Unibasic

From Dynamic Concepts Wiki

Contents

UniBasic

uniBasic Reference Guide

Revision 9.3

This document is intended for users of UniBasic IRIS or UniBasic BITS.

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 guide may be reproduced in any way, shape or form, for any purpose, without the express written consent of DCI.

UniBasic is a trademark of Dynamic Concepts Inc.

dL4 is a trademark of Dynamic Concepts Inc.

Dynamic Windows is a trademark of Dynamic Concepts Inc.

BITS is a trademark of Dynamic Concepts Inc.

IRIS is a trademark of Point 4 Data Corporation.

About this Guide

This guide is written for experienced BASIC programmers. It is a reference that includes a brief introduction to UniBasic and information on files and file handling, UniBasic commands, statements, calls and utilities. If you need elementary information about programming in BASIC, please refer to one of the many books available on that subject.

The terms and conventions used for demonstrating commands and BASIC statements in this guide provide a consistent format.

This guide covers UniBasic version 9 and greater.

Conventions

Literal elements of a UniBasic command, utility, statement, and unix command, utility or shell Environment Variable are shown in bold type.

Metalinguistic variables are shown in italic type for clarity and to distinguish them from elements of the language itself.

OPEN *# channel expression ; filename string*

Mono-spaced type is used to display screen output and keyboard input commands and program examples.

```
LIBR [$LPT]
```

The right and left brace characters ({optional items}) indicate an item that is optional.

```
LIST {-v}
```

A series of three periods (...) indicates that the preceding item can be repeated as many times as desired.

```
KILL filename {filename...}
```

```
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.
```

WINDOW (ON | OFF)

This guide has been grouped into topical sections. Whenever a topic or function of another

section is referenced, that topic is followed by a **See also:** reference for the section where it may be found.

For example:

> When **OPEN** is used to access a data file ...
>
> **See also:** OPEN

In this example, a reference to the UniBasic language element **OPEN** informs the reader to find the complete text of **OPEN** by using the index.

When the information may be found in documentation other than this guide, for example:

> To relocate the file, issue the Unix **cp** command. ...

the sentence includes a descriptor identifying the command and other documentation to reference. In this example, the user is referred to the Unix documentation.

Installation & Configuration

The installation of UniBasic under Unix is an interactive process. Upon completion, UniBasic and other supplied C utilities are placed into the directory /usr/bin. A master directory ub is created with a system logical unit, containing DCI supplied drivers ($LPT) and system processors (BUILDXF, QUERY, MAKEIN, etc.). Following installation, any user familiar with the IRIS or BITS systems can operate UniBasic and feel quite comfortable. Before you can convert an existing end-user, or install a new system, you will require more Unix knowledge than is provided in this guide.

Configuring Unix for UniBasic

Prior to installation for an end-user, several Unix system parameters may require re-configuration for multi-user operation. This process varies from system to system. When purchasing from a UniBasic Distributor, inquire whether these parameters have been pre-configured for your needs. If changes are required, most systems include a system administrator shell to assist you in necessary reconfiguration. For specific information, contact your manufacturer or distributor before changing any system parameters.

The group ID and user ID must be less than 65536.

Number of Processes

Each program or command, including login (getty) or a copy of UniBasic, is called a process. Unix maintains a table of all active processes on the system. The UniBasic statements **SWAP, SPAWN,** and **CALL 98** (phantom port operations) initiate additional processes. Opening a printer may invoke as many as 5 processes temporarily. If the maximum number of system processes is exceeded, an error message may be reported to the console (such as NO MORE PROCESSES for SCO-Unix systems), or UniBasic may generate a negative (system) BASIC error to the

application.

To accommodate Windows, **SWAP, SPAWN** and print jobs, set the number of processes no less than the number of users * 5. Applications that provide linkage to other Unix applications (such as IQ, Word Processing, etc) may require additional processes per user. The current processes may be displayed using the Unix **ps -ef** command.

Number of Open Files

Unix maintains tables for all opened files on the system. Each process requires a minimum of three (3) channels referred to as: standard input, standard output and standard error. In addition, a process may require additional channels if other files or devices are opened for access. UniBasic itself is an example of a process under Unix. For each additional concurrent process, an additional (3) channels minimum are required.

UniBasic requires a total of 4 channels per user process. These include the standard (3), plus one for the error message file (**ERRMESSAGE**). In addition, each device or data file opened requires one system channel; Indexed files require 2 channels.

See Also: Indexed Data Files

When the configured number of system-wide channels is exceeded, an error message may be reported to the console (such as NFILE for SCO-Unix systems), or the program may generate a negative (system) BASIC error.

To compute the approximate number of channels required for your system, multiply the Number of Processes * 3 to yield the minimum number of channels. Add to that result the average number of opened channels per-user times the number users. Remember to count each Indexed file as two channels, and include provisions for other applications, such as IQ.

Example: An 8-port system with 10 open files per user and 50 processes, might require 300 open files.

Number of Open i-nodes

Unix maintains a table of opened inodes (or header blocks). Each unique file or device opened requires one entry. Ten users accessing the same file typically share the same open i-node.

Number of Locks

Unix maintains a table of read and write locks placed on files by individual processes. Each locked region requires one entry in this table. A locked UniBasic data file record is an example of an entry in the lock table. Indexed file key maintenance temporarily requires several locks for the various levels in the ISAM tree structure. A minimum of 5 locks plus 1 lock per open file per process should be adequate for most installations.

Message Queues

For all inter-process communication, UniBasic relies on Unix message queues. Each DCI product creates a message queue at startup to transmit and receive data between users. Such messages include:

- SIGNAL 1 & 2 and SEND/RECV data between ports
- CALL 98 and PORT statement commands and status
- PORT ALL MONITOR status requests
- CALL $INPBUF type-ahead data returned to parent process
- MSG command text
- Security communications

On most systems, the Unix command **ipcs** may be used to display information about message queues. Each message queue is identified by a unique 32-bit number, usually displayed as an 8-character hexadecimal value.

DCI products are identified by our own numbering sequence, which when viewed in hexadecimal, take on an appearance such as DC00pnnn. The digits correspond to:

DC	Dynamic Concepts Product
00	Always zero
p	DCI Product ID:
0	Passport daemon
1	UniBasic IRIS
4	IQ
5	dL4
nnn	UniBasic port number, in hexadecimal, associated with this queue. For example, port 15 is displayed as "00F".

See also: Terminating a UniBasic process

Message queue requirements for UniBasic are based on the number of concurrent users and overall message traffic on the system. The default values on many systems are sufficient to support a few users, but certainly will need to be increased for large installations. If they are not configured, UniBasic may fail at start-up, possibly with a message such as "Bad system call."

The following 7 parameters affect message queues on most systems. The actual parameter names may vary:

MSGMNI Maximum number of message queues. Configure based upon the maximum number of concurrent UniBasic users plus phantom ports plus other DCI products such as IQ for Unix users plus one for the passport security daemon.

MSGMAX Maximum size of a message in bytes; at least 516.

MSGMNB Maximum number of bytes per message queue. Set to the maximum allowable

value; typically 32768.

MSGTQL Maximum number of outstanding system wide messages. Suggested setting is at least 256, but may be adjusted if message activity is known to be greater or smaller.

MSGSSZ Size (in bytes) of a message segment. Memory for message data is divided into segments of the defined size. A value of 32 is recommended.

MSGSEG Number of message segments within the system. MSGSEG * MSGSSZ determines the total number of bytes reserved for message data. The recommended formula is MSGSEG = (MSGTQL * 512)/MSGSSZ. For 256 UniBasic concurrent messages, the value would be: (256 * 512) / 32 = 4096.

MSGMAP Number of entries in the message map table. Each entry represents a contiguous free area in the message segments. The recommended formula is MSGMAP = MSGSEG/8 which, using our example, would be 512. If UniBasic reports "Communication buffer is full" when the actual number of outstanding messages is < MSGTQL, first increase MSGMAP. If that doesn't correct the error, increase MSGSEG.

AIX Note: There are no user-configurable message queue parameters on AIX. The parameters are hard-coded in the kernel, and seem adequate for most installations.

The following points must considered during configuration:

- Free message space must be available on the system. If the queues become full, additional users, including phantom ports, cannot be launched into UniBasic or IQ. In addition, existing users may be prevented from performing **SWAP, SPAWN, CALL 98** and **PORT** statements, as well as commands such as **PORT ALL MONITOR**.
- A processes queue and any waiting messages are deleted if and when the port exits normally. If a process is killed, it cannot delete its queued messages.
- The configuration guidelines shown above consider only UniBasic requirements. They do not include requirements of other Unix applications which rely on message queues.

Unix Accounting & Protection System

Access to Unix files is regulated by file permissions. Permissions are generally *read, write*, and *execute* (other permissions and attributes exist, but are not important for discussion here). These permissions are applied against three levels: The owner/creator of the file, other users in the same group as the creator, and other users in different groups. The permissions are either expressed as letters (rwx) or numbers (4 2 1) added together. When expressed as letters, a nine-character field represents the three levels; numbers are shown as three digits.

Each user gains access to the system through a *login user name* which is assigned to a user

number; the *user id*. Normally, no two users share the same *login user name* or *user id*. Each *user id* belongs to a *group*. Group numbers are equated to names in the Unix system file */etc/group*, and *user id* numbers are equated to *login id's* in the file */etc/passwd*.

Creating a Unix Account for UniBasic

Prior to installation, a master (manager) account must be created to own the UniBasic distribution files, programs and directories. Most systems supply a menu-driven administration program to assist with user account management. Please refer to the System Administrator's Guide included with your operating system. Before proceeding, please ensure that the following is completed; bracketed information is user-selectable:

- Create a login id, [UniBasic], belonging to a new group, [UniBasic], with its own home directory, [/usr/ub].

See also: Configuring a UniBasic Environment

UniBasic Security & Licensing

You may select either Hardware or Software licensing (security) for an installation of DCI software. Both are controlled by the daemon, **/etc/passport**, which is automatically launched by UniBasic. Whereas Software licensing is based upon information derived during installation, Hardware licensing is based upon the external DCI Passport™ device. Passport is not part of UniBasic and must be installed separately.

