""
Hot-key program: "swap.run"
```

where the *Default input channel* is the channel number used by any non-channel **INPUT** statement.

The *Default output channel* is the channel number used by any non-channel **PRINT** statement.

The *Trace channel* is the channel number used for program tracing, if enabled by the **TRACE** statement or command.

The *Command line string* is is the command line by which the current program was invoked.

The *Hot-key program* is the name of the program, if any, to be invoked by the **SWAP** key. The program name can be set by using the **SWAPF** intrinsic procedure.

If you enter **STATUS** with the {SYSTEM} parameter, the screen displays:

```
Current directory: C:\Program Files\dL4\SAMPLES
Port number:       4094
Number of users:   2
MSC(7) value:      257
SPC(5) value:      257
SPC(7) value:      0
```

If you enter **STATUS** with the {**UNIT**} parameter, the screen displays:

```
status unit
Current position: [0] 90;1
Last error number: 0
Last error position: n/a
Last error text: ""
Last END or STOP: 0
DATA position: 0
Last determinant: <Not-A-Number>
Last input element: 0
Last input size: 0
Input pend mode: On
Number precision: %3
Date precision: %3
LIB directory: ""
```

status unit is a status listing.

Current position is the current execution location within the program unit. [0] 90;1 indicates level 0, line 90, statement 1 within the line.

Last error number indicates line where the last error occurs. 0 indicates there was no last error.

Last error position is not available: n/a indicates there is no last error position.

Last error text is an English phrase when available. "" means none was found.

Last END or STOP is the location, if any, of the last **END** or **STOP** statement that was executed. 0 indicates no such location exists.

DATA position is is the current **DATA** line number to be used by non-channel **READ** statements. 0 is the current value.

Last determinant is the current value of the "**DET**(0)" function, the determinant generated by the last matrix inversion statement. *<Not-A-Number>* indicates the current value is undefined; most likely no **MAT INV** statement has been executed.

Last input element is the current value of "**MSC**(1)". In this case, the current value is 0.

Last input size is is the current value of "**SPC**(17)". The current value is 0.

Input pend mode is either On or Off.

Number precision is the precision to be applied to any newly-created numeric variable. The current precision is %3.

Date precision is the precision to be applied to any newly-created date variable. The current precision is %3.

LIB directory is the current value of "**MSC**$(6)".

See also

END, EXAMINE, LEVEL

TRACE

Synopsis

Enable statement trace debugging.

Syntax

TRACE {ON {*#channel*}| OFF}

Parameters

channel indicates the channel where TRACE output is to be sent.

Remarks

The **TRACE** command is used to enable **TRACE ON** and **TRACE OFF**. *Trace mode* is used to observe the line number program flow without performing single steps. dL4 provides various ways to handle tracing:

To turn Trace mode on, use **TRACE ON** or **TRACE ON** *#chn.num*.

To turn Trace mode off, use **TRACE OFF**.

The **TRACE ON** statement can be followed by an optional channel number for redirecting trace output to a file or driver. The channel number that is given must be opened prior to executing the **TRACE** statement. If the channel is subsequently closed, trace output defaults to the terminal.

Tracing is automatically disabled when another program is loaded using **CHAIN**, **SWAP**, or **SPAWN**. Cancelling a running program does not turn **TRACE** off.

Examples

```
TRACE
TRACE ON
TRACE OFF
TRACE ON #5
```

See also

SYSTEM 20, SYSTEM 21 in dL4 Language Reference Guide

VARIABLE

Synopsis

Display variable names that are defined in the currently selected procedure.

Syntax

VARIABLE

Parameters

None.

Remarks

VARIABLE displays a list of the variable names defined in the currently selected procedure.

Examples

```
VARIABLE
```

See also

LEVEL, LIST, STATUS

XBREAK

Synopsis

Create a breakpoint to the Debugger at a specified position or event, where processing or reception of data is to be interrupted.

Syntax

XBREAK *position*
XBREAK IF ERROR

Parameters

position is the point in the program at which processing or reception of data is to be interrupted. A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of a position parameter, refer to Appendix C. Breakpoints created with **XBREAK** will be applied to both the current program and to any program which is loaded by CHAIN, SWAP, or CALL statements during execution. Thus, an "XBREAK 60" command would create a breakpoint at line 60 in the current program and at line 60 in any program that was entered during execution. The position specified in the **XBREAK** command does not have to exist in the current program.

If error sets a breakpoint for the Debugger. The breakpoint occurs on error detection, before executing any program error trapping (e.g., if err 0).

Remarks

XBREAK is used to create a breakpoint for any purpose, such as examining data to allow for a display of variables and statuses.

The number of breakpoints that can be created within a program is limited only by memory available.

Examples

```
XBREAK 60
XBREAK if error
```

See also

BREAK, NOBREAK, STATUS

Chapter 4 - Debugger Commands

A Debugger session is started whenever any of the following events occur:

- step (“.” or “..”) command line count reached
- non-trapped BASIC error or forced termination (**ESCAPE** or **CTRL D**)
- Breakpoint
- **STOP** or **SUSPEND** statement
- Abort
- Untrapped **ESC**ape event

To resume execution, type **GO**. To exit, type **END**.

The Command Abbreviation feature of Debugger mode allows you to issue a command by entering only enough of its letters to form a unique abbreviation, instead of typing the whole word. For example, you could enter “ST” for “**STATUS**”, or “T” for “**TRACE**”. It would not be possible to enter “E” for “**END**”, because “E” could also apply to **EXAMINE**. This abbreviation facility is available only in Debugger mode.

To display a list of commands, type “?”.

The Debugger is available only through the SCOPE Command Line IDE. Those programs that are run from outside the Command Line IDE do not have access to the debugger.

This chapter describes the Debugger commands in detail. The table below lists and briefly describes the commands.

COMMAND	DESCRIPTION
? (Question Mark)	Display a list of commands or a description of <command>.
; (Semicolon)	Display the values of specified variables of the current running program.
! (Exclamation Point)	Execute external operating system command
.	Execute the next n program lines
..	Execute next program line and step through function
BREAK	Create a breakpoint at specified position or positions.
CONTINUE	Resume execution of stopped program.
DISPLAY	Display the values of specified variables of the current running program.
DUMP	List a program to a text data file.
END	Exit from Debugger.
EXAMINE	Examine and select which is the current program mode.
EXIT	Abort program and exit BASIC.
FILE	Display current program and open all files.
FIND	Search and list selected program statements.
GO	Resume execution of stopped program.
HELP	Print text description of an error.
LEVEL	Moves current Debugger view between levels in the CALL/SWAP stack.
LET	Assign value to variable.
LIST	Decode dL4 statements.
NOBREAK	Delete a breakpoint at specified position or positions.
OEM	Lists the currently authorized OSNs (OEM Security Number).
PDUMP	List program status, variables, and other information to a text data file
RETURN	Continue execution until the current procedure or function exits
SHOW	Search and list selected program statements.
SIZE	Display memory usage for current program/data.
STATUS	Print the name of the current program file and execution status.
TRACE	Enable statement trace debugging.
VARIABLE	Display variable names in current procedure
WB	Move the debug window to the bottom of the screen.

WF	Resize the debug window to full screen.
WH	Resize the debug window to half screen.
WINDOW	Move, resize, or change treatment of the debug window.
WS	Resize the debug window to quarter screen.
WT	Move the debug window to the top of the screen.
XBREAK	Create breakpoint at specified position or positions

? (QUESTION MARK)

Synopsis

Display a list of commands or a description of <command>.

Syntax

? {*command*}

Parameters

command is one of the debug commands as listed by entering the "**?**" without parameters.

Remarks

Displays the available debugger commands when **?** is entered without parameters.

When a valid debugger command is entered as a parameter, displays a HELP message for that command.

Examples

```
?
? go
```

See also

; (SEMICOLON)

Synopsis

Display the values of specified variables of the current running program.

Syntax

; <*var*>[,<*var*>...]

Parameters

var indicates the names of specified variables of the current program. Each variable structure member, or array element, appears on a separate line. You can specify an unlimited number of variables, and their display extends onto new pages as necessary. Semicolon (";") prints all members of a structure variable, and all elements of an array variable. A subscript of the form "i;j" can be used to display array elements i through j.

; is restricted solely to the values of variables.

Examples

```
; rec.
; rec. salary
; a$
; a$[1,10]
; a.b[5,2].c.d[3;5]
```

See also

! (Exclamation Point)

Synopsis

Execute external operating system command

Syntax

! *command*

Parameters

command is any operating system (or null) command to be executed by a sub-shell.

Remarks

The Exclamation Point command is used to execute an external operating system command.

All system commands are executed by a separate child process, effectively putting dL4 to sleep until the command terminates. Changes to environmental variables and current working directory within a child processes are effective only during that process. Upon termination of the command, the parent (dL4) resumes execution unaware of the child's activities.

Examples

```
!ls -l
!vi query.bas
!edit query.bas
```

See Also

CD, operating system documentation.

. (Dot)

Synopsis

Continue execution for one or more program lines..

Syntax

. {*n*}

Parameters

n is an optional integer used to specify the number of lines to step through the program. If n is omitted, 1 is assumed.

Remarks

Single statement execution is performed by entering a **.** and pressing Return. The current statement is executed and the next statement to execute is displayed. Subsequent dots are used to step through the program. To resume normal execution of a program, issue the command **CONTINUE**. For applications relying on **CALL**ed subprograms, single statement execution can be performed for both *stepping through* a subprogram by entering a . and pressing return.

The "." form executes subprograms. If "." encounters a function, it steps into it. This means it enters the function and afterwards remains at the first line.

There may be as many spaces as desired between "." and the integer.

Examples