In either case, each UniBasic installation is identified by a unique 32-bit license number, generated by the Passport daemon. This license number, along with a DCI supplied Software Selection Number (SSN) activates your installation for various DCI products and configurations.

A license number is expressed as an 8-character hexadecimal value, such as 99D04832. The first two characters represent a specific operating system and/or hardware platform, in this example 99 = SCO Unix, for which the license is granted. Licenses are not transferable to other platforms.

A special directory, /etc/DCI, is created during installation to maintain security specific information and files for use by all Dynamic Concepts software products. Typically, the following text files are recorded within the directory:

- *ssn* DCI activation key for this installation.
- *osn* OEM activation keys enabling encrypted application software.
- *passport.cmd* Command text used to initiate the Passport daemon.
- *passport.log* Log file maintained by the Passport daemon with security information, licensing methods and errors.

If the installation utilizes software security, one additional binary file is created:

- *license* License number information for systems installed with Software licensing.

Warning: Modification, deletion, renaming or moving the *license* file will possibly deactivate a software license number.

Configuration of licensing is performed during installation of Passport, or by later usage of the **ppconfig** utility (refer to the Passport User's Guide for more information) .

Software Licensing

Software licensing is based upon information derived from the system by the **/etc/passport** daemon. When launched for the first time, a license file, */etc/DCI/license* is created by the daemon to record the unique license number for this installation. Although several types of software licensing methods are supported, the type is fixed by DCI for each specific hardware and operating system platform. The actual type used on a system is recorded in the Passport Log file.

The unique 32-bit license may change due to any number of conditions, including, but not limited to, any of the following:

- Replacement of a disk drive and/or restoration of all data
- Upgrade and/or replacement of the operating system
- Disturbing the */etc/DCI/license* file
- Replacement of a CPU board or network interfaces

Should your system lose it's license, a new license number will be generated automatically. Contact your supplier with your old and new license numbers for a replacement.

Hardware Licensing

Hardware licensing is a older licensing mechanism based upon the connection of a Passport device to an unused serial RS232 communication channel on the computer. Each Passport device is pre-programmed with its own unique 32-bit license number and any given SSN for that license number is perpetual. The Passport device and associated SSN may be installed on another like platform at any time.

For information concerning physical Passport installation and testing, please refer to the documentation supplied with the device.

Loading the Installation File

A UniBasic installation file is normally supplied as a compressed cpio archive file. The installation file can be downloaded from www.unibasic.com or the /dist/pub directory of

ftp.unibasic.com.. If an installation file is first downloaded on a PC and then copied to a server, be certain to perform a binary transfer of the file. The file is named using the format *pp*-ub-*vvvv.Z* or *pp*-ubdev-*vvvv.Z* where "*pp*" is the platform code (such as "99" for SCO OpenServer 5) and "*vvvv*" is the version number (such as "8.1").

For example, the AIX installation file for UniBasic 8.1 is named 07_ub_8.1.Z. After signing on as root and copying the installation file to */tmp*, the commands to load this distribution would be:

```
cd /tmp

uncompress 07_ub_8.1.Z

cpio -iavcdu <07_ub_8.1
```

On some systems, particularly Linux systems, the cpio options will have to be changed to omit the "c" option:

```
cpio -iavdu <6D_ub_8.1
```

If the command is successful, a list of filenames is displayed as the data is loaded into the */tmp* directory.

Loading the UniBasic Installation File

Verify that you are signed on as *root* and defaulted to the /tmp directory. Issue the following commands to load the installation file:

```
uncompress filename.Z (if the filename ends with a ".Z")
cpio -iavcdu < filename (i.e. 99_ub_8.1, 6D_ub_8.1.4, etc.)
```

A list of filenames similar to the following should be printed:

ub	ub/sys/buildxf	ub/irislist
ub/loadlu	ub/sys/copy	ub/license.txt
ub/makesp	ub/sys/dokey	ub/email.mail
ub/sys/batch	ub/sys/keymaint	ub/email.sendmail
ub/sys/clk	ub/sys/lpt.bits	ub/sys/change
ub/sys/dir1	ub/sys/make	ub/sys/dir
ub/sys/format	ub/sys/makeitem	ub/sys/dsp
ub/sys/libr	ub/sys/pdp	ub/sys/kill
ub/sys/lpt.sample	ub/sys/query	ub/sys/lpt.iris
ub/sys/makein	ub/sys/term.ansi	ub/sys/makecmnd
ub/sys/port	ub/sys/term.wyse60	ub/sys/mfdel
ub/sys/term	ub/ubconvert	ub/sys/pdphelp
ub/sys/term.wyse50	ub/ubconvertfiles	ub/sys/scan
ub/ubcompress	ub/ubterm	ub/sys/term.tvi925

ub/ubrebuild	ub/ubtestlock	ub/sys/who
ubinstall	ub/errmessage	ub/ubkill
ub/README	ub/sys	ub/unibasic
ub/lptfilter	ub/sys/attr	ub/sys/term.linux

Loading the UniBasic Development File

Verify that you are signed on as root and defaulted to the /tmp directory. Issue the following commands to load the installation file:

```
uncompress filename.Z (if the filename ends with a ".Z")
cpio -iavcdu < filename (i.e. 99_ubdev_8.1, 6D_ubdev_8.1.4, etc.)
```

A list of filenames similar to the following should be printed:

license.txt	ubdev/unibasic.o	ubinstall
ubdev	ubdev/var.h	ubdev/Release.h
ubdev/call1.c	ubdev/Makefile	ubdev/call105.c
ubdev/call11.c	ubdev/call10.c	ubdev/call120.c
ubdev/call121.c	ubdev/call114.c	ubdev/call123.c
ubdev/call126.c	ubdev/call122.c	ubdev/call18.c
ubdev/call19.c	ubdev/call15.c	ubdev/call20.c
ubdev/call21.c	ubdev/call2.c	ubdev/call23.c
ubdev/call24.c	ubdev/call22.c	ubdev/call27.c
ubdev/call28.c	ubdev/call25.c	ubdev/call3.c
ubdev/call30.c	ubdev/call29.c	ubdev/call44.c
ubdev/call45.c	ubdev/call43.c	ubdev/call47.c
ubdev/call48.c	ubdev/call46.c	ubdev/call5.c
ubdev/call51.c	ubdev/call49.c	ubdev/call56.c
ubdev/call57.c	ubdev/call53.c	ubdev/call60.c
ubdev/call65.c	ubdev/call59.c	ubdev/call7.c
ubdev/call72.c	ubdev/call68.c	ubdev/call76.c
ubdev/call77.c	ubdev/call73.c	ubdev/call81.c
ubdev/call82.c	ubdev/call78.c	ubdev/call96.c
ubdev/call97.c	ubdev/call88.c	ubdev/callavport.c
ubdev/callcimi.c	ubdev/call99.c	ubdev/calldate.c
ubdev/callenv.c	ubdev/callclu.c	ubdev/calldev.c
ubdev/callhelp.c	ubdev/callinpbuf.c	ubdev/callmemcmp.c
ubdev/callphil.c	ubdev/callrpcs.c	ubdev/callswapf.c
ubdev/calltrack.c	ubdev/callwindow.c	ubdev/callwlock.c

ubdev/comm	ubdev/comm/comm.h	ubdev/crt.h
ubdev/ctree	ubdev/comm/libcomm.a	ubdev/ctree/ctport.h
ubdev/decode.h	ubdev/ctree/ctifil.h	ubdev/extern.h
ubdev/dl4.h	ubdev/eval.h	ubdev/misc.h
ubdev/files.h	ubdev/math.h	ubdev/runtime.h
ubdev/pcode.h	ubdev/read_me	ubdev/term101.h
ubdev/pdn.c	ubdev/term0.c	ubdev/ubdef1.h
ubdev/str.h	ubdev/ubdef.h	ubdev/ubport.o
ubdev/timer.h	ubdev/ubdefs.h	ubdev/usercalls.c
ubdev/ubdef2.h	ubdev/unix.h	

ubinstall - Installing UniBasic Packages

ubinstall is a shell-script designed to run under the borne shell only. If the command does not execute immediately, enter the command: **chmod 500 ubinstall** and try starting **./ubinstall** again. If ubinstall still fails to begin operation, verify that you are running under the borne shell (usually the file /bin/sh). You can usually start a borne shell by typing **/bin/sh**.

If a license has not already been installed, Passport should be installed on the system before installing UniBasic. Passport is not included in the UniBasic installation. Please see the Passport User's Guide for information on installing Passport.

After the desired distribution media is loaded, enter the command:

```
./ubinstall
```

ubinstall will display the following:

```
Installation for "UniBasic" BITS/IRIS Business BASIC emulation

All Rights Reserved. Copyright (C) 1987 - 2015 by:

Dynamic Concepts Inc. Irvine, California USA

Installing the following packages:
```

ubinstall will locate all packages loaded for installation. Your display should include one or more of the following packages:

```
    UniBasic BITS/IRIS Business BASIC emulator

    UniBasic Development

Do you wish to continue? (Yes or No, default = Yes)
```

If this is a re-installation, **ubinstall** checks the revision of UniBasic currently installed in /usr/bin:

```
Checking old UniBasic... Level = 7.2

Checking new UniBasic... Level = 8.1

"/usr/bin/unibasic" already exists. If you install this version, the current version
```

```
will be renamed and saved as "/usr/bin/ub7.2".

Do you wish to continue? (Yes or No, default = Yes)
```

A response of NO terminates the installation process. */tmp* will still contain the installation files. All existing UniBasic files and data are unchanged. You may initiate the **ubinstall** operation at a later time without reloading the media. Many systems, however, remove the files in the */tmp* directory whenever the system is shutdown and subsequently restarted.

The next phase assumes you have previously created an account to own the UniBasic distribution files. This master account is the group manager of the UniBasic group, and the owner of the **HOME** directory and *sys* directories (Logical Unit 0) inclusive of all files. Additional utilities placed into the /usr/bin directory are also owned by UniBasic.

```
Part II) Accounting Information

UniBasic is distributed with a set of system utilities, an error message file, sample
terminal drivers, printer scripts, etc. These files have permissions making them
generally accessible to all users, but are installed into the user and group you
select.

Enter the user name to receive the distribution files: (default = "unibasic")
```

Enter the user name previously created as the UniBasic group manager.

```
Part III) System directory

The system directory is where the distribution files are placed and where the .profile
for UniBasic is created or modified. It is normally placed in your account's HOME
directory. Other logical units required by your application are best placed in HOME
also, unless they should be elsewhere for security or space reasons. The default HOME
for new installations is "/usr/ub".

However, the choice of "/usr/ub" is only a default; any directory name on any file
system can contain UniBasic logical units, subject to access permissions.

Enter directory to contain system files: (default = "/usr/ub")
```

If this is a re-installation to the same **HOME** directory (/usr/ub in this case), a warning similar to the following is printed to avoid overwriting any files or programs normally supplied by DCI that may have been customized by you:

```
Note: "/usr/ub/sys" already exists.
```

```
Installing will overwrite the following files:
```

attr	dokey	lpt.bits	mfdel	term.tvi925
batch	dsp	lpt.iris	pdp	term.wyse50
buildxf	email.mail	lpt.iris.sco	pdphlp	term.wyse60
change	email.sendmail	lpt.sample	port	who
clk	format	make	query	
copy	keymaint	makecmnd	scan	
dir	kill	makein	term	

dir1	libr	makeitem	term.ansi	

If you have made custom modifications to any of these files, you may want to abort the installation at this point and make copies. Otherwise, you can continue and update them to the latest revision.

Do you wish to continue? (Yes or No, default = Yes)

A response of NO terminates the installation process. */tmp* will still contain the installation files. All existing UniBasic files and data are unchanged. You may initiate the **ubinstall** operation at a later time without reloading the media. Many systems, however, remove the files in the */tmp* directory whenever the system is shutdown and subsequently restarted.

Part IV) Run-time options

Several options in UniBasic are configurable through use of "environment variables". These are generally set up in the file ".profile" in your HOME directory, and are also changeable on-demand from the Unix shell. None are required to be set up; defaults are used if not specified.

Variable	Description
BASICMODE	Specifies the operating environment for UniBasic. I=IRIS, B=BITS. (default = IRIS).
	Select the default emulation mode for users. IRIS mode provides for complete emulation of IRIS commands, syntax and visual operation. CTRL+C, Scope mode and Basic modes are enabled. Selecting BITS mode still permits execution and programming of IRIS applications, however command formats are BITS style.
SPC5	Value to be returned by SPC 5 (account number): (default = 65535)
	Choose the value to be returned to your programs for this user whenever **SPC 5** function is performed. Since the Unix group, user and protection scheme is numerically different, you are permitted to specify this value rather than have to create a special Unix account number to return your desired value. When different users require different **SPC 5** values, the system is easily changed to test who signed on, and set a different value.
DATESEP	Character used to separate MM/DD/YY strings: (default = "/")
	Choose the normal date separator used by your applications.
CURRENCY	Character used to replace $ in PRINT USING statements:(default = "$")
	Select an alternate currency character to be replaced when $ is used in **USING** formats.
WINDOWS	Maximum numbers of windows open per user. (default = "20")
	If your application uses Dynamic Windows, enter the maximum number of opened windows permitted for each user.
EUROPEAN	Mode for date verification calls (CALL 24, 27, 28). 0 = MM/DD/YY, 1 = DD/MM/YY. (default = "0")
	For European dates: 31/12/88, choose option 1

You may tailor these Environment Variables as well as a number of other configuration options by later editing the file **HOME/.profile**. For further information on configuration parameters, refer to Configuring a UniBasic Environment.

```
Do you wish to automatically run UniBasic after login? (y/n) (default = "n") y
```

This configures an automatic launching of UniBasic whenever signing onto an account that executes this standard **HOME/.profile**. You can also specify a BASIC program to start by editing the last line of the .profile script.

See also:: Launching UniBasic from Unix.

```
Installation started: Mon Mar 5 17:42:08 PST 1990

  Creating directory "/usr/ub/sys"...Done

  Installing configuration options in "/usr/ub/.profile"...Done

************************************************************

DCI strongly encourages usage of BCD file types for future file

compatibility and portability. Please refer to UB Reference Guide

for details on PREALLOCATE environment variable values.

************************************************************
```

Note: If this is a new installation, the environment variable **PREALLOCATE** is set to 32 by the **ubinstall** program.

```
  Installing UniBasic in "/usr/bin"...Done

  Installing distribution files in "/usr/ub/sys"...Done

  Creating directory "/usr/ub/ubdev"...Done

  Installing development source files in "/usr/ub/ubdev"...Done

Installation completed: Mon May 5 17:42:23 PST 1998
```

To run UniBasic, logout ("exit" or "^D") and login to "unibasic". Then type "unibasic".

Finally, the */tmp* directory is cleared. If an error occurs while removing the directory, the following message is printed:

```
There has been an error removing the distribution directory. Type <CR> to continue, or
Q to quit.
```

The installation process has successfully performed the following procedures:

1) Placed the required files in */usr/bin*: UniBasic, ubcompress, ubconfig, ubkill, ubrebuild, ubterm, lptfilter, and makesp.
2) Placed into *$HOME*: errmessage - UniBasic error message file.
3) Placed into *$HOME/sys*: All system commands, LPT scripts, drivers and terminal control files (term.tvi925, term.ansi, etc).
4) Created the full *$HOME/.profile* environment and startup file.
5) Optionally created the directory *ubdev* under $HOME if UniBasic Development

files were installed.

Errors During Installation

If, for some reason, you did not load the files into the */tmp* directory, an error message is printed and you are asked for the actual directory where you loaded the installation file:

```
Distribution files not found in "/tmp/ub". Make sure you have loaded all files
into the /tmp/ub directory.
```

If you are not logged in from root and attempt to run ubinstall, an error message notifying the user is printed and the installation procedure is aborted.

```
"ubinstall" must be run from the super-user (root) account.

Installation procedure aborted.
```

If you have not created the account to own the UniBasic files, an error is generated and the installation procedure is aborted.

```
You must create an account under which UniBasic can be installed. Refer to the
System Administrator Guide for your system. Most systems have a menu driven
program to assist with account management referred to as the System Administrator
Shell. This program is known on some systems as "sysadmsh", "sysadm", "adm",
"va", etc. and must be run as super-user (root).

Installation procedure aborted.
```

If the installation was successful, sign off root, and sign on using the UniBasic master *login id*. Running from root level while performing conversions or building files may render those files protected and inaccessible from other accounts.

Configuring a UniBasic Environment

When you login to Unix, the system typically executes two shell program files. The first is */etc/profile*, owned by root, followed by any optional user *.profile* (dot profile) in the user's **HOME** directory.

The root */etc/profile* usually includes a definition for **PATH**; the directory search path for commands entered at the shell (The system Command Line Processor). It may also contain commands to print a banner, news of the day or mail.

The user's **HOME**/*.profile* contains definitions of environment variables, and special commands unique to the particular user signing on to the system. This may include changing the default working directory, and/or automatically launching an application environment such as UniBasic. During **ubinstall**, the *.profile* is modified within the **HOME** directory defining only the required configuration environment variables. The following sections describe configuration options, using Environment Variables, for UniBasic.

All users created with an identical **HOME** directory automatically run the same *.profile* at login. When creating multiple user accounts, you may default all users to the same **HOME** directory, or

copy the supplied default *.profile* to each of the users newly established **HOME** directories. Once copied, modify the environment variables (such as **LUST** or **SPC5**) specific to that user accordingly.

During installation, the directory **HOME/sys** is created to contain the *sys* logical unit (0). Other logical units may be created under **HOME**, or in another file system entirely.

Directories and Paths

A Unix file system directory is tree-structured beginning at the level known as *root*. Files are accessed by supplying a *pathname* in the form *dir1*/.../*filename* through the tree. Since IRIS and BITS applications have been designed for a single level directory, UniBasic provides a Logical Unit Search mechanism to facilitate single to multi-level directory organization. An Environment Variable may be defined specifying the Unix directories to search for Logical Units and/or Packnames. The environment variable named **LUST** (Logical Unit Search Table) in the *.profile* is used to define the paths to the final level directories with unit numbers (or packnames).

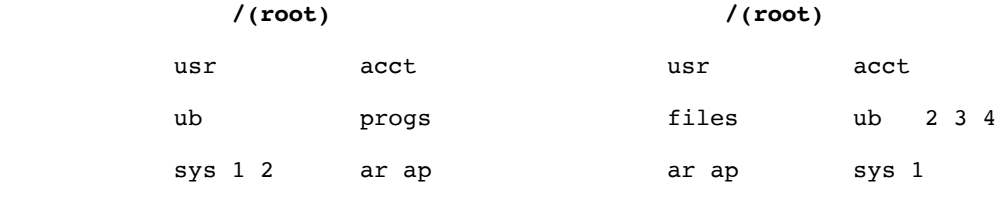
Filenames and Pathnames

Filenames are converted to a series of pathnames, appended one at a time to the entries defined by the Environment Variable **LUST** (Logical Unit Search Table) until a match is found. Standard BITS or IRIS filenames are converted to lower case characters; the Unix standard. Filenames beginning with / are assumed to be full Unix names, and no conversion or logical unit search list is performed. The form *pack:file* is converted into *pack*/*file*. Account branch characters (%&#, etc) and account [grp-usr] suffixes are discarded. Filenames in the form *0*/*filename* are converted into *sys*/*filename*; other files in the form *lu*/*filename* remain as is except leading zeros are dropped from the lu number.

Note: An ISAM file is made up of (2) separate files; the lower-case filename holds the data portion and an uppercase filename is created to hold the ISAM portion. (In the case of Universal files the ISAM portion is the filename with a .idx extension.) Filenames that do not contain at least one letter cannot be used for ISAM data files. See Indexed Data Files.