```
.100
.  3
.       7
```

See also

.. (Double Dot), RETURN

.. (Double Dot)

Synopsis

Continue execution for one or more program lines without showing procedure calls

Syntax

.. {*n*}

Parameters

n is an optional integer used to specify the number of program lines to be executed. If n is omitted, 1 is assumed.

Remarks

Double dot is used to execute the next program line(s) of a program.

Double dot executes the next n program lines where n is the command argument or one (1). Single-Step Program Execution

If Double dot encounters a procedure or function call, it steps through it. This means the procedure is executed and control advances to the next statement of the calling program without showing or stopping at the lines in the called procedure.

There may be as many spaces as desired between Double dot and the integer.

Examples

```
. . 3
. .         5
```

See also

. (Dot), RETURN

BREAK

Synopsis

Create a breakpoint to the Debugger at a specified position or positions, where processing or reception of data is to be interrupted.

Syntax

BREAK *position*
BREAK if error

Parameters

position is the point in the program at which processing or reception of data is to be interrupted. For a full definition of the position parameter, refer to Appendix C.

Remarks

If error indicates that a break into debugger occurs if an error is encountered.

BREAK is used to create a breakpoints for any purpose, such as examining data to allow for a display of variables and statuses.

The number of breakpoints can be created within a program is limited only by memory available.

Examples

```
BREAK 60
BREAK if error
```

See also

CONTINUE, NOBREAK, STATUS, XBREAK

CONTINUE

Synopsis

Resume execution of stopped program.

Syntax

CONTIN
CONTINUE

Parameters

None.

Remarks

CONTINUE is used to resume the execution of a program stopped by Breakpoint, **STOP**, non-trapped error, or by the user (usually **ESCAPE** or **CTRL D**).

Execution resumes at the first instruction not yet executed in sequence.

Examples

```
CONTINUE
CONTIN
```

See also

BREAK, NOBREAK, XBREAK

DISPLAY

Synopsis

Display the values of specified variables of the current running program.

Syntax

DISPLAY <*var*>[,<*var*>...]

Parameters

Var indicates the names of specified variables of the current program. Each variable structure member, or array element, appears on a separate line. You can specify an unlimited number of variables, and their display extends onto new pages as necessary. DISPLAY prints all members of a structure variable, and all elements of an array variable. A subscript of the form “i;j” can be used to display array elements i through j.

Remarks

DISPLAY is restricted solely to the values of variables.

Examples

```
DISPLAY rec.
DISPLAY rec. salary
DISPLAY a$
DISPLAY a$[1,10]
DISPLAY a.b[5,2].c.d[3;5]
```

See also

DUMP

Synopsis

List a program to a text data file.

Syntax

{ *starting line-no* } **DUMP** { *switches* } *filename*{**!**}{/*text*/} { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the output. Each parameter is a single letter preceded by a hyphen (**-**)**.** There is only one switch available with **DUMP**:

- u Force line number mode. The -u option causes the command to display lines with line numbers whether or not the program uses or needs line numbers.

filename is a relative or absolute filename to which you have write permission. If the *filename* already exists, it must be terminated by an exclamation point (**!**) to replace its contents. The file is built as a standard Text File. If *filename* begins with a dollar sign (such as $LPT), then the pipe driver will be opened instead of a Text File.

/*text*/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Examples

```
100 DUMP FILENAME 200
INPUT: DUMP  FILENAME! END_INPUT:
```

See also

LIST, FIND, Starting and Ending Line numbers

END

Synopsis

Exit from Debugger.

Syntax

END

Parameters

None.

Remarks

END is used to exit from the Debugger. The command:

- closes all channels
- forces an exit from all stack levels, leaving the user in the root program (level 0)

You are placed in BASIC mode and the READY prompt is displayed.

Examples

```
END
```

See also

EXAMINE

Synopsis

Examines and selects the current program file.

Syntax

EXAMINE <*procedure/line*>

Parameters

<*procedure*> is the procedure specified for examination.

<*line*> is the line specified for examination.

Remarks

EXAMINE is used to examine and select the current program file. The command selects the current program for examination, not merely a line. The current program is the one **LIST**ed and **DUMP**ed. In addition, **EXAMINE** also changes the current debugging stack level. The command selects the first stack level applying to the given procedure or line, and informs you which level is the current stack level and which line is the current program line on that level.

EXAMINE and **LEVEL** perform much the same function, although in different ways. The **LEVEL** command is used to move the current Debugger view between levels in the **CALL/SWAP** stack, as displayed in the **STATUS** command.

Examples

```
EXAMINE 1000
Examining [0];1
EXAMINE 1030
Examining 1020;1
```

See also

EXAMINE

EXIT

Synopsis

Abort program and exit BASIC.

Syntax

EXIT

Parameters

None.

Remarks

EXIT is used to exit from the Debugger and the BASIC environment. The command:

- closes all channels
- forces an exit from all stack levels, leaving the user at the SCOPE prompt

Examples

```
EXIT
```

See also

END

FILE

Synopsis

Display current program and all open files.

Syntax

FILE {*options*}

Parameters

{*options*} is:

-h	Display all open hidden channels
#no	Display all channel function information for channel #

Remarks

FILE is used to display the name of the program loaded into memory and information about all opened channels.

chnl#	Driver Class	Driver Title	Filename
11	Window	Terminal Window	" Keyword "
12	Raw	Raw File	"/(fd0)/(fd2)

chnl# is the number of a dL4 channel.

Driver Class is the class of dL4 driver associated with the opened channel.

Driver Title is the name of the actual driver servicing the opened channel.

Filename is the name of the opened file or device.

If a channel number is used, the file displays **Chf**(0) to **Chf**(11) and **Chf**$(0) to **Chf**$(11) values.

Examples

```
FILE -h
FILE #7
```

See also

Chf( ) and Chf$ in dL4 Language Reference Guide

FIND

Synopsis

Search and list selected program statements.

Syntax

{ *starting line-no* } **FIND** { *switches* } ***/*** *text* ***/*** { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to search and decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (-) :

V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is enabled by default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text,* that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to search and decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Remarks

FIND is used to search for desired program statements by reading BASIC object code. When a statement is found, **FIND** converts the BASIC object code back into text and lists it. The command looks for every full and partial match of the desired program statement. For example, if you specify Swap, both Swap and Swapf are located.

Examples

```
FIND -V /open #/
100 FIND -V /variable =/ 500
100 FIND /chain/ INPUT:
```

See also

LIST, SHOW, Starting and Ending Line numbers

GO

Synopsis

Resume execution of stopped program.

Syntax

{line} **GO** {*break-procedure/break-line*}

Parameters

Line indicates where execution is to resume.

Break-procedure and *break-line* indicate where execution is to stop.

Remarks

GO is used to resume the execution of a program stopped by:

- Breakpoint
- **STOP**
- a non-trapped error

If debugging options such as Breakpoint or Single Step are used, execution resumes at the first instruction in sequence not yet executed or at the specified line. Entry into Debugger mode using **STOP**, Breakpoint, or a non-trapped error leaves all channels open.

Entry into command mode automatically closes all open channels. To perform operating system commands, use the command to invoke a shell or another copy of dL4.

Examples

```
GO
GO 780
```

See also

BREAK, CONTINUE, XBREAK

HELP

Synopsis

Print text description of an error.

Syntax

HELP *error number*

Parameters

error number is any positive integer representing a dL4 error number as returned by the **SPC(8)** function. Enter only one number at a time: **HELP** 25 26 is a format error.

Remarks

HELP is used to investigate the causes of error situations. If no *error number* is specified, the text description of the last error is displayed. If no error exists, the string "No such error" is displayed. This string is also displayed if you enter **HELP** with no error number.

error number is assumed to be an error number returned by **SPC(8)**. Alternate error numbers, such as those returned by **ERR**(0) may not be supplied to the **HELP** command as the *error number*.

To obtain an error number, use the **ERR**(0) function.

Examples

```
HELP 23
HELP 255
```

See also

LET

Synopsis

Assign value to variable.

Syntax

LET *var* = *value*

Parameters

var is the name of a variable in the program.

value is the data to be assigned to the variable *var*.

Remarks

LET is used to modify the value of a program variable. The data type of var and value must correspond.

Examples

```
LET A$="Y"
LET I=10
```

See also

LEVEL

Synopsis

Moves current Debugger view between levels in the CALL/SWAP stack.

Syntax

LEVEL *number*
LEVEL +
LEVEL -

Parameters

number is the requested level, with 0 being the bottom of the stack, the main program.

"+" means to rise one level above the current level.

"-" means to descend one level below the current level.

Remarks

The **LEVEL** command attempts to select the requested level, and then prints the current level. Any attempt to exceed the top or bottom of the stack simply selects the top or bottom of that stack. For example, **LEVEL** prints the level you request and keeps printing the top level when the limit is reached. In other words, if you ask for Level 5 when there are only 3 levels, **LEVEL** prints Level 3 for you.

Examples

```
LEVEL +
LEVEL -
LEVEL 3
```

See also

EXAMPLE

LIST

Synopsis

Decode and list dL4 program statements.

Syntax

{ *starting line-no* } **LIST** { *switches* } { / *text* / } { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is assumed.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (**-**)**.** There is only one switch available with **LIST**:

- V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is enabled by default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text,* that statement is decoded and listed. All other statements are not listed.

ending line-no is an optional last *line-no* in the current program to decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is assumed.

Remarks

LIST is used to decode BASIC object code which has been **LOAD**ed, convert it back into text, and list it in statement number sequence onscreen.

To decode statements to a *file*, *device* or *pipe,* use **DUMP**.

LIST always uses line numbers to display statements.

Examples

```
LIST -V
LIST -V /WRITE #0/
100 LIST -V 500
INPUT: LIST END_INPUT:
```

See also

FIND, DUMP, Starting and Ending Line numbers

NOBREAK

Synopsis

Delete a breakpoint at the specified position or event.

Syntax

NOBREAK {*position*}
NOBREAK IF ERROR

Parameters

position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of position parameter, refer to Appendix C.

Remarks

NOBREAK is used to delete or clear a breakpoint at a specified position or positions. It is not a toggle for **BREAK**, although you may use **NOBREAK** and then **BREAK** together to reassign breakpoints.

Entering **NOBREAK** without a procedure or line causes a deletion of all breakpoints.

Examples