Organizing Logical Units and Packnames

The following illustration shows various ways to organize directories. You simply list all of the paths in the **LUST** variable to your final logical unit or packname directories. A null path (leading or trailing colon) is replaced by your current default working directory.



```
                    1   1                   2   2
```

The rightmost example shows the simplest structure. Logical Unit zero (sys) and 1 (containing application programs) are under the path */usr/ub*. Data files are on units 2, 3 and 4 under a separate file system (or disk drive) referenced (mounted) as */acct*. The search path for this configuration would be:

```
LUST=:/usr/ub:/acct:/usr/ub/sys
```

In the leftmost example, the sys or LU 0 directory as well as Logical Units 1 and 2 are under */usr/ub*. Both Programs and Files are separated into their own directories (*progs* and *files*) with duplicate logical units 1 and 2 underneath. Assuming all files are accessed as "lu/filename", the appropriate search path for this configuration would be:

```
LUST=:/usr/ub:/acct/progs/ar:/acct/progs/ap:/acct/files/ar:/acct/files/ap:/usr/ub/sys
```

In both cases, you may specify paths to a specific directory if your applications do not specify a hard-coded LU. The entry */usr/ub/sys* is normally included as the last entry in **LUST** to force a search of LU 0 when a command is entered; such as **LIBR** or **DIR**.

Other default units can be selected as well, but it is recommended that they be at the end of the **LUST** to minimize searches. Always construct the search paths in a way that minimizes the total number of searches done for each **CHAIN**, **OPEN**, etc.

Environment Variables

This section discusses the user-configurable UniBasic Environment Variables. Definitions are added to, and exported from, a user's .profile when the default value is insufficient. It is unnecessary to include definitions when a variable's default value is adequate.

ALTCALL Defines the set of BASIC **CALL** numbers used within your application that have equivalents as a different number. For example, your application uses **CALL** 64 to verify date inputs. UniBasic includes a **CALL** 24 functionally compatible for your requirements. Setting alternate 64=24 invokes a **CALL** 24 whenever the application requests **CALL** 64. Multiple **CALL**s can be defined separated by colons, i.e.: ALTCALL 64=24:62=22 See **CALL**.

AVAILREC Defines the numeric value to be returned whenever an **INDEX / SEARCH** Mode 1 requests the number of available records in an ISAM file. If **AVAILREC** is defined, its value is always returned. When undefined, the number of available records is computed by subtracting the number of active records from the created or current file size. See also: Indexed Data Files.

BASEYEAR Defines the system Base Year to be returned for the function **SPC** 20. It is also used to compute the hours counter returned for the **TIM** 2 function. The default for **BASEYEAR** is 1980 unless specified in the environment. Because Unix systems maintain clock values beginning in 1970, you may set **BASEYEAR** to any value from 1970 to the present year. Setting this value outside this range will result in very large (or negative) values for these functions.

BASICMODE Selects the desired operating environment. The default is IRIS emulation with separate SCOPE and BASIC Program command modes. By setting

	BASICMODE=BITS, you operate in a BITS environment, that is both commands and BASIC statements are performed at a single command prompt. The **NEW** command defaults to IRIS or BITS syntax based upon the **BASICMODE** selected. The **NEWI** or **NEWB** commands override the default **BASICMODE** for creation of new programs. Either **BASICMODE** runs both types of programs. Program Files are flagged for IRIS or BITS execution automatically. Text files accessed using **LOAD** or **MERGE** take on the type of the current mode. The BITS **GET** or **GETI** commands allow you to choose the encoding and runtime format for the text files you access.
BCDVARS	If defined and non-zero, all BASIC variables are stored in memory using IRIS BCD format. **BCDVARS** is required when special **CALL**s indiscriminately copy data between numeric and string variables by straight memory copy. Do Not set this environment definition without specific instructions from your Distributor or Dynamic Concepts Inc. **See also:** IRIS BCD Files.
BITSPROMPT	Change the default prompt * displayed in BITS mode. Format is: **BITSPROMPT**='replacement string'
CURRENCY	Define a single character to be output by **USING** whenever the **$** operator is used. Format is: **CURRENCY**=replacement character.
DATESEP	Define a single character other than '/' to separate MM/DD/YY or DD/MM/YY strings. Format is: **DATESEP**=replacement character.
DXTDSIZ	Specifies the number of records to extend an Indexed file when the data portion is full. The default is 1 record. During creation of an indexed file, this value (or default) is read and stored in the file header. Later expansion of the data portion is based upon this size. Once created, this parameter cannot be changed for a file. Depending on your application, changing this value and **IXTDSIZ** can have some effect on performance. **See also: IXTDSIZ**, and Indexed Data Files.
EURINPUT	Selects the programming mode used for **USING**. The default (or zero) mode requires programs to use comma and period in the form: #,###.## When set to one, programs use the international form: #.###,##. **See also:** USING and EUROUTPUT
EUROPEAN	Mode for date input/output formats; 0 for USA Format: MM/DD/YY, 1 for the international format: DD/MM/YY.
EUROUTPUT	Selects the output mode for **USING**. Periods and commas are reversed at output. The default (or zero) mode outputs in the format: 1,234.56. When set to one, commas/periods are reversed; output is represented by the form: 1.234,56. **See also:** USING and EURINPUT.
GOSUBNEST	Selects the maximum number of **GOSUB** and **RETURN** nesting levels in any program. Default is 8 levels deep.
FORNEXTNEST	Select the maximum number of **FOR** and **NEXT** nesting levels in any

program. Default is 8 levels deep.

IBITSFLAG Set to 1 to eliminate the standard IRIS errors: Channel Already Opened (on **OPEN** Statement), and Selected Channel is not OPEN (**CLOSE** Statement). An **OPEN** issued to an already open channel performs an implied **CLOSE** of the channel first.

INPUTSIZE Size in bytes of the input buffer. This size limits the length of a BASIC statement, **LOAD**, **GET** and other operations, such as Long **CHAIN**, that require the Input buffer.

ISAMBUFS Number of buffers allocated for shared memory ISAM files. This parameter is unused at the time of this writing. DO NOT USE THIS VARIABLE.

ISAMFILES Maximum number of opened Indexed file directories. This variable is not needed in UniBasic 9.3 or later. For each file opened, one entry is required for each Directory (index) plus 1 for the data file. In older releases of UniBasic, the default value of 40 supports 8 indexed files open with an average of (4) directories (indices) each. If this value is too small, the error "Illegal Channel (or ISAMFILES value too small)" is printed.

ISAMMAXSECT Determines the maximum C-tree node size that can be read. The node size is given by **ISAMMAXSECT** *128. Default **ISAMMAXSECT** is 8 supporting up to 1024-byte nodes such as those used by dL4.

ISAMOFFSET Define the displacement within ISAM records for maintenance of a system Deleted Record Flag and Delete Link List pointers. Change this offset (Default 0) when your applications write data within the first 5 bytes of a record following deletion. This offset is not used with Universal Data files.

See also: Indexed Data Files and Universal Data Files.

ISAMSECT Determines the C-tree node size. The node size is given by **ISAMSECT** * 128. The default **ISAMSECT** is 4 resulting in 512-byte nodes. If large keys or a large number of keys are needed, then setting **ISAMSECT** to 8 is recommended.

Note: Following deletion of a record in a Non-Universal Data file, DO NOT WRITE (clear) the entire record or the delete list will be corrupted.

IXTDSIZ Specifies the size in bytes to expand a file's Index portion when the index is full. The default is 512 bytes. During creation of an indexed file, this value is read and stored in the file header. All further access and expansion to the file's index portion is based upon this size. Once a file is created, this parameter cannot be changed for that file unless the file is rebuilt using a new **IXTDSIZ** value. Depending on your application, changing this value along with **DXTDSIZ** (and then rebuilding a file) can have a great effect on performance.

See also: DXTDSIZ

LOCKRETRY Record lock retry counter. A value of zero (default) provides for unlimited

record lock return (aka IRIS Revision 7). Any other positive value selects the number of retries (in 5 second intervals) attempted prior to issuing a Record Lock error to the application.

See also: Record Locking

LONGVARS Change the default (0) setting to provide for the global use of long variable names. When set to 1, long variable names are allowed globally; setting 0 disables long names. The **variable** command may be used to override this default at any time.

LUST Logical Unit Search Table. Defines the entire series of Unix paths to search for program and filenames in the form *filename*, *lu/filename* or *pack:filename*. If this parameter is not defined, only the current working directory is searched. Filenames beginning with / are assumed to specify the entire path to the file and the **LUST** definition is not used. The following table illustrates the search paths used for a simple *filename* and *lu/filename*.

```
LUST=:/usr/ub:/ub/sys:/usr/ub/1:/usr/acct:/usr/acct/2
```

filename	pack:file or lu/file
filename	lu/filename
/usr/ub/filename	/usr/ub/lu/filename
/usr/ub/sys/filename	/usr/ub/sys/lu/filename
/usr/ub/1/filename	/usr/ub/1/lu/filename
/usr/acct/filename	/usr/acct/lu/filename
/usr/acct/2/filename	/usr/acct/2/lu/filename

LUST should be constructed to minimize number of searches required to locate programs and files. If an application under IRIS or BITS defaults to a specific logical unit containing programs or data, set the current working directory to that same location. This is accomplished by including a **cd** *pathname* command within the *.profile*.

Note: The maximum number of entries in the **LUST** is 24.

If all file and program access is in the form *lu/filename*, or *pack:filename*, define **LUST** to provide the path to the directory containing the actual numbered (or named) logical units only.

If you rely on the IRIS or BITS LU Search for other Logical units, then you must include full paths directly to each directory.

To ensure the fastest access to programs and files, determine whether your

application performs more **OPEN** or **CHAIN** statements. List your entries in **LUST** accordingly. If most filenames include a Logical Unit or packname, list entries terminating at the directory containing the *lu*, and finally list direct paths to each named or numbered directory.

MAXACCSLEEP Define accuracy vs. performance of the Unix sleep timers utilized by **PAUSE**, **SIGNAL 3**, **INPUT TIM**, record locking, etc. Since some older Unix systems provide timer accuracy only to the nearest second, UniBasic employs the following software method to ensure accurate tenth-second timers:

First, the specified delay is rounded down to the nearest whole second. If at least one-second of delay is warranted, the process sleeps, allowing other processes to run, for that number of seconds. Following the sleep period, UniBasic 'spins', i.e. wastes CPU time, by watching the clock for the remaining partial second.

Most applications are not timing critical. Substantial system wide performance is realized by configuring delays to round up to the nearest whole second. That is, a delay of 5 tenth-seconds is rounded to a full second.

MAXACCSLEEP defines the delay value, below which, exact accuracy is required. Delays at or above this value always round to the next whole second. A value of zero, the default, provides the highest accuracy at the expense of additional system overhead. A value of one always rounds, etc. To ensure accuracy on all delays below two seconds, set the value to 20.