```
NOBREAK 600
NOBREAK IF ERROR
NOBREAK
```

See also

BREAK, STATIUS, XBREAK

OEM

Synopsis

List the currently authorized OSNs (OEM Security Number).

Syntax

OEM {TEMP*}*

Parameters

TEMP causes the command to prompt for a temporary OSN (OEM Security Number)**.**

Remarks

The **OEM** command lists the currently authorized OSNs. If the TEMP option is used ("OEM TEMP"), the **OEM** command will prompt for a temporary OSN to be used only by the current SCOPE session.

Examples

```
#OEM
#OEM TEMP
```

See also

LOADSAVE, PSAVE, SAVE

PDUMP

Synopsis

List program status, variables, and other information to a text data file

Syntax

PDUMP *filename*{**!**}

Parameters

filename is a relative or absolute filename to which you have write permission. If the *filename* already exists, it must be terminated by an exclamation point (**!**) to replace its contents. The file is built as a standard Text File. If *filename* begins with a dollar sign (such as $LPT), the pipe driver will be opened instead of a Text File.

Examples

```
PDUMP FILENAME
```

See also

DUMP

RETURN

Synopsis

Continue execution until the current procedure or function exits.

Syntax

RETURN

Parameters

None.

Examples

```
RETURN
```

See also

GO

SHOW

Synopsis

Search and list selected program statements.

Syntax

{ *starting line-no* } **SHOW** { *switches* } ***/*** *text* ***/*** { *ending line-no* }

Parameters

starting line-no is an optional first *line-no* in the current program to search and decode. If omitted, the first *line-no* is selected. If the *starting line-no* does not exist, the first existing higher *line-no* is used.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a hyphen (**-**) :

V Visual mode. The -v option causes the command to display lines a screenful at a time. Specifically, if there are more than number-of-lines-per-screen minus two lines to be listed, then the command issues a "[**MORE**]" prompt after displaying number-of-lines-per-screen minus two lines. If the user types a space, the next number-of-lines-per-screen minus two lines are displayed and the "[**MORE**]" prompt is repeated. If the user types the **ENTER** key, the next line is displayed and the "[**MORE**]" prompt is repeated. If all of the selected lines have been displayed, the command terminates. The user can terminate the command at any time by using the **ESCAPE** or **INTERRUPT** character defined for the terminal. The "-V" option is enabled by default for interactive sessions.

/text/ is any optional string to search each statement for. The search is case-insensitive. If omitted, all statements of a program are decoded. To decode only statements containing a specific string, enclose the search *text* within / /. For each statement containing *text*, that statement is decoded, otherwise it is omitted.

ending line-no is an optional last *line-no* in the current program to search and decode. If omitted, the highest line number is selected. If the *ending line-no* does not exist, the first existing lower *line-no* is used.

Remarks

SHOW is used to search for desired program statement by reading BASIC object code. When a statement is found, **SHOW** converts the BASIC object code back into text and lists it. The command looks for every full and partial match of the desired program statement. For example, if you specify Swap, both Swap and Swapf are located.

Examples

```
SHOW -V /open #/
100 SHOW -V /variable =/ 500
100 SHOW /chain/ INPUT:
```

See also

LIST, SHOW, Starting and Ending Line numbers

SIZE

Synopsis

Display memory usage for current program/data.

Syntax

SIZE { -l }

Parameters

The -l option causes the sizes and names of all linked libraries to be displayed.

Remarks

SIZE is used to display the amount of memory allocated for the storage of a current program and variables. For example:

```
Code    Data  By
4376       0  menudispatch
4376       0  Total
```

Code is the number of bytes used to store the *BASIC object code* encoded program.

Data is the number of bytes used to store data for variables.

Code and Data space is displayed separately for each nested subprogram.

Examples

```
SIZE
```

See also

NEW

STATUS

Synopsis

Prints the name of current program file and execution status.

Syntax

STATUS {BREAKPOINT | MACHINE | SYSTEM | UNIT }

Parameters

{BREAKPOINT} displays each of the existing breakpoints.

{MACHINE} displays current characteristics of the current program as a whole.

{SYSTEM}displays the current directory and other system information.

{UNIT} displays current characteristics of the program unit being examined.

Remarks

STATUS is used to determine the current execution status and program filename, breakpoints, and other status interaction, which it prints. The current execution status is displayed after a breakpoint. One line is printed for each function call, procedure call, subprogram call, program **SWAP**, or **GOSUB** that is still in progress.

Most of the displayed lines take the form "xxx [n] location", where "xxx" is either "->" if this is the current examination level, as controlled by the **LEVEL** and **EXAMINE** commands, or three (3) spaces before the status indication.

The value of "n" is the stack level, starting at zero in the main program and increasing with each function call, procedure call, subprogram call, or program **SWAP**. The string "location" has one of the following forms (l = line number; s = statement number within that line, starting at 1):

l;s
local: l;s
external: l;s
pgm: l;s
external: local: l;s
lib: external: local: l;s
lib: external: l;s
pgm: l;s
pgm: local: l;s
pgm: external: l;s
pgm: external: local: l;s
pgm: lib: external: l;s

pgm is the name of the program containing the line.

lib is the name of the library containing the line.

external is an external procedure or function name.

local is a local procedure or function name.

For example:

```
-->[1] PrintHello:20;1
   [0] 40;1
```

In the above example, the program has stopped at the first statement of line 20 within a **CALL** to the procedure "PrintHello".

If a subprogram call (**CALL** "filename") or **SWAP** is in progress, then either:

" filename - CALLed"

or:

" filename - SWAPed"

is printed at the point in the stack at which the call or **SWAP** occurred:

```
--> [1] SUBPROGRAM:10;1
  SUBPROGRAM - CALLed
    [0] 50;1

--> [1] SWAPPROGRAM:10;1
  SWAPPROGRAM - SWAPed
    [0] 60;1
```

The **STATUS** command considers each program, external function, or external procedure to be a separate program unit with its own **GOSUB** stack. If any entries are present in a **GOSUB** stack to indicate that a **GOSUB** has occurred without a **RETURN** or other action to pop the stack, then the stack is displayed as in the following example:

```
-- > [1] PRINTHELLO:20;1
        GOSUB Stack:
      [0] 50;1
    [0] 40;1
```

Each line of the **GOSUB** display shows the line number and the statement number at which the **GOSUB** was executed.

Status Breakpoint

The *status breakpoint* command displays each of the existing breakpoints, using the same "pgm:lib:external:local:line;stmt" format as the *current execution status*. If error breakpoints (**BREAK IF ERROR**) are enabled, the line "Break If Error" is also printed:

```
dbg>status b
PrintHello:20;1
```

Status Machine

The *status machine* command, which displays current characteristics of the current program as a whole, is described in the EXAMPLES subsection.

Status System

The *status system* command, which displays current directory and other system information, is described in the EXAMPLES subsection.

Status Unit

The *status unit* command displays current characteristics of the program unit being examined. Each program, external function, or external procedure is a separate program unit. These characteristics are described in the EXAMPLES subsection.

Entering **STATUS** * produces a Format error.

Examples

If you enter **STATUS** without a parameter, the screen displays:

```
status
-->[0] 90;1
```

where [0] indicates level 0, 90 indicates line number, and 1 indicates statement number within the line number.

If you enter **STATUS** with the {MACHINE} parameter, the screen displays:

```
status machine
Default input channel: 102
Default output channel: 102
```

```
Trace channel: None
Command line string: “”
Hot-key program: “swap.run”
```

where the *Default input channel* is the channel number used by any non-channel **INPUT** statement.

The *Default output channel* is the channel number used by any non-channel **PRINT** statement.

The *Trace channel* is the channel number used for program tracing, if enabled by the **TRACE** statement or command.

The *Command line string* is is the command line by which the current program was invoked.

The *Hot-key program* is the name of the program, if any, to be invoked by the **SWAP** key. The program name can be set by using the **SWAPF** intrinsic procedure.

If you enter **STATUS** with the {SYSTEM} parameter, the screen displays:

```
Current directory: C:\Program Files\dL4\SAMPLES
Port number:       4094
Number of users:   2
MSC(7) value:      257
SPC(5) value:      257
SPC(7) value:      0
```

If you enter **STATUS** with the {UNIT} parameter, the screen displays:

```
status unit
Current position: [0] 90;1
Last error number: 0
Last error position: n/a
Last error text: “”
Last END or STOP: 0
DATA position: 0
Last determinant: <Not-A-Number>
Last input element: 0
Last input size: 0
Input pend mode: On
Number precision: %3
Date precision: %3
LIB directory: “”
```

status unit is a status listing.

Current position is the current execution location within the program unit. [0] 90;1 indicates level 0, line 90, statement 1 within the line.

Last error number indicates line where the last error occurred. 0 indicates there is no last error.

Last error position is not available: n/a indicates there is no last error position.

Last error text is an English phrase when available. “” means none was found.

Last END or STOP is the location, if any, of the last **END** or **STOP** statement that was executed. 0 indicates no such location exists.

DATA position is is the current **DATA** line number to be used by non-channel **READ** statements. 0 is the current value.

Last determinant is the current value of the “**DET**(0)” function, the determinant generated by the last matrix inversion statement. <*Not-A-Number*> indicates the current value is undefined: most likely no **MAT INV** statement has been executed).

Last input element is the current value of “**MSC**(1)”. In this case, the current value is 0.

Last input size is is the current value of “**SPC**(17)”. The current value is 0.

Input pend mode is either On or Off.

Number precision is the precision to be applied to any newly-created numeric variable. The current precision is %3.

Date precision is the precision to be applied to any newly-created date variable. The current precision is %3.

LIB directory is the current value of "**MSC**$(6)".

See also

BREAK, END, EXAMINE, LEVEL, XBREAK

TRACE

Synopsis

Enable statement trace debugging.

Syntax

TRACE {ON {*#channel*}| OFF}

Parameters

channel indicates the channel where **TRACE** output is to be sent.

Remarks

The **TRACE** command is used to enable **TRACE ON** and **TRACE OFF**. *Trace mode* is used to observe the line number program flow without performing single steps. dL4 provides various ways to handle tracing:

To turn Trace mode on, use **TRACE ON** or **TRACE ON** *#chn.num*.

To turn Trace mode off, use **TRACE OFF**.

The **TRACE ON** statement can be followed by an optional channel number for redirecting trace output to a file or driver. The channel number that is given must be opened prior to executing the **TRACE** statement. If the channel is subsequently closed, trace output defaults to the terminal.

Tracing is automatically disabled when another program is loaded using **CHAIN**, **SWAP**, or **SPAWN**.

Canceling a running program does not turn **TRACE** off.

Examples

```
TRACE
TRACE ON
TRACE OFF
TRACE ON #5
```

See also

VARIABLE

Synopsis

Display variable names that are defined in the currently selected procedure.

Syntax

VARIABLE

Parameters

None.

Remarks

VARIABLE displays a list of the variable names defined in the currently selected procedure.

Examples

```
VARIABLE
```

See also

LEVEL, LIST, STATUS

WB

Synopsis

Move the debug window to the bottom of the screen.

Syntax

WB

Parameters

None.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WB
```

See also

WT

WF

Synopsis

Resize the debug window to full screen

Syntax

WF

Parameters

None.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WF
```

See also

WH, WS

WH

Synopsis

Resize the debug window to one half the screen.

Syntax

WH

Parameters

None.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WH
```

See also

WF, WS

WINDOW

Synopsis

Move, resize, or change treatment of the debug window.