Most systems support highly accurate timers without the requirement to waste CPU time, including Linux, AIX, SCO Unix, NCR Tower 7xx/8xx, MIPS and Motorola 88000. These systems default, automatically, to 65535 which enables the system specific timer. On systems that do not support accurate timers, the value defaults to zero.

MAXPORT Change the default automatic *port number* assignment to a value other than 999. The maximum port number is 2048. Used for automatic **port number** assignment by the **SPAWN** statement, and during sign on when **PORT** and/or **PORTS** are undefined. Set to 99 to prevent automatic assignment of 3-digit **port numbers**.

See also: PORT, **PORTS** and Port Numbering and Phantom Ports.

MAXVARS Control the maximum number of variable names that can be used within a program. By default, 348 unique variable names may be used within each program. Setting **MAXVARS** to a number limits the number of variable names to that number. This value is only checked when a program statement is entered adding a new variable to a program. Setting **MAXVARS** to 93 ensures backwards compatibility to IRIS or BITS. Setting **MAXVARS** to the string "extended" increases the normal limit from 348 to 1113. The increased limit is only effective for programs that are newly created while **MAXVARS** is set to "extended". **MAXVARS** does not need to be set in order to load, run, or modify a program created with the extended variable table.

MSC7 Define the numeric value to be returned by the **MSC(7)** function. If **MSC7**

is not defined (or defined as 65535), your UNIX group number * 256 plus the user number is returned. **MSC(7)** will yield unpredictable results when the group or user numbers are greater than 255.

PFCHAR Define a single replacement character for any @ character terminating a *filename*. Format is: **PFCHAR**= replacement character. When **PFCHAR** is not defined, the trailing @ character is ignored and terminates a *filename*. Therefore, the filenames DATA and DATA@ both select the same *filename*. This default operation is recommended as a method of preventing @ characters from becoming part of a Unix filename. @ is not a portable filename character, and its use may interfere with some Unix shell commands.

On some IRIS systems, users may have nearly identical files, such as DATA and DATA@. Defining this option removes the requirement to modify applications and *filenames*.

To define this option, choose a single character to replace @, such as **PFCHAR**='-'. In this example, any attempt to **BUILD** or **OPEN** a filename such as DATA@, results in an operation to DATA-.

Note: This option should only be utilized on systems where a blind conversion is being performed. It will safeguard against conversion errors when an IRIS system has nearly identical data and poly *filenames*. Resellers converting known systems are advised to rename or delete conflicting filenames. Most often, duplications are the result of an older Indexed file (itself no longer in use) being recreated as a Polyfile.

Once files have been built with this substitution in effect, the option must remain set, or all program occurrences of the @ must be changed to the specified replacement character.

See also: Setting up .profile for Multiple Users.

PORT Force the current session to operate as a specific **PORT** number, i.e. PORT=23. The value of **PORT** can also be set to the string “any” in order to ignore the terminal name and use the first available port number (starting down from the maximum port number). The maximum Port number is typically 1023 unless your system is licensed for a greater number of users. The value “any” is sometimes used within a profile script to prevent telnet pseudo-devices from conflicting with users logging in on serial lines.

PORTS Define a specific port numbering order. The format of this definition is:

PORTS=tty00:tty1b:#7:tty1c . . .

In this example, Port 0 is tty00, Port 1 is tty1b; starting at Port 7 is tty1c leaving ports 2-6 unused. When neither **PORT** or **PORTS** is defined in the environment, port numbers are assigned based upon the tty name (tty23 is port 23). If a name conflicts with an existing port (or a port already in use), a number is assigned backwards starting at **MAXPORT**. To prevent

automatic assignment, all system tty device names not in the form ttynnn (where 'nnn' are digits) should be listed. Ports conforming to the normal numbering conventions need not be defined.

See also: Port Numbering and Phantom Ports.

PREALLOCATE This variable contains several flags which, when added together, define options for processing data files.

Options fall into two categories, runtime and permanent. Permanent options are indicated by •. Runtime options affect current file operations when enabled. Permanent options affect all future access to files created when that option was enabled. Permanent options are stored within the file's header and typically define file limits or data storage formats.

1 Preallocate all blocks for contiguous files and initialize to zero bytes. You might set this value on a new system to force files to occupy physically contiguous space on the disk. Note: Indexed files store keys in a separate file, and may be built too large using older style IRIS or BITS creation algorithms. If this flag is set, modify your file creation sizing algorithms. <u?Runtime option</u>.

2 Do not allow writing past the original created size of a contiguous file (no expansion). Runtime option.

4 When expanding a contiguous file, do not fill in all records between current end of file and new record to write. ** Runtime option.

See also: Contiguous Files.

8 Check Formatted files and return a Record Not Written error if a record has never been written or contains only null (zero) bytes. Runtime option.

16 When expanding a Formatted file, do not fill in all records between current end of file and new record to write. ** Runtime option.

See also: Formatted Files.

• 32 Always **BUILD** and **CREATE** new files in IRIS style BCD record format. This flag may be required if: a) data files were converted from IRIS and b) your application indiscriminately copies entire records from one file to another using variables other than the actual field specification. For example, a **MAT READ** of a string or 1% array. Setting this flag for new installations forces creation of potentially transportable data records for future relocation to other hardware platforms. Permanent option for files created while enabled.

See also: IRIS BCD Files.

Note: DO NOT set this mode during IRIS or BITS conversions.

• 64 Always **BUILD** and **CREATE** indexed files in IRIS/BITS 8-bit key format. Forces keys to be stored in exact IRIS/BITS format. This flag is required when applications utilize binary information in the keys. DO NOT

set this mode during conversion of files from IRIS or BITS. Permanent option for files created while enabled.

See also: IRIS BCD Files and Indexed Data Files.

Note: DO NOT set this mode during IRIS or BITS conversions.

---

128 Restrict Indexed files from dynamic expansion. When built, the number of records specified to **BUILD** or **CREATE** is retained in the file header as the maximum number of records for the file. The status **E=3** is returned from **SEARCH #** and **INDEX#** when the file dynamically expands to this record number. Runtime option.

See also: Indexed Data Files.

256 During Indexed File Record Deletion, check for record already deleted. When deleting records and adding them to the delete chain, this runtime flag forces an initial check of the delete flag prior to deletion. If the record is already flagged as deleted, an exception status (E=1) is returned, and the record is not added to the deleted record list. This flag may be required if your applications arbitrarily delete records not currently in use. Runtime option.

512 Permit writing past the record boundary of an Indexed file in a single operation. Normally, error 144 is generated whenever a single write operation will cross a record boundary. This option should only be used when the application is certain that all records to be written are previously allocated, otherwise the file's deleted record list might be corrupted. This option is runtime in nature, affecting all open Indexed files. Runtime option.

• 1024 Always **BUILD** and **CREATE** new files in IMS style BCD record format. This flag may be required if: a) data files were converted from IMS and b) your application indiscriminately copies entire records from one file to another using variables other than the actual field specification. Permanent option.

Note: DO NOT set this mode during conversion of files from IRIS or BITS.

See also: IMS BCD Files.

2048 Reserved for future use. DO NOT enable this option within your application.

4096 Prevent all write operations to deleted records within Indexed files. Prior to each write operation, the record's delete flag is checked. If the record is

flagged as deleted, set **ERR(8)** c-tree status to 144 and return BASIC error 123. Runtime option.

Note: Formatted and Contiguous, including Indexed, files are typically created containing a 512-byte header and no data records. For Contiguous and Indexed files, the number of records specified to **BUILD** or **CREATE** is stored within the header for use by **CHF** and runtime-limiting **PREALLOCATE** options. Only when **PREALLOCATE** option 1 is set are records physically allocated at creation.

Prior to each write operation, the number of records between the current physical end-of-file and the end of the record being written is computed. Missing (intervening) records are automatically written to the file. This process may take several seconds depending upon the number of intervening records that must be written.

When setting **PREALLOCATE** to prevent intervening record allocation, only the record to be written is allocated. Reading any non-existent record results in the transfer of a null data without error. Although these files are completely valid, warning messages may be printed by the Unix command **fsck** (File System Check) when 'gaps' are detected in the structure. These files are sometimes referred to as sparse files.

Within Formatted files, **PREALLOCATE** option 8 is used to interpret null records as Records not written.

- 8192 Always **BUILD** and **CREATE** new files as a Universal type file. The file will contain IRIS style BCD data. If this flag is set, the 32 and 64 option flags are ignored. Permanent option.
- 16384 Always **BUILD** and **CREATE** new files as a Huge Universal type file. The file will contain IRIS style BCD data. If this flag is set, the 32 and 64 option flags are ignored. Permanent option.

SCOPEPROMPT Choose an alternate prompt while in SCOPE Command Mode (**BASICMODE=IRIS** only). The default prompt # is replaced using the form: **SCOPEPROMPT**='replacement characters'.

SPC5 Define the numeric value to be returned whenever the **SPC(5)** function is called. If **SPC5** is not defined as an environment variable (or set to 65535), your UNIX group number * 256 plus the user number is returned. The **SPC(5)** function will yield unpredictable results when the group or user numbers are greater than 255.

See also: Setting up .profile for Multiple Users.

SPC7 Define the numeric value to be returned whenever the **SPC(7)** function is used. If **SPC7** is not defined as an environmental variable, zero (0) is returned.

STRING Select alternate string processing for BASIC to match HAGEN Business Basic. To invoke HAGEN String Processing, use the form: STRING=HAGEN.

TABFIELD Change the number of spaces between comma fields in **PRINT** statements from 20 to the new numeric value specified.

UBKEYFILE If defined, **UBKEYFILE** is a path to a key file containing encryption keys used to create or open encrypted files (see Encrypted Files). The path can be a **LUST** relative path or an absolute path. If the key file itself is encrypted, a **SYSTEM** 100 statement must be used to define the key **SYS_KEYFILE** before opening or creating any encrypted files.

WINDOWS Define the maximum number of Windows that may be opened by this user. If **WINDOWS** is defined, the main screen is counted as the first Window. Each WINDOW requires approximately 64 bytes of storage for the array. As Windows are created, memory is allocated based upon twice the number of characters in the Window. The main screen occupies (80 *24 *2) characters of memory for a 80 column, 24 row screen.

See also: Windows and Output Considerations, WINDOW, CALL $WINDOW, and MSC Functions.

WARNING: THE FOLLOWING UNIX ENVIRONMENT VARIABLES MAY BE EXAMINED OR CHANGED AS REQUIRED. HOWEVER, CHANGING THESE VARIABLES WILL LIKELY AFFECT THE OPERATION OF OTHER UNIX APPLICATIONS.

HOME The home directory of the user, i.e. */usr/ub*.

HZ The clock rate used internally by the Unix system. For most systems, this value is either pre-defined to the compiler or is already in the environment. This value is used to compute certain **TIM** and **SPC** functions; the **BYE** command and pause durations less than 1 second. Do not change this variable unless incorrect times are reported by the above noted functions.

See also: MAXACCSLEEP Environment Variable.

TERM Many applications, including UniBasic, retrieve the value of this variable to select a terminal driver for screen operations. While many applications rely on the Unix **termcap** or **terminfo** drivers, UniBasic developers have the flexibility of their own driver system.

See also: Configuring Terminal Drivers

PATH The Logical search path for Unix commands issued to the shell. PATH=:path:path:path: ... The **PATH** is only referenced when shell commands (or Unix commands) are entered while in *command mode*. To open pipes without supplying the full pathname (i.e. **DUMP $more**), append **PATH** definitions to **LUST**, i.e.: LUST=$LUST:$PATH

Note: The following are useful Unix commands that may be of interest to the user. For more detailed information, consult your Unix documentation.

---

stty Command to reset terminal configuration, Baud rate, parity, backspace and control characters, xon/xoff protocol, character length, mapping of return to return-linefeed, etc.

Unix typically assigns the characters BREAK and DELETE for QUIT and INTERRUPT functions used to abort a process. These functions are reset upon entry to UniBasic to the characters ^D **[EOBC]** and **ESCAPE**.

When a Unix command is performed from UniBasic (Command mode, **SYSTEM** statement), the functions are reset to their initial Unix definitions for the duration of the system command. Some users find it desirable to use ESC and ^D for both system and UniBasic commands. The stty command may be executed from within the *.profile* to change the default Interrupt and Quit functions.

Note: To ensure proper terminal operation, incoming **stty** parameters are saved whenever a UniBasic process is launched. Issuing **stty** or similar commands, within UniBasic, have little effect since UniBasic restores and resets these parameters. Certain changes are permitted, using the **!** command, such as changing the baud rate.

---

cd $HOME/1 Command within *.profile* to set the user's default Logical Unit to 1 when LU 1 directory is below **HOME**.

umask Set to zero to provide for pass through protections to Unix. Any non-zero value forces Unix to XOR supplied protection digits with this umask value. For example, if umask=7, then all lower protection digits are cleared. See File Attributes, Protection and Permissions for a complete discussion of the Unix protection system.

ulimit The ulimit command sets the upper limit (in blocks) for files created on the system. Set this value to the largest allowed value to allow your applications to control file size. If this value is set too low, a Write Error will be given when a file reaches this maximum size. This value may be defined in */etc/profile*, as part of the user's account or within the Kernel. Contact your supplier if this value is too small for your needs.

Setting up .profile for Multiple Users

When multiple users default to the same **HOME** directory, you may insert statements within *.profile* to determine the *login name* used, and configure environment variables accordingly. The following statements might be added to **HOME**/*.profile*.

To set a different **SPC5**, **MSC7** or **LUST** (Logical Unit search path) based upon the user signing on:

```
case   $LOGNAME in

     "doug")  SPC5=32774;LUST=$LUST:/usr/drive1;;

     "laura") SPC5=32896;LUST=$LUST:/usr/drive2;;
```

```
"mike")  SPC5=32768;LUST=$LUST:/usr/drive3;;
*)       SPC5=16384;;     #Default other users.
esac
```

The previous example configures different **SPC5** values and alters **LUST**, appending to its previously defined value the additional search of *drive1*, *drive2*, or *drive3* only for doug, laura or mike. By appending a previous base value, it is unnecessary to redefine the entire **LUST** specification for each user. A total re-definition would take the form:

```
LUST=/usr/ub:/usr/ub/sys:/usr/drive1.
```

For further information , refer to the Unix manuals on Shell Programming.

Command Line Interpreter

Two separate command line interfaces are provided within a running UniBasic process. *Command Mode* is signified by the prompt character # (**SCOPEPROMPT**) printed at the left margin. System commands (UniBasic or Unix) and program names may be entered while in Command Mode.

BASIC Program Mode is entered by the **BASIC** Command and has no prompt character. Programming and debugging is performed while in BASIC Program Mode.

```
#ls              Issue Unix Directory command
#LIBR {param}    Command Mode example
READ var.list    BASIC Program Mode example
```

It is also possible to configure all commands for operation from a single *command mode* by setting **BASICMODE**=BITS. In this configuration, a single prompt * (**BITSPROMPT**) is always displayed at the left margin.

```
*ls              Issue Unix Directory command
*LIBR {param}    Command Mode example
*READ var.list   BASIC Program Mode example
```

Launching UniBasic From Unix

SYNOPSIS: Launch a UniBasic Process

unibasic {-ffilename} {-Ffilename} {-Pfilename} {-Xfilename} {-s} {-o} {-t} {-v}

DESCRIPTION

Start a UniBasic session on your terminal. The current environment is read for all

pertinent variables, a Port Number is established, a Message Queue is created and the terminal modes are reconfigured. If this is to be an interactive keyboard session, the terminal is placed into *command mode*.

filename is an optional name of any BASIC program file. The specified *filename* must be in the current working directory, or in one of the supplied *pathnames* specified in the environment variable **LUST**. The *filename* may also include a *lu* identifier, or be a full Unix pathname beginning with '/'.

The *-f* switch is used to immediately execute the named *program* file. If the specified *program* terminates or an error occurs, the terminal remains within UniBasic in command mode.

The *–F* switch is also used to immediately execute the named *program* file. However, if the specified *program* terminates using **STOP**, **BYE**, **SYSTEM 0**, **END**, **CHAIN ""**, non-trapped **ESC**, **[EOBC]** (CTRL+D) or an abortive error, the session is terminated, and control returns to the point UniBasic was launched; see below.

The *–P* switch is identical to *–F* except no terminal translation will be used and the UniBasic startup messages are suppressed.

The *–X* switch is used by DynamicXport to run UniBasic applications and must not be used outside of that environment.

The *-s* switch requests entry of a new Software Selection Number (SSN). The SSN might be changed when you are installing additional terminals, installing additional products (such as IQ) or converting a demonstration License into a paid-up License of UniBasic.

The *-o* switch requests the entry of a new OEM Selection Number (OSN). The OSN is used to control execution of one or more dealer-protected software packages.

The *-t* switch requests the entry of a new OEM Selection Number (OSN) similar to the *-o* switch. This OSN is considered temporary and is not stored into the system. The *-t* option is used when the owner of protected software wants to temporarily grant access to the source code. This access is restricted to the single terminal issuing the *-t* switch.

(Release 9.2.2) The *-v* switch displays the full version text. The *-V* switch displays just the version number and then exits.

When a session terminates using **BYE**, **SYSTEM 0** or **1**, or an aborting condition using the *-F filename*, the process is exited, and all terminal characteristics are reset to the incoming values. If the UniBasic session was started from the shell, then the shell is resumed. If launched from the *.profile* using a **UniBasic {switches}** command, the *.profile* resumes at the following statement. To return the user to login mode at process termination, place an **exec UniBasic {switches}** command as the last line of the *.profile*.

EXAMPLES

```
unibasic -f menu

unibasic -s
```

tee savefile

ERRORS

```
No SSN currently entered

Demonstration system, not for resale

License number from ssn does not match actual license

Cannot allocate sufficient memory

Cannot initialize ISAM. Check ISAMBUFS/ISAMFILES definitions

Cannot open term.xxx file. No CRT translation in effect!

Error loading CRT file term.xxxx. No CRT translation in effect!

Could not open 'errmessage', no error messages available!

Too many users; max = n

Port n is already signed on and in use
```

See also:

Environment Variables, Entering an SSN, PORT, PORTS, CRT TERM Files, Program Files, Port Numbering

Terminating a UniBasic Process

Once initiated, an interactive UniBasic process remains active until terminated. Interactive, as well as Phantom Port, termination is provided for with the **SYSTEM 0** and **BYE** commands.

Non-interactive UniBasic processes, such as those launched using UniBasic *-F* or **SPAWN**, terminate when the specified program stops execution.

All of the above (normal) methods provide for a graceful termination of UniBasic. Open files and devices are closed, the Message Queue is removed, the terminal driver is reset to the modes present upon entry and the process terminates.

Abnormal termination, resulting from the following events, may require operator intervention before other tasks may be performed:

- Memory Fault - core dump
- Hardware failures.
- Receipt of a non-supported signal. UniBasic supports the signals HANGUP(1), TERMINATE(15), SYSCHILD, SIGPIPE, INT, QUIT, SIGUSR1, SIGUSR2. Any other signal may cause abnormal termination.

The following functions may be performed manually, from the failing terminal, when an orderly shutdown did not occur. From a remote location, only the Message Queue must be deleted, after which you should kill any remaining processes, including the shell, associated with the port.

- Issue the Unix command: **stty sane** and press CTRL+J or RETURN if the terminal is misbehaving.
- Issue the Unix command: **ipcs** to review, and **ipcrm** to remove the Message Queue for the offending port.
- Issue the Unix command: **ps** to determine and kill any remaining suspect

processes under the port's control.

- Sign off and back on to reset all terminal parameters before re-launching another UniBasic process.

See Also: Message Queues

Licensing a New Installation

If this a new installation, you may be asked to enter an SSN the first time UniBasic is launched:

```
$ unibasic

UniBasic Level 8.1

All Rights Reserved. Copyright (C) 1987 - 2006 by:

Dynamic Concepts Inc. California USA

No SSN currently entered

Enter Software Selection Number (SSN), RETURN to remain the same
```

If you do not yet have an SSN, press [RETURN] to invoke a single-user grace period. A special warning about the grace period is printed periodically until you enter an authorized SSN.