Syntax

WINDOW [*@x,y*] [*columns,rows*] [*rows*] [CLOSE] [HIDE]

W [*@x,y*] [*columns,rows*] [*rows*] [CLOSE] [HIDE]

Parameters

@x,y are numerics that move the upper left corner of the debug window to the *column, row* coordinates.

columns,rows are numerics that resize the debug window by number of *columns* and *rows*.

rows is a numeric that resizes the debug window by number of *rows*

CLOSE will cause the debugger to close the debug window when program execution continues.

HIDE will cause the debugger to keep the debug window open when program execution continues. This is the default state of the debug window.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WINDOW 20,70
WINDOW @0,10
```

See also

LOADSAVE, PSAVE, SAVE

WS

Synopsis

Resize the debug window to one quarter of the screen.

Syntax

WS

Parameters

None.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WS
```

See also

WF, WH

WT

Synopsis

Move the debug window to the top of the screen.

Syntax

WT

Parameters

None.

Remarks

A debug window is only used if dL4 windows are open.

Examples

```
WT
```

See also

WB

XBREAK

Synopsis

Create a breakpoint to the Debugger at a specified position or event, where processing or reception of data is to be interrupted.

Syntax

XBREAK *position*
XBREAK IF ERROR

Parameters

position is the point in the program at which processing or reception of data is to be interrupted. A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. For a full definition of a position parameter, refer to Appendix C. Breakpoints created with **XBREAK** will be applied to both the current program and to any program which is loaded by CHAIN, SWAP, or CALL statements during execution. Thus, an "XBREAK 60" command would create a breakpoint at line 60 in the current program and at line 60 in any program that was entered during execution. The position specified in the **XBREAK** command does not have to exist in the current program.

If error sets a breakpoint for the Debugger. The breakpoint occurs on error detection, before executing any program error trapping (e.g., if err 0).

Remarks

XBREAK is used to create a breakpoint for any purpose, such as examining data to allow for a display of variables and statuses.

The number of breakpoints that can be created within a program is limited only by memory available.

Examples

```
XBREAK 60
XBREAK if error
```

See also

BREAK, NOBREAK, STATUS

Chapter 5 - loadsave

loadsave encodes BASIC source code from a text file into BASIC object code which is saved as an executable dL4 program. **loadsave** enables you to develop applications outside the dL4 Command Line-oriented IDE environment.

Since **loadsave** works with text files, it lets you use the Source Code Control System (SCCS) on Unix as a maintenance and enhancement tracking tool. In addition, **loadsave** may be invoked in a make or nmake description file to maintain up-to-date versions of programs. A detailed description of SCCS and make is beyond the scope of this guide. Consult your system documentation for SCCS and make utilities.

loadsave

Synopsis

Load and save a BASIC program.

Syntax

loadsave {*switches*} *source file -o object file*

Parameters

switches are optional command line options to loadsave.

source file is a required text filename containing a valid dL4 BASIC program.

object file is the required output filename where the final encoded BASIC object code is saved.

Remarks

Option switches associated with **loadsave** are:

-h or -?	Output simple usage information..
-H	Output complete usage information..
-c *profile*	If you are converting from other versions of BASIC, you may need to use this option to convert older programs. `'profile' is the name of a 'conversion profile' used to control the conversion. Commented examples of conversion profiles can be found in the dL4 Tools directory (“convert.prf” or “convbits.prf”). Options can be set in the conversion profile to select the language dialect (IRIS, BITS, or IMS), to generate missing line numbers used by GOTO or GOSUB statements, or perform other non-standard conversions.
-C *outfile*	Specifies the text output file for the converted program (used with –c and without -o).
-e	Do not display the program source line of an error.
-i *n{,m}*	Specifies indentation for IF, DO, and other multiline structures. The number “*n*” is the number of columns to indent and the optional “m” is the initial left margin. The default values are equivalent to “-i 2,0”. This option can only be used with “-C”.
-l *n*	Create an OSN protected program. The value "*n*" is the number of a master OSN as listed by the SCOPE **OEM** command.
-L	Convert line numbers to labels.
-n *n{,i}*	Specify initial output line number used for source files that do not use line numbers. If ‘i’ is specified, the line number will be incremented by ‘i’ between each line.
-o *outfile*	Specifies the output file for the compiled program (required unless –C or –O specified).
-O *outfile*	Specifies the output file for the compiled program. Unlike “-o”, the output file will be produced even if errors are detected.
-ro	Output a run-only program (implies -s).
-s	Strip all remarks.
-t *n*	Specify the number of leading spaces that are equivalent to a tab character. This option is used with “-C” to produce an output text file with tabs replacing leading spaces wherever possible.
-u	Check program for undeclared variables.
-U	Force output of line numbers to the text file specified by the “-C” option.
-v	Output the version number of **loadsave**.
-V	Output version number of **loadsave** without any other explanatory text.
-w	Print warning messages for possible errors such as unDIMmed variables.

loadsave loads a BASIC program from a text file and saves it as a BASIC program file.

The -ro option creates a Run-only file which cannot be listed.

If the source file contains an error or does not exist, the object file is neither saved nor created. The object file is created only if the entire encoding process succeeds. If the object file already exists, it is overwritten.

In addition to the standard dL4 statements, **loadsave** supports include statement in the source text file to insert lines from other text files. To avoid placing paths in include statements, the runtime parameter INCSTRING can be used to provide a space separated search list of directories that contain include files. Example:

Include “filename”

On Unix systems, options and permissions can be added to 'outfile' to make the object file directly executable from the operating system command line. The stdexec and netexec options assume that dL4 run has been installed as /usr/bin/run. The netexec option uses '/usr/bin/run -NB' to execute the program without a terminal definition. The exec option is used to specify the run path or run options. A dL4 program file with Unix execution options can still be used in dL4 scope. On Windows systems, file associations can be configured to provide a similar direct execution feature.

Examples

```
loadsave {-s} {-c profile} -o outfile srcfile
loadsave -{vh?}
loadsave -o "<755> (stdexec) outfile" srcfile
```

Chapter 6 - run

run executes a BASIC object code file in a non-Command Line IDE environment.

The **run** session begins when the user types “run filename”.

The **run** session terminates when the program relinquishes control, or when a non-trapped error occurs during program execution.

The long chain statement is not supported for **run** because there is no System Command Line Processor (SCOPE). Long chain is described in the dL4 Language Reference Guide.

run

Synopsis

Executes BASIC programs.

Syntax

run {*option switches*} *filename* {*arguments*}

Parameters

option switches are optional command line options to run.

filename is any filename or path to a dL4 program file (not text file) to which you have read-permission.

arguments are additional information passed to the BASIC program.

Remarks

run is the BASIC interpreter used to execute a previously-saved BASIC program. The filename can be either a relative or absolute filename. The arguments are passed to the BASIC program.

Option switches associated with **run** are:

-h or -?	Output usage information.
-B	Specify binary terminal input and output.
-k *n*	Specify the socket “keepalive” interval in seconds for the standard input channel.
-N	Specify dumb terminal mode.
-t *filename*	Specify terminal definition file.
-X	Specify dynamicXport mode. This option should only be used by dynamicXport.

Examples

```
run payroll
run payroll 7/4/76
```

Chapter 7 - Tools

This chapter describes the utility programs that are supplied with dL4. These programs are dL4 BASIC programs and are installed in the Tools subdirectory.

BATCH

Synopsis

Start and execute commands on a different port.

Syntax

batch

batch *port* { *command* | ^*commandfile* }

batch /h

Parameters

port is the port number to start and execute command on.

command is a command to execute.

commandfile is the path of a text file containing commands to execute..

Remarks

The **BATCH** utility allows a user to attach an interactive or phantom port and transmit commands to that port.

The /H option displays instructions for using **BATCH**.

port is an optional port number. If *port* is not supplied on the command line, prompt mode is selected (see below). The port must be a valid port number. If an interactive dL4 session is currently running on the selected port, it is terminated to command mode. If not, a background process is created assuming the identity of the specified port number.

command is any dL4 command, such as the name of a program or command. The form ^*commandfile* instructs **BATCH** to read and transmit all of the commands in the text file to the selected port. If *command* or *commandfile* is not supplied, prompt mode is selected (see below).

BATCH is designed to operate in one of two modes - immediate and prompt. Immediate mode is assumed whenever both a *port* and *command* or *commandfile* is specified on the command line. This mode is useful when a single specific command is to be performed in background which requires no additional input.

Prompt mode is assumed when any required parameter is not supplied and **BATCH** enters a dialogue mode with the user. A port is requested if one was not supplied as part of the command line. Once the port is attached **BATCH** repeatedly prompts for entry of a command. Multiple commands, such as starting a program followed by the entry of required prompts is permitted. After successful transmission of each command, you are prompted for another. Pressing **ESCAPE** terminates entry of commands and requests a new port number for another prompt-mode session. Pressing **ESCAPE** a second time terminates **BATCH**.

Examples

```
batch 87 libr [output] ^
batch
```

BITSDIR

Synopsis

List files in a directory.

Syntax

bitsdir {*switches*}

Parameters

switches are optional command line options to **bitsdir**.

Remarks

switches are optional, and used to limit, select and control the list of filenames printed from a directory. If no *switches* are entered, all public files in the current working directory are displayed. The following switches may be entered in any order, separated by spaces:

/H	Print instructions for using **BITSDIR**. An abbreviated list of commands and their formats is displayed.
/L	Output to printer, $LPT. All output is paginated and directed to the executable script lpt.
/L=$filename	Output to device 'filename'. Select any executable pipe to direct the output. All output is paginated and directed through the pipe.
/L=filename	Create and output to a text file 'filename'.
/S	Abbreviate the information displayed using two columns. Only the filename, account, and size is displayed.
path:	Specify the pathname from which to create the directory listing. pathname must be terminated by a colon.
[GRP-USR]	List public files on the group id (GRP) and user id (USR). Public files are those which you have read or write permission. Up to 10 different [GRP-USR] selections may be entered.
[GRP-*]	List all public files for one group, any user.
[*-USR]	List all public files for one user, any group.
[*-*] or @	List all public files on any account.
/A	Alphabetize by filename. All selected files are sorted by filename.
/AA	Alphabetize by user account numbers. Files are sorted first by [GRP-USR], followed by filename.
T=type	Restrict listing to specific file types. These types are: T Tree-Structured Data Files. $ Executable device drivers, shell scripts or native OS programs. C Contiguous Data Files. I Indexed Data Files; all, whether poly or normal. B BASIC Saved Program files. S System BASIC Saved Program Files.
>X	List only those files not accessed within X hours.
<X	List only those files accessed within X hours.
<<X	List only those files created within X hours
>>X	List only those files older than X hours.
(abc*)	Restrict listing to files beginning with 'abc', such as "abc", "abcdata".
(*xyz)	Restrict listing to files ending with 'xyz', such as "xyz" and "dataxyz".
(ab*z)	Restrict listing to files beginning with 'ab' and ending with 'z'.
(*ijk*)	Restrict listing to files containing 'ijk'.

Up to 20 selections, separated by commas may be included within ( ).

.

Examples

```
dir /L=textfile @T=I (A.*, *.dat)
dir /usr/ub/1: @ /A
```

BITSTERM

Synopsis

Display or control status of active ports.

Syntax

bitsterm *{portrange}* monitor *{switches}*

bitsterm {*portrange*} evict

bitsterm /h

Parameters

portrange is a continuous range of port numbers. It can be a single port number, a range expressed in the form “first – last”, or the keyword “all” which selects all ports.

switches are options for the monitor display.

Remarks

The **bitsterm** utility has several functions controlled by the function keyword:

Monitor	display status of selected ports
Evict	terminate selected ports
-h	display usage information

The function keywords “monitor” and “evict” can be abbreviated as “m”or “e”. Case is ignored in all keywords and switchs. The supported *switches* for the **bitsterm** monitor function are:

C	repeat display every 10 seconds
F	display open channel and file information for the program running on the port

Examples

```
bitsterm all mf
bitsterm 5-12 evict
```

BUILDFI

Synopsis

Interactively create a Full-ISAM file.

Syntax

buildfi

Parameters

None.

Remarks

buildfi is a simple interactive utility that allows the user to define and create a Full-ISAM file without writing a dL4 program.

Examples

```
buildfi
```

BUILDXF

Synopsis

Interactively create an Indexed-Contiguous file.

Syntax

buildxf

Parameters

None.

Remarks

buildxf is a simple interactive utility that allows the user to define and create an Indexed-Contiguous file without writing a dL4 program.

Examples

```
Buildxf
```

CHANGE

Synopsis

Modify file permissions or attributes

Syntax

change {*switches*} {*filename*}

Parameters

switches are optional command line options to **change**

filename is the path of the file to be modied.

Remarks

Option switches associated with **change** are:

-h or -? Output usage information.

CHANGE operates in a interactive mode, displaying the current attributes and requesting new values. Press **RETURN** to move to the next prompt without changing the displayed information. To change an item, enter the new information and press **RETURN.** Press **ESCAPE** to terminate the command.

The prompt for NEW COST is printed only for UniBasic compatibility and has no affect.

Examples

```
change data/customer
```

CHECKSUM

Synopsis

Calculate or compare a file checksum.

Syntax

checksum *{switches} { {-c checksum} filename} …*

Parameters

switches are optional command line options to **checksum**.

filename is the path of a file to be checksummed. If the filename is preceded by a "-c checksum" option, then the calculated checksum is compared with "checksum" and a "Matched" or "Different" status message is displayed instead of the checksum.

Remarks

Option switches associated with **checksum** are:

-h or -?	Output usage information.
-m	Use MD5 checksum.

Examples

```
checksum –m programs/mainmenu
```

CONVBITS.PRF

Synopsis

Sample conversion profile for converting BITS program source text.

Syntax

N/A.

Parameters

None.

Remarks

The convbits.prf file is a sample conversion profile for use with the **CONVERT** command or with **LOADSAVE**. The profile was written to convert BITS BASIC source text files to dL4.

Examples

```
convert program.bas,/usr/lib/dl4/tools/convbits.prf
```

CONVERT.PRF

Synopsis

Sample conversion profile for converting IRIS program source text.

Syntax

N/A.

Parameters

None.

Remarks

The convert.prf file is a sample conversion profile for use with the **CONVERT** command or with **LOADSAVE**. The profile was written to convert IRIS BASIC source text files to dL4.

Examples

```
convert program.bas,/usr/lib/dl4/tools/convert.prf
```

COPY

Synopsis

Copy file.

Syntax

copy {<*attr*>} *destination* = *sourcefile* {,*sourcefile*} …

Parameters

attr are optional file attributes such as access permissions.

destination is the path of the destination file.

sourcefile is a path of a file to be copied to *destination*.

Remarks

The source files are copied to *destination* which is a new file to be created. If *destination* begins with a dollar sign, it will be opened with the pipe driver. If more than one source file is specified, the source files will be concatenated.

Examples

```
COPY backup/payrollbackup = data/payroll
COPY <644> programsave=program
COPY $lpt=data2
```

DOKEY

Synopsis

Examine or modify indexed contiguous files.

Syntax

dokey {*filename*}

Parameters

filename is the path of an indexed contiguous file.

Remarks

The **dokey** utility is identical to the **keymain**t utility. Please see the description of **keymaint** for a description of both **keymaint** and **dokey**.

Examples

```
dokey data/customers
```

FORMAT

Synopsis

Create a formatted or contiguous file.

Syntax

format { {<*attr*>} { [*numrecs*:*reclen*] } *filename* }

format /h

Parameters

attr are optional file attributes such as file permissions.

numrecs is the number of records in the contiguous file.

reclen is the contiguous record length in bytes.

filename is the path of the file to be created.

Remarks

Option switches associated with **format** are:

/h Output usage information.

The **format** utility is used to create formatted or contiguous files. A formatted file will be created unless the "[*numrecs*:*reclen*]" option is specified. If no optional parameters are specified, the utility will prompt for the filename and file attributes. When creating a formatted file, the utility will prompt item types and sizes in the following formats:

Sn String data where n is the length of the field. Valid lengths are greater than zero and less than 65535. For example, S20 will create a 20-byte string field.

Dn Numeric data where n is the precision to be specified. Valid precisions are 1 through 5. See the dL4 Language Reference Guide for a description of numeric precision. For example, N2 will create a 4-byte numeric field.

Bn Binary strings, where n is the length of the field in words (2 bytes per word). Valid lengths are greater than zero and less than 32768. For example, B20 will create a 40-byte binary field.

Examples

```
format data/orders
```

IC2FI

Synopsis

Convert indexed contiguous files or data to Full-ISAM files or data.

Syntax

ic2fi

Parameters

None.

Remarks

The **IC2FI** utility is an interactive program to convert indexed contiguous files to Full-ISAM files. The utility may also be used interactively or non-interactively to copy data from indexed contiguous files to an existing Full-ISAM file. Separate documentation is available that describes how to use the **IC2FI** utility.

Examples

```
ic2fi
```

KEYMAINT

Synopsis

Examine or modify indexed contiguous files.

Syntax

keymaint {*filename*}

keymant /L{=*logfile*} {*filename*}

Parameters

filename is the path of the indexed contiguous file to be accessed.

logfile is a log output filename.

Remarks

KEYMAINT is an interactive utility to access, modify, and repair indexed contiguous files. If a filename is not specified on the command line, **KEYMAINT** prompts for a filename. The commands are shown below. Many commands prompt for additional information. The prompts are shown in bold face and the responses are explained in italics.

Cmd	Name	Description
A	Add Key	Insert keys into the index currently selected.
		Enter Key to add:
		Enter the key to insert.
		Enter Record # for (key):
		Enter the record # to be associated with (KEY).
C	List Count	Displays the number of keys that were listed using the last L option.
D	Delete Key	Delete keys from the selected index.
		Enter the key you wish to delete:
		Enter the key to delete.
		(KEY) deleted, return record # (rec) to free list?
		Enter N if you do not want the record returned; any other response will return the record to the free list. The (KEY) field will display the KEY you deleted and the (rec) field displays the record number used by the KEY.
F	New File	Change from one file to another.
		Enter Filename:
		Enter a new filename in the form filename or filename-index-number.
G	Get Key	Scan the selected index (from a specified starting point) to locate a key to delete.
		Enter beginning key to delete:
		Enter the key that you wish to start the scan from. The key and associated record number are displayed.
		(D)elete, (S)can, (E)xit:
		Enter E to return to the command prompt.
		Enter S to scan up to the next key.
		Enter D to delete the key.