To obtain an authorized SSN for this installation, contact your supplier with the following information:

- License Number displayed
- Number of ports desired
- Type of system
- End-User name
- Options, other DCI products such as IQ runtime, IQ development or IMT.

SSN entry is space and case insensitive. After entering all characters, press [RETURN]. You will be prompted to enter the User Name. Enter the name exactly as printed on the SSN License Agreement. Entry of the name is case and space sensitive. Backspace may be used to correct input errors.

Following entry of the SSN and User Name, immediately issue a **BYE** command, and restart UniBasic. If the SSN was accepted, the *command mode* prompt is displayed. If you are again asked to enter an SSN, either an error occurred during entry, or the License Number does not match the supplied SSN Report.

The SSN contains the licensed configuration for the specific License Number. Currently, an SSN includes Demonstration options (Permits operation for up to 90 days), the number of concurrent Ports that may run UniBasic, and additional information to enable IMT and IQ.

Note: When using Software Licensing, the license number is keyed to your specific system. Prior to updating the operating system (Unix), or replacing or re-

formatting your disk drive, contact your distributor or Dynamic Concepts concerning the deactivation and replacement policy for your license.

Changing the SSN Activation Key

Prior to changing a system's SSN, verify that you have a copy of the existing SSN number, as contained within the file */etc/DCI/ssn*. Prior to installing a new **SSN**, you may print this text file, or use the Unix **cp** command to make a copy of this file. You will need root permission to access this special file.

To change an existing SSN, for example to add additional users, enable additional products or convert a demonstration license into a full license, issue the command:

```
ssnmaint
```

Any existing SSN is displayed.

Enter the new SSN (case and space insensitive) and Customer Name (space and case sensitive). After pressing return, *command mode* is entered.

Following entry of the SSN and User Name, restart UniBasic.

A new ssn can also be entered by using the command: unibasic -s

See also: Launching UniBasic from Unix.

Launching UniBasic Ports at Startup

You may provide for turn-key operation whereby Unix automatically launches terminals directly into UniBasic, and/or your application. Start-up is performed at system initialization (IPL) or whenever a terminal is evicted or a user signs off.

This feature may be used for interactive or phantom (background) jobs.

The following instructions apply to most Unix based non-server environments.

Make the following changes for each port to be initialized:

1. When starting an interactive terminal, change the **getty** command inside the */etc/inittab* entry for the terminal to:

   ```
   login unibasic </dev/ttyxx >/dev/ttyxx 2>&1
   ```

   where 'xx' is the system tty name.

2. Change the *.profile* to set the necessary tty options. The **PORTS** environment variable should be defined within *.profile* to ensure the same *port number* assignment for each automatic startup.

a. *.profile* based upon a Login User Id: Create a login 'ubauto' with the same **$HOME** directory, group and user id as your 'unibasic' login. Then add a single line in *.profile* to handle all automatic startup ports:

```
[ $LOGNAME = ubauto ] && stty sane
```

---or---

b. *.profile* based upon which tty when ports require different settings:

```
case `tty` in

    *tty01) stty 9600 sane ;;

    *tty02) stty 1200 sane ;;

esac
```

3. When starting a *phantom port*, change the command to:

```
PORT=n login unibasic </dev/null >/dev/null 2>&1
```

where 'n' is the desired *port number* for the process. No changes are required to *.profile*. You may also include **PORT=n** for interactive ports when the **PORTS** environment variable is not defined, or special numbering for each process is desired.

The 'login unibasic' forces a direct login and execution of the *.profile* as if the login id 'unibasic' was entered on a terminal.

The *.profile* must contain the line **exec unibasic** as the last line to launch the session. The initial copyright is printed and the session is waiting input at *command mode*. You may also force a starting program using the form:

```
exec unibasic -f program.
```

See also: Setting up .profile For Multiple Users, **PORT**, **PORTS**, Port Numbering and Phantom Ports, Launching UniBasic from Unix, Port Number

Configuring Printer Drivers

Two printer drivers are supplied for use with your applications; **lpt.iris** and **lpt.bits**. An additional file **lpt.sample** documents various modifications and sample printer drivers.

lpt.iris is designed for applications requiring locked printers. Users attempting access to a locked device receive an error until it is available.

lpt.bits is designed for multi-user spooling applications. Both drivers are similar and may be used with either IRIS or BITS applications.

You may examine and change the driver saving copies using the *filenames* required by the application, i.e. lpt1, lpt2, etc. A driver must use a lower-case *filename* and be stored within a directory listed in the **LUST** Logical Unit Search Table. Do not place a $ as the first character of the *filename*. The $ is a flag recognized by UniBasic as a request to open a *pipe* to an executable file.

For a printer driver to operate correctly, it should be owned by the master UniBasic account with the permissions 555. Before using the driver, issue the Unix command: **chmod 555 *filename*** to set the proper permissions. If further modifications are necessary, issue **chmod 666 *filename***, perform editing as required and reset the permissions to **555**.

The following is a line by line description of the supplied **lpt.iris** printer script. It is designed to run as an executable shell-script under the borne shell only. It operates as a *pipe*, taking as its standard input data transmitted by **PRINT #** statements.

```
#lock LPT - Printer Driver for UniBasic
```

If the first line begins with '#lock', locking is employed to guarantee single user access to the device. Typically required for check or form printers.

Note: No tabs, spaces, blank lines or other characters may exist before the '#lock'.

```
#Module: lpt Level: 1.2 Modified: 7/18/88
```

Comment indicating revision of supplied lpt script.

```
trap "" 1 2 3

INODE=`ls -i $0`

INODE=`expr "$INODE" : ' *\([0-9]*\)'`

LOCKFILE=/tmp/lk.$INODE

trap "rm $LOCKFILE" 0
```

Setup for cancellation, and signals. Determine the filename of the lock file built, and setup to remove the lock file on script termination.

```
OPENSTR='\c'
```

Define the string of characters to be sent to the printer when opened. The \c is a special flag for the Unix **echo** command to avoid sending a return and line-feed following the characters. Enclose within single quotes; characters as themselves, \0nn for octal using 7-bit form, such as \015 for carriage return; \? special characters such as \n new-line, \r return, \f form-feed. For a complete list, refer to your Unix documentation on the **echo** command.

```
CLOSSTR='\f\c'
```

Define the string of characters to be sent when all output is complete. The same rules apply as with OPENSTR.

```
FILTER='lptfilter BX \010'
```

Define output filtering. Supplied by Dynamic Concepts, **lptfilter** provides output translation. Modify the data between quotes to contain 'lptfilter' and pairs of parameters representing data sent by the application, and replacement strings. The above example changes all BX mnemonics (Begin Expanded Print) to the replacement string ASCII character 10 (octal). For additional information, see also **lptfilter**. **lptfilter** prints directions for its use when typed as a command at *command mode*, or at the shell.

```
PTRDEV='/dev/lp00'
```

Define the device to actually receive the finalized data sent by this script. To send the data through the spooler, this line would contain the actual spool command within single quotes, such as lp -s.

PTRBAUD='9600 opost onlcr istrip ixon cs8 -parenb' Define for a serial port the baud rate and other characteristics required to define the port for the printer. The above options indicate 9600 baud, process post output, change new-line to carriage return, strip high bit, Xon/Xoff protocol, etc. This string is not required for parallel printers, and it is not used (only defined) in our example. **See also:** Configuring Serial Printers below.

Standard Parallel Operation to device:

```
(echo "$OPENSTR";cat -;echo "$CLOSSTR") | $FILTER >$PTRDEV
```

Standard Parallel Operation to a spooler:

```
(echo "$OPENSTR";cat -;echo "$CLOSSTR") | $FILTER | $PTRDEV
```

tandard Serial Operation to device:

```
(stty $PTRBAUD >$PTRDEV <&1; echo "$OPENSTR"; cat -; echo "$CLOSSTR") |
$FILTER > PTRDEV
```

Create a sub-shell to perform the following processes under the process of the script itself:

1. Invoke **echo** to transmit the defined opening string.
2. Invoke **cat** getting its input from standard input (the pipe).
3. Invoke **echo** to transmit the defined closing string.

All of the output from the sub-shell process is optionally piped again through **lptfilter** and finally redirected to the selected device or through the spooler.

If **lptfilter** is required, add the command **| FILTER** immediately following the close parentheses before the **>PTRDEV** or **|PTRDEV** respectively. If not, remove the **| FILTER** command. This increases the speed of the script, preventing an additional process from starting.

By opening the **lpt** printer, we have started the process **sh** (shell) to interpret the script, another sub-shell to perform items 1-3. The sub-shell will have **echo** or **cat** opened and running until the BASIC program closes the channel. Finally, the optional **lptfilter** process may be running. If you have directed output to the spooler, additional processes may also be started.

The entire operation is quite fast, and easily configured. For special applications, you might write in C a printer driver specifically for your needs.

See also: Pipes, lptfilter, filename

Configuring Serial Printers

In the previous section, each time the printer is opened the Unix **stty** command is sent to initialize the device. With some printers, this may cause problems such as overflowing buffers, or losing flow control when the device is turned off-line or out of paper.

If you experience problems with serial printers, check the following conditions:

1. Is the printer set for Xon/Xoff protocol, and if so, does the PTRBAUD definition contain the option for *ixon*?
2. Is the printer set for DTR protocol, and if so, is the wiring correct for the mux, and does the mux support this protocol ?
3. Is the script properly set for serial operation including the Unix **stty** command as the first command within parenthesis?

These conditions should be checked by your installer with a break-out box. You may also have to check with the manufacturer of the printer, system and mux to verify that your configuration and use is supported by the hardware and Unix drivers.

If you continue to have problems:

1. Modify the PTRDEV definition to specify a temporary file for printer output, i.e. */tmp/printerdata*. Run your report and examine the contents of the file to verify that the data is being correctly sent by the application through the **lpt** script.
2. From *command mode* or shell, use the Unix Commands **stty** and **cat** to configure the port and direct the data to the device:

```
(stty options; cat /tmp/printerdata >/dev/...)
```

3. Once you are able to print data, modify the script using the same parameters remembering to reset PTRDEV to the desired device name.