H	Help	Displays the help information.
I	Info on File	Recall file information for display.
L	List Index	Displays keys in the selected index.
		Enter Key to start at:
		Enter the key from which you wish to start the display. The display shows 14 keys, then responds:
		Press 'Return' to see more:
		Press Return to see the next 14 keys.
		Press ESCape to return to the command prompt.
N	New Index	Change the selected index.
		Enter index number:
		Enter the index (directory) number you wish to browse.
O	Output Data	Output up to 512 bytes of a data record as a string. All non-printable characters are displayed as ^.
		Enter Record # for (O)utput:
		Enter the Record number you want to output.
R	Read Data	Read a data record one item at a time.
		Enter key to read:
		Enter the key for the record you want to read. If you press <Return>, the response is:
		Enter the Record # to read:
		Enter a physical record number.
		Enter type (1-6=Numeric, S###=String):
		Enter a number from 1 to 6 to specify numeric precision.
		Enter S and the length for a string. String length can be up to 512 bytes.
		Enter Displacement:
		Enter the byte displacement in the data record for the item you want to read. You will then see the record number, displacement, the type, and the data item.
W	Write Data	Write a data record one item at a time.
		Enter Key to write:
		Enter the key of the record you want to write. If you press <Return>, the response is:
		Enter the Record # to write:
		Enter a physical record number.
		Enter type (1-6=Numeric, S###=String):
		Enter a number from 1 to 6 to specify numeric precision.
		Enter S and the length for a string. String length can be up to 512 bytes.
		Enter Displacement:
		Enter the byte displacement in the data record for the item you want to write.

		Enter data to write:
		Enter the data you want to write. You will then see the record number, displacement, the type, and the data item.
X	Exit	Allows you to exit KEYMAINT
Z	Get Record	Get or release records.
		(G)et or (R)elease Record
		Enter G to get a record from the free list.
		Enter R to release a record to the free list.
		If you enter G, the display is "Record number (rec) is now yours!" where (rec) is the record number removed from the free list.
		If you enter R, the display is:
		Enter Record number to release:
		Enter the record number that you want to release back to the free list.

Press **ESCAPE** to return to the previous prompt. You will move back one prompt each time you press **ESCAPE**.

The /L option can be used with any command to print the output as a log:

/L	Sends the output to the system printer '$lpt'.
/L=$file	Sends the output to a secondary printer named file.
/L=file	Sends the output to a text file named file.

Examples

```
keymaint data/customers
```

LIBR

Synopsis

List files in a directory.

Syntax

libr {*switches*}

Parameters

switches are optional command line options to **libr**.

Remarks

The *switches* shown below are optional, and used to limit, select and control the list of filenames printed from a specified directory. If no switches are entered, all public files in the current working directory are displayed. The switches may be entered in any order, separated by spaces:

@	List all accessible files for all accounts. An accessible file is any file with read permission set for the user issuing the command.
@g	List all accessible files belonging only to accounts in group g, where g is a decimal number
@g,u	List all accessible files belonging only to the account group g, and user u.
*type	Restrict listing to specific file types. Valid types are:
T	Text Files.
$	Executable device drivers, shell scripts or 'C' programs.
C	Contiguous Data Files.
I	Indexed Data Files; all, whether poly or normal.
B	BASIC Saved Program files.
S	System BASIC Saved Program Files.
F	Formatted Data File
abc	List all only files whose names begin with the characters given. For example: abc, abcc, abcd, abcz, etc.
^	Alphabetize listing by filename. All selected files are sorted by filename. Without the up-arrow option, files are listed in order of occurrence in directory.
>X	List only those files not accessed within X hours.
<X	List only those files accessed within X hours.
dir/	List files in directory dir. Only directories within the LUST environment variable will be searched.
_	Abbreviate the information displayed using only the File Type and Filename columns.
[dest]	Output the listing to either a pipe ($lpt) or a textfile.

Examples

```
libr sys/ @ *B
libr 1/ >20 <40 *I ^
```

MAKE

Synopsis

Create multiple data files with the same attributes..

Syntax

make {<*attr*>} *filename* {,*filename*} …

Parameters

attrs are optional file attributes such as file access permissions.

filename is the path of a file to be created.

Remarks

Examples

```
make <100:512CP> ABC D17 DISK1 FILE-17
```

MAKECMND

Synopsis

Generate a command file for **BATCH** or **EXEC**.

Syntax

makecmnd { *cmdfile* using *DIRfile* }

makecmnd /h

Parameters

cmdfile is a text file to be created.

DIRfile is a text file in the format produced by a “DIR /l=file” command.

Remarks

Option switches associated with **makecmnd** are:

/h	Output usage information.

MAKECMND generates a command file for use by **BATCH** or **EXEC** commands. A command file generally consists of a set of commands repeated for a number of filenames read from a **DIR** utility directory listing. If no options are present on the command line, the user is prompted for the file to create (*cmdfile*) and for the directory file (*DIRfile*).

The user is prompted to enter a series of commands to apply to each of the filenames in the *DIRfile*. Up to 20 command lines may be entered. Command lines are normally duplicated to the command file, with the following replacement options:

Characters	Replaced with (from *DIRfile* listing)
?	A filename.
? (X,Y)	Characters X through Y of a filename.
@	The account [GRP-USR]
<?>	The file's attributes.
<?+Y-Z>	Add or subtract individual letters from the file's attributes.
(SAV)	The appropriate save command for the BASIC program (SAVE or PSAVE). The DIR listing must be of the “/V” option type.

Negative subscripts can be used with the "?" character to specify a displacement from the end of the filename, for example:

?	"FILENAME"
?(,1,)	"FILENAM"
?(-3)	"NAME"

Examples

```
makecmnd cfile using dirlist
```

MAKEHUGE

Synopsis

Convert files to huge file format.

Syntax

makehuge {*switches*} {*path*} …

Parameters

switches are optional command line options to **makehuge**.

path is the path of a file or directory to be converted.

Remarks

Option switches associated with **makehuge** are:

-h or -?	Output usage information.
-d	Convert all files in specified directory.
-v	Output additional conversion status messages

Examples

```
makehuge data/history
```

MAKEUNIV

Synopsis

Convert non-portable UniBasic files to Universal files

Syntax

makeuniv {*switches*} -o *outputdir profile*

makeuniv {*switches*} -p *profile sourcedir*

makeuniv –h

makeuniv -H

makeuniv -v

Parameters

switches are optional command line options to **makeuniv**.

outputdir is the path of a directory into which converted files will be written. The files to be converted are specified in the *profile* file. If the *profile* file specifies files from different sub-directories, those sub-directories will be recreated in *outputdir*. The output directory cannot contain the source data files that are being converted.

profile is the path of a text profile file. When using the "-p" option, the *profile* file is created and written by **makeuniv**. When using the "-o" option, the *profile* file must be an existing text file using the format specified in the **Remarks** section.

sourcedir is the path of a directory containing files to be converted. The directory can contain sub-directories.

Remarks

MAKEUNIV converts non-portable UniBasic formatted, contiguous, and indexed contiguous files to Universal or Portable equivalents. Because UniBasic is not available for Windows and conversion must be performed on the system that created the files, **MAKEUNIV** is not provided in dL4 For Windows.

Conversion is normally performed in three steps:

1. Run **MAKEUNIV** with the "-p" option to generate a prototype conversion file which describes all of the UniBasic files in the specified directory and its subdirectories. Example:

    makeuniv -p ubfiles.txt datafile-directory

2. Use a text file editor to modify the prototype conversion file to add any needed record field definitions and to check for warning messages. For BCD ("Q") data files, the prototype conversion file will be produced with all of the information needed to convert the files and no changes will be needed. For non-BCD files, the user must add record field definitions using the syntax described below.

3. Run **MAKEUNIV** again but with the "-o" option and the conversion file created in steps 1 and 2. This step performs the actual conversion of the files in the source directory to Universal or Portable files in the destination directory. The utility will create the destination directory and any subdirectories if they do not already exist. Example:

    makeuniv -o newfile-directory ubfiles.txt

Because non-Universal UniBasic files are not portable, conversion must be performed on the same type of system on which the files were created. The file and directory names must not contain spaces.

One of the following mode switches must be specified when using **MAKEUNIV**:

-h	Output basic help.
-H	Output extended help.
-v	Output version number

-o dir	Build destination files in directory "dir".
-p filename	Output conversion layout profile to "filename".

When generating a conversion profile with the “-p” option (step 1), the optional switches are:

-c arg	Set contiguous and indexed contiguous file conversion options according to "arg". The only option is "allstring" which treats non-BCD file records as all string data. A record section will be output for non-BCD files defining the record as a single string. Manual editing of the record section will not be required.
-f arg	Set formatted file conversion options according to “arg". The option can either be "inttobcd" or or "extended". The "inttobcd" option converts 16-bit binary integer fields to BCD integers and may cause overflow errors during conversion. The "extended" option allows the use of binary integers and 5 word BCD floating point. An "extended" file can not be accessed by UniBasic.
-i arg	Set index conversion options according to "arg". The only option is "iriskeys" which converts IRIS or binary keys ("k" files) to ASCII strings.
-l	Use dL4 LUMAP and/or DL4LUST when evaluating file paths.
-t arg	Select files according to "arg" which can be any combination of "b" (BCD files), "n" (non-BCD files), "c"(contiguous), "f" (formatted), and "i" (indexed contiguous).

When converting files to the output directory with the “-o” option, the optional switches are:

-k	Use random key insertion algorithm.
-l	Use dL4 LUMAP and/or DL4LUST when evaluating file paths.
-r	Replace destination file.
-t arg	Select files according to "arg" which can be any combination of "b" (BCD files), "n" (non-BCD files), "c"(contiguous), "f" (formatted), and "i" (indexed contiguous).
-u	Enable "records-in-use" count (default setting).
-U	Disable "records-in-use" count.

The conversion profile is a text file and consists of sections which contain value or definition lines. Sections start with a section name enclosed in square brackets, for example: ‘[FILE]’. Value or definition lines are denoted by a keyword equal to a value, such as ‘FILE=test’. Lines beginning with a semicolon (;) are comments and blank lines are allowed. The conversion profile can viewed or modified with normal text file editors.

A ‘[FILE]’ section must exist in the conversion profile for each file to be converted. The ‘[FILE]’ section must begin with a value line 'FILE=filepath'. It may also contain optional values REPLACEFILE, RECORDORIGIN, SKIPTHISFILE, INTTOBCD, EXTENDEDTYPES, and IRISKEYSAREASCII. 'REPLACEFILE=Yes' (or ‘No’) and 'RECORDORIGIN=0' (or ‘1’). If not specified, REPLACEFILE defaults to NO and RECORDORIGIN defaults to 0. For example:

```
[File]
File=filename
ReplaceFile=Yes
RecordOrigin=1
```

The FILE value is the path of the file to be converted.

The REPLACEFILE value may be either ‘Yes’ or ‘No’ and determines the action if a file of the same name already exists in the destination directory. It is similar to the '-r' command line option but controls replacement on a file by file basis. The default value is ‘No’. The '-r' option, if used on the **MAKEUNIV** command line, will override this specification.

The RECORDORIGIN value allows the specification of byte positions, in the record definition section that follows, to begin at 0 or 1. This allows the user to think of the first byte of the record as either byte 0 or byte 1. The default value is zero.

The SKIPTHISFILE value instructs **MAKEUNIV** to process or skip the file. The value may be either 'Yes' or 'No' with a default value of 'No'.

The INTTOBCD value controls whether binary integer fields in formatted files are converted to BCD integers. The value can be either 'Yes' or 'No' with a default value of 'No'.

The EXTENDEDTYPES value controls whether binary integer and 5% floating point fields can be used in converted formatted files. The value can be either 'Yes' or 'No' with a default value of 'No'.

The IRISKEYSAREASCII value controls whether the IRIS ASCII key values in an indexed contiguous file with IRIS keys should be converted to ASCII character values. The value can be either 'Yes' or 'No' with a default value of 'No'.

If a contiguous or indexed contiguous file is being converted, a '[RECORD]' section must follow the '[FILE]' section. The '[RECORD]' section contains FIELD definition lines, one for each data field in the record. A FIELD definition line has the form 'FIELD=*parm1*,*parm2*,*parm3*{,*parm4*}'.

parm1 is an optional identifier used to document the field usage and may be omitted. If omitted a comma must precede *parm2*. **MAKEUNIV** does not use this identifier, but rather uses a count of the FIELD definitions. For example, if **MAKEUNIV** reports a problem with FIELD 3 then this refers to the third field defined in the [RECORD] section.

parm2 specifies the field starting byte position in the record. Unless RECORDORIGIN was set to one in the '[FILE]' section, the first byte position is zero.

parm3 specifies the length of the field. For string or binary fields, *parm3* is the byte count. For numeric fields, *parm3* must be the precision. The precision is entered as the mapped precision.

parm4 is used for binary and array fields. If the field is binary, enter a B for *parm4*. If the field is a numeric array, enter the DIMed value of the array for *parm4*.

The following example defines 5 fields where the fifth field is an array DIMed to 10 (11 elements):

```
[Record]
Field=Alpha1,1,24
Field=Alpha2,25,24
Field=Numeric,50,4%
Field=Binary,58,10
Field=Array,68,2%,10
```

In the example above, the field names reflect the field types, but this isn't required.

Putting it all together, the following is an example of a conversion profile for multiple files:

```
[File]
File=ub/cust.master
ReplaceFile=Yes
RecordOrigin=1
[Record]
Field=Name,1,24
Field=Addr1,25,24
Field=Addr2,49,10
Field=Zip,59,2%
Field=Binary,63,10,B
Field=L4YS,73,4%,4
.
.
.
[File]
File=ub/detail.file
[Record]
;Detail file
Field=RecNumber,0,1%
Field=OrderDate,2,5%
Field=PartNumber,6,7
```

Files with multiple record types.

If converting a file with multiple record types (an MRT file), it will be necessary to use a RECORDNUMBER or RECORDID definition in the [RECORD] section. A RECORDNUMBER or RECORDID definition, if used, must precede the FIELD definitions. A RECORDNUMBER definition is used if the field layout is dependent upon the location of the record in the file. A RECORDID definition is used if the field layout is determined by a field in the record.

Case 1 - field layout dependent upon location of the record in the file.

A file has records with either type A or type B fields as determined by the record number. Records 1 to 10 are type A fields. Records 11 to 20 are type B fields.

Record Fields

1	Field 1a	Field 2a	Field 3a
2	Field 1a	Field 2a	Field 3a
.			
.			
.			
10	Field 1a	Field 2a	Field 3a
11	Field 1b	Field 2b	Field 3b
.			
.			
.			
20	Field 1b	Field 2b	Field 3b

The RECORDNUMBER= label is followed by comma delimited parameters that specify an optional name, a byte offset which is just a placeholder, a byte length which is just a place holder, a record number that identifies the starting record number for the following field definitions, and an optional ending record number that specifies an inclusive range for the field definitions to follow.

```
[File]
File=ub/mrt_by_record_number.file
;MRT by record number file
[Record]
RecordNumber=a_Fields,0,0,1,10
Field1a=RecNumber,0,5%
Field2a=OrderDate,4,5%
Field3a=PartNumber,8,7
[Record]
RecordNumber=b_Fields,0,0,11,20
Field1b=RecNumber,0,1%
Field2b=OrderQty,2,1%
Field3b=PartNumber,4,7
```

Case 2 - field layout dependent upon the value in a field in the file.

A file has records with either type A or type B fields as determined by the value in a field.

Record Fields

1	ID	Field 1a	Field 2a	Field 3a
2	a	Field 1a	Field 2a	Field 3a
3	a	Field 1a	Field 2a	Field 3a
4	b	Field 1b	Field 2b	Field 3b
5	a	Field 1a	Field 2a	Field 3a
6	a	Field 1a	Field 2a	Field 3a
7	b	Field 1b	Field 2b	Field 3b
8	b	Field 1b	Field 2b	Field 3b

The RECORDID= label is followed by comma delimited parameters that specify an optional name, the starting byte offset of the field that identifies the record type, the byte length of the field that identifies the record type if the identifier is a string or the precision of the field that identifies the record type if the identifier is numeric, and the value of the record type identifier for the field definitions that follow. Note that if the record type identifier value is alphanumeric, it is case sensitive.

```
[File]
File=ub/mrt_field_value.file
;MRT by field value file
```

```
[Record]
RecordId=a_Fields,0,2,a
Field1a=RecNumber,2,5%
Field2a=OrderDate,6,5%
Field3a=PartNumber,10,7
[Record]
RecordId=b_Fields,0,2,b
Field1b=RecNumber,2,1%
Field2b=OrderQty,4,1%
Field3b=PartNumber,6,7
```

The file may have records where multiple fields are used to define the field layout. This is refered to as an ANDed MRT file. For example:

Record Fields

1	ID1	ID2	Field 1ab	Field 2ab	Field 3ab
2	a	b	Field 1ab	Field 2ab	Field 3ab
3	c	d	Field 1cd	Field 2cd	Field 3cd
4	c	d	Field 1cd	Field 2cd	Field 3cd
5	a	b	Field 1ab	Field 2ab	Field 3ab

In this case, the [Record] section would be

```
[Record]
RecordId=ANDed_IDField,0,2,a
RecordId=ANDed_IDField,0,2,b
Field=RecNumber,4,1%
Field=OrderQty,6,2%
Field=PartNumber,10,7
[Record]
RecordId=ANDed_IDField,0,2,c
RecordId=ANDed_IDField,0,2,d
Field=RecNumber,4,1%
Field=OrderQty,6,1%
Field=PartNumber,8,7
```

If the file also has a field layout that is not determined by the value in the defining field, a default [Record] section must be defined. This default [Record] section need only contain FIELD labels and is the last [RECORD] section of the applicable [FILE] section. For example, any of the the above sections may be followed by:

```
[Record]
Field1d=RecNumber,2,1%
Field2d=OrderQty,2,3%
Field3d=PartNumber,8,7
```

and all records without a matching value in the RECORDID fields will be defined as having the above fields.

Examples

```
makeuniv –f inttobcd –p convfiles.prf olddata
```

```
makeuniv –o newdata convfiles.prf
```

MFDEL

Synopsis

Delete multiple files.

Syntax

mfdel {*list*}

Parameters

list is a list of one or more files or options.

Remarks

The *list* consists of a series of filenames to be deleted. Special options are permitted as follows:

Convention	Explanation
@dirname@	Specify a default directory to apply to all subsequent filenames with the exception of filenames in the form dirname:filename.
^Dirfile	Extract the filenames to be deleted from DIRfile. Any @dirname@ selection is overridden for the files within the DIRfile.

Examples

```
mfdel MINE @progs@ DONM THAT file files:ZZZ
```

PGMCACHE

Synopsis

Control or examine the program cache.

Syntax

pgmcache *{switches}*

Parameters

switches are optional command line options to **pgmcache**.

Remarks

Option switches associated with **pgmcache** are:

add *filename*	Add program or library file to program cache.
delete	Delete program cache when all users have exited.
status	Display the program cache status and contents.

The **pgmcache** utility is used to control and examine the program cache. Program caching is described in the dL4 installation and configuration manuals. If no switches are specified, the utility displays the cache status.

Examples

```
pgmcache
pgmcache add standardcalls.lib
```

PORT

Synopsis

Display or control status of active ports.

Syntax

port *{portrange}* monitor

port {*portrange*} evict

Parameters

portrange is a continuous range of port numbers. It can be a single port number, a range expressed in the form “first – last”, or the keyword “all” which selects all ports.

Remarks

The **PORT** utility is similar to the **TERM** utility, however, **PORT** is a converted UniBasic utility and has fewer capabilities than the **TERM** utility.

PORT EVICT terminates each port numbers selected by *portrange*.

PORT MONITOR displays the activity of all port numbers selected by *portrange*. The letter M or the word ACTIVITY may replace the word MONITOR. Monitor mode displays the following information:

Port	The dL4 port number.
Group	The group number a particular user is assigned to.
User	The user number a particular user is assigned to.
Processor	The process running which is always dL4.
Program	The program running under dL4. If a port is at command mode or at SCOPE, the display may be empty for that port's program.

Examples

```
port all monitor
port 5-12 evict
```

QUERY

Synopsis

Display file status.

Syntax

query {*switches*} *filename {*as *drivername}*

Parameters

switches are options as described in the Remarks section. Each switch must be preceded by a dash character.

filename is the path of the file to be examined.

drivername is the name of the driver to be used with *filename*. This parameter should only be used when the driver cannot be determined from the filename (for example, with SQL tables).

Remarks

Option switches associated with **query** are:

-k	Scan the indexes of an indexed contiguous file and report the number of keys in each index.
-l	Sends the output to the system printer '$lpt'.
-l=$file	Sends the output to a secondary printer named ‘file’.
-l=file	Sends the output to a text file named ‘file’.
-p	Pages the screen output.
-s	Output dL4 DEF STRUCT and MEMBER statements for the record and field definitions of a Full-ISAM file. This option can be used with the “-l=file” option to generate a dL4 source file.

The query utility displays file status information such as file type and size. The information displayed is dependent on the file type.

Examples

```
query data/customers
query –l=$someprinter customers as MySQL Full-ISAM
```

SCAN

Synopsis

Obtain detailed information about a file.

Syntax

scan {*switches*} {*directory*} {*filename* | *DIRfile*} …

Parameters

switches are options as described in the Remarks section.

directory is a directory to be used with all subsequent filenames.

filename is a file to be examined.

DIRfile is a DIR utility style list of filenames.

Remarks

If no switches or filenames are entered on the command line, the user is prompted for the file to be examined. Press **RETURN** to terminate this method of operation. Switches and options may be used to affect the operation as follows:

Option	Meaning
/H	Output instructions for using SCAN.
/L=$name	Re-direct all output to the pipe driver as $name.
/L=filename	Re-direct all output to filename as a text file.
packname	Specify the packname (directory) to be searched for all subsequent filenames. This option may be used to simplify command input when a number of filenames on the same pathname are to be scanned.
filename	A specific filename to obtain detailed information for.
^DIRfile	A list of filenames, created by the **DIR** utility to obtain detailed information for. Each filename within the **DIR** output file is scanned.

Examples

```
scan icfile
```

TERM

Synopsis

Display or control status of active ports.

Syntax

term *{portrange}* monitor *{switches}*

term {*portrange*} evict

term {*portrange*} dump

term -h

Parameters

portrange is a continuous range of port numbers. It can be a single port number, a range expressed in the form “first – last”, or the keyword “all” which selects all ports.

switches are options for the monitor display.

Remarks

The **term** utility has several functions controlled by the function keyword:

Monitor	display status of selected ports
Evict	terminate selected ports
Dump	trigger port dump on selected ports. This function will have no effect on ports that do not have the DL4PORTDUMP runtime parameter defined. See the description of CALL FORCEPORTDUMP in the dL4 Language Reference Guide for more information.
-h	display usage information

The function keywords “monitor”, “evict”, and “dump” can be abbreviated as “m”, “e”, or “d”. Case is ignored in all keywords and switches. The supported *switches* for the **term** monitor function are:

B	Display only those ports that are blocked waiting for a record lock to be released. If known, the port number that is currently locking the desired record is displayed.
C	repeat display every 10 seconds
F	For each selected and active port, display the open channel numbers, the filename of the file open on the channel, and the current record number (if available). The current record number is followed by the letter “U” if the record is unlocked, “L” if the record is locked, and “B” if the port is waiting because the record has been locked by another port.
L	display the current line number of the program running on the port
P	page the display

Examples

```
term all ml
term 5-12 evict
term 22 dump
```

TESTLOCK

Synopsis

Test if record locking works on a file system.

Syntax

testlock {*file offset length*} …

testlock -h

Parameters

file is the path of an existing file on the file system to test.

offset is the byte offset at which to apply a record lock (typically 0).

length is the number of bytes to lock in the file at the specified offset.

Remarks

The **testlock** utility is normally used to check if record locking works on a remote file system such as NFS.

Examples

```
testlock /networkfs/accounting/customers 0 100
```

VERINDEX

Synopsis

Check indexed contiguous file integrity.

Syntax

verindex

Parameters

None.

Remarks

The verindex utility examines the index portion of an indexed-contiguous file and attempts to detect corrupted index blocks. The utility is run interactively and can examine either individual files or all files within a specified directory.

Examples

```
verindex
```

WHO

Synopsis

Display information about the your port.

Syntax

who

Parameters

None.

Remarks

The **WHO** utility displays the following information about your dL4 process:

Port	The UniBasic port number
CPU Secs	(not used)
Connect	The dL4 session time in hours and minutes
Time	System date and time
Disk	(not used)
User	User and Group Number
Default	The current working directory name
Total Used	(not used)
Limit	(not used)
Left	(not used).

Examples

```
who
```

Appendix A - Glossary

This glossary defines terms in the context of dL4:

absolute pathname	the full pathname, starting at the root.
BASIC object code	SEE **object code**.
block	one or more statements treated as though they were a single statement.
channel	a communication method between an application and a dL4 driver for requesting specific file operations.
character	a representation of a letter, number, or other special data representation.
character code	a numeric value that represents a particular character in a set, such as the ASCII character set.
character data type	a representation of a letter, number, or other special data representation.
character set	a mapping of characters to their identifying numeric values.
context	SEE **runtime context**.
c-tree	a keyed file structure developed by Faircom, Inc. and used by dL4.
driver	a dL4 driver acts as a translator converting a generic file operation request from an application program into a specific command that carries out the requested operation.
executable	a program that is ready for execution.
file	a collection of records.
index	a mechanism of locating data.
infinite loop	the never-ending repetition of a block of dL4 statements.
interface	SEE **port**.
ISAM files	ISAM (Indexed Sequential Access Method) is a storage and retrieval system that allows efficient access to data records using key values.
key values	identifying values used in a file to describe and locate a desired record.
keyword	a **reserved word** used as part of dL4 syntax.
loop	the repeated circular execution of one or more statements**.**
member	each individual data type in a structure data type. See **structure data type**.
nested loop	a loop within a loop.
object code	a translation, not readable to the user, of a program source code that can be directly executed by the computer.
OSN	OEM Security Number.
phantom port	a port that does not have access to its display device. Typically it runs in background.
portable	capable of being ported to different systems.
position parameter	A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. SEE Appendix C for description of **position parameter**.
program	a set of executable instructions.
relative pathname	a partial pathname relative to your current working directory.
record	a set of related fields.
reserved word	in dL4, a word that has a fixed function and cannot be used for any other purpose. Same as **keyword**.
root	the root directory, which is the main directory that contains everything on the disk.
run time	related to the events that occur while a program is being executed.
runtime context	a machine state when a dL4 program is executed.
SCCS	Source Code Control System (SCCS) is a Unix utility that allows source code level revision control for a project.
source code	a user-readable text file containing dL4 BASIC language statements.
step into	trace inside a function.
step through	execute a function but do not trace inside a function. Trace resumes outside the function.
string	a sequence of alphanumeric characters. dL4 converts all strings to Unicode characters.
structure data type	a data type that organizes different data types so that they can be referenced as a single unit. Typically, used to define a record in a data file.
subscript	a number inside brackets that differentiates one element of an array from another**.**
UniBasic	a state-of-the-art Business BASIC language, matched with the UNIX operating system; developed and marketed by Dynamic Concepts, Inc.
Unicode	a 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard.

Appendix B - dL4 Command Summary

COMMAND	SCOPE	BASIC	Debugger
!	●	●	●
;			●
.		●	●
..		●	●
AUTO		●	
BASIC	●		
BREAK		●	●
BYE	●		
CANCEL		●	
CD	●		
CHECK		●	
CLU	●		
CONTINUE		●	●
CONVERT		●	
DELETE		●	
DISPLAY		●	●
DRIVERS	●		
DUMP		●	●
EDIT		●	
END			●
EXAMINE		●	●
EXIT		●	
FILE		●	●
FIND		●	●
GO		●	●
HALT	●		
HELP		●	●
KILL	●		
LABEL		●	
LEVEL			●
LIST		●	●
LOAD		●	
NEW		●	
NOBREAK		●	●
PACK	●		
RETURN			●
RENUMBER		●	
RUN	●	●	
SAVE	●	●	
SHOW		●	●
SIZE		●	●
STATUS		●	●
TIME	●		
TRACE		●	●
USERS	●		
VARIABLE		●	●
XBREAK		●	●

Appendix C - Position Parameter

A position parameter is used by some BASIC/Debugger commands to specify a line in a dL4 program. The line may be in the main program or in a library that is linked with the main program. A position parameter should be in one of the following formats:

line-number
local-procedure-name
external-procedure-name
library-name
program-name
external-procedure-name:local-procedure-name
library-name:line-number
library-name:local-procedure-name
library-name:external-procedure-name
library-name:external-procedure-name:local-procedure-name
program-name:line-number
program-name:local-procedure-name
program-name:external-procedure-name
program-name:external-procedure-name:local-procedure-name
external-procedure-name::
library-name::
program-name::
library-name:::
program-name:::
library-name:::line-number
program-name:::line-number

Any format without a line-number will select the first executable line of the procedure, library, or program. The more complex formats are used to distinguish between procedures, libraries, and programs that have the same name. A "position" parameter is interpreted relative to the current BASIC or Debugger view of the program and so the meaning of the simple formats can vary.

The formats "program-name:::line-number" and "library-name:::line-number" can be used to avoid any ambiguity.

Index