If printing works, but the printer occasionally loses data or overflows on multiple jobs, it may be necessary to remove the Unix **stty** command from the script. Follow the above example for a parallel printer. Next, add the following code to the system file */etc/rc* or other Unix startup file:

```
(stty ; while : ; do sleep 40000; done ) </dev/... &
```

Insert the proper parameters following stty, and </*dev*/... is the name of the physical device driver, such as */dev/tty23*. This must be a background process as indicated by the terminating '&'.

It should be noted that these changes are only required on systems redirecting data to a physical device, i.e. PTRDEV, is the actual name of a device driver.

When configuring a printer for use with the spooler, these changes are not required.

Configuring Terminal Drivers

Terminal drivers translate keyboard and display mnemonics between applications and various brands of terminals. When launching a UniBasic process, the value of the environment variable **TERM** selects the terminal translation driver for this session. A filename in the form: *term.name* is opened, where *name* is the value of the **TERM** variable.

Terminal files, typically stored within the *sys* directory, must use a lower-case *filename* and be within a path of the **LUST** environment variable. If a matching terminal driver is not located, an error is printed and no terminal translation functions are available for that session.

A number of terminal driver files are supplied for use with your applications including term.ansi, term.tvi925, term.wyse50 and term.wyse60. term.ansi is designed specifically for use with ANSI style terminals and the primary monitor supplied with many systems. The other drivers are for use with Televideo 925, Wyse 50 and Wyse 60 terminals respectively. These may be duplicated and modified for use with other TERM definitions. The Unix cp command may be used to make additional copies of these drivers. For example, to create a Televideo 910 driver, issue the command:

```
cp term.tvi925 term.tvi910.
```

Any standard editor, such as **vi** may be used to adjust the new driver file accordingly.

For a terminal driver to be properly recognized, it must have read-permission enabled and be located within the path specified by the environment variable **LUST**. Once configured, it is recommended that only read-permission remain enabled to prevent corruption.

The names assigned to the **TERM** environment variable are usually defined in the */etc/inittab* or

/etc/gettydefs files. Refer to your Unix system documentation for additional information relating to equating **TERM** names with terminal drivers.

See also: Terminal Translation Files $TERM files

Creating a Customized Installation Media

You may customize the supplied DCI Installation program, **ubinstall**, to include provisions to install your applications, data files, printer and terminal drivers.

To ensure proper operation of DCI supplied products, your customized installation procedure should be added to the existing **ubinstall** script. Failure to perform all of the steps contained therein can lead to problems in an installation.

Within the */tmp* directory during installation, the files at the level */tmp/ub* are moved into */usr/bin*, except for the system error message file *errmessage*.

Files at the level */tmp/ub/sys* are moved into **HOME/sys** as defined during installation.

Directories at the level */tmp* are not moved. Directories at the level */tmp/ub* are moved to **HOME/ub**.

Files in *ubdev* (UniBasic Development) are moved under **HOME/ubdev**.

To create a custom version:

1. Follow the installation instructions on the various DCI supplied installation files (omitting the entry of the *ubinstall* command).
2. Move copies of custom printer drivers, system BASIC programs and any other sys or LU 0 custom items into */tmp/ub/sys* using the Unix **cp** command.

   If you have a complex *.profile*, such as one containing settings which are not prompted during **ubinstall**, place a copy of that *.profile* into */tmp/ub* . It will be necessary to modify the ubinstall script to accommodate this option. Add code in the script following the move of the **errmessage** file to **HOME** to move your custom *.profile* in a similar manner. Be sure that the code is inserted after the initial creation of a **.profile**. Properly coded, installation will replace the default file with your customized *.profile*.
3. Under */tmp/ub*, create any directories that are to be placed under the **HOME** level on your customer's systems. Even if these directories are empty, the **cpio** command will create them for you during installation.
4. Use the Unix **cp** command to move copies of program and data files into the associated installation directories under */tmp/ub*. You may use the **ln** command (link) instead of **cp** to reduce disk space requirements.
5. Use the Unix commands **ls, chown, chgrp, chmod** to verify and set the permissions, *user id*, and *group id* of your directories and files. Verify that your LPT scripts have the *x* attribute (i.e. 555). It is recommended to select a default *group* and *user id*, as is the case with DCI supplied programs and files. During installation, **ubinstall** changes the *group* and *user id* of the supplied DCI files and directories to the prompted owner/manager of the UniBasic installation.

6. Modify the supplied **ubinstall** script to automatically create and/or move your directories to the desired location (optional). Also add code to allow for other directories loaded at the level */tmp* to be installed or moved onto another file system, drive or directory.
7. Your */tmp* directory is now ready to be copied onto a master distribution archive file. Issue the following commands from root:

```
cd /tmp

find . -print | cpio -ovc >filename
```

Note: If you prefer to use the Unix **tar** command, that format is acceptable for your master media. Change your installation instructions accordingly.

Introduction To UniBasic

UniBasic is a formal language used to communicate with a computer. It is in the family of computer languages that have been designed using Dartmouth BASIC (Beginner's All-purpose Symbolic Instruction Code). Unlike the binary language of the computer, however, BASIC is easy to learn and use. And like any language, UniBasic has a set of rules, syntax, and conventions. This chapter introduces the rules, syntax, and conventions for UniBasic programming.

UniBasic has two basic modes of operation; Command mode and Program mode. Command mode is the outer shell of UniBasic, just above the unix operating level. While in the Command mode you can type BASIC commands that deal with the system and the UniBasic environment.

One of the commands that you can enter while in the Command mode is BASIC.

UniBasic lends itself to a variety of applications. The computer operates as a calculation or programming device. In *immediate mode*, the computer works as a calculation device, and executes instructions directly as they are entered. In BASIC *programming mode*, instructions are not executed until the computer is instructed to run them. In this form, the BASIC instructions comprise a program that can be stored for later use.

A program is a set of computer-recognized instructions that perform a desired series of operations. For example, a payroll preparation system written in BASIC is a program that a computer can execute.

Data

Data is the information that is supplied for a program to produce a result. Data may come from outside the system, or it may be in the computer memory as a result of a previous computation. An important characteristic of a data element is its type. In UniBasic there are two basic data types; numeric and string.

Numeric data is made up of numbers that can be manipulated by arithmetic operators. String data is comprised of any ASCII character. Although string data may contain numeric characters, there can be no direct arithmetic manipulation of string data. There is a special type of string data called CRT mnemonics and expressions. This group of data is used to control video terminal functions.

Both numeric and string data can have two forms; constants and variables. A constant is data that is used by a program and does not change. An example of this form of data is the mathematical constant pi. This is the ratio of the circumference of a circle to its diameter, and is approximately 3.14159. A variable is a storage area that contains the current value assigned to the name associated with it.

Example:

```
PI = 3.14159              variable equals constant

Fed_ID$ = "31-555642"     variable equals constant

A = A + 1                 variable equals expression

C = A + B                 variable equals expression

D$ = A$                   variable equals variable
```

Numeric Data

Numeric data is operated and stored in binary integer, Binary-coded-decimal (BCD) or base 10,000 (decimal). The valid range for numbers is approximately 10^{-64} thru 10^{63} with 20-digit precision. All arithmetic calculations are performed to this degree of accuracy, although results may be truncated depending on the type of variables used and its precision. Numeric values supplied in statements are referred to as numeric constants.

Very large or small numbers are expressed using floating-point E-notation (scientific notation).

E-notation is used for output whenever a number's decimal point does not lie among its 16 most significant digits. Numeric data may be entered using E-notation at any time.

For example, the large value: `13429178952112216`

is output as: `1.342917895211222E+16`

and is read "One point three four ... times ten to the sixteenth power".

The small value: `.00000000000000000034`

is output as: `3.4E-19`

and read as "3.4 times ten to the negative nineteenth power."

Numeric Precision

Several numeric data representations are supported, with differing representation, accuracy and

performance. The ten numeric p**Bold text**recisions determine the storage representations and the valid range of values for all numeric variables.

Prec %	Data Type	Bytes req'd	Significant Digits	Range of values supported by precision
1	Integer	2	5	± 32768
2	Integer	4	10	± 2,147,483,648
3	Decimal float	6	9-12	± .999999999999 E±63
4	Decimal float	8	16	± .9999999999999999 E±63
5	Decimal float	4	6	± .999999 E±63
6	Decimal float	12	17-20	± .99999999999999999E±63
7	IRIS BCD 1%	2	4	± 7999
8	IRIS BCD 2%	4	6	± .999999 E±63
9	IRIS BCD 3%	6	10	± .9999999999 E±63
10	IRIS BCD 4%	8	14	± .99999999999999 E±63
11	IMS BCD 2%	4	6	±.999999 E±63
12	IMS BCD 3%	6	10	± .9999999999 E±63
13	IMS BCD 4%	8	14	± .99999999999999 E±63

The default precision for variables is based upon the type of program running. IRIS programs default internally to %5 (2-word floating), while BITS programs default to %4. Newly created BITS programs may specify any of the above precisions in a **DIM** or **COM** statement.

IRIS programs may specify one of 4 precisions in the form 1%, 2%, 3% or 4%. These precisions map to %1, %5, %3, and %4 respectively. When the environment variable **BCDVARS** is enabled, the precisions map to %7, %8, %9 and %10 forcing all variables to be processed in BCD. This option is only required in applications performing unique processing of internal BCD formats (such as indiscriminate moving of data between numeric and string variables using **CALL 72/73**).

During file access, variable precisions are internally changed as data is read or written between IRIS BCD files and other integer or Base 10000 data files. This process eliminates conversion of numeric data during **READ** and **WRITE**.

See also: IRIS BCD Files

Proper selection of variable precision is required when memory space is limited. For example, a 1,000 element array using Double-precision %4 requires 8,000 bytes of program space (1,000 X 4 words X 2 bytes per word). The same array using one word per element (%1) requires only 2,000 bytes. It is best to choose precisions based upon the worst-case data you expect to place in the variables. Precision affects the amount of bytes required in data files to hold a given variable during normal **READ** and **WRITE** operations.

Special Notes on %3 and %6 Numerics

The number of significant digits retained by %3 and %6 varies depending upon the number of integer versus fractional digits being represented. To determine whether the precision can

correctly represent a specific number, locate the required number of integer or fractional digits in the first column. The second column then gives the maximum number of digits for the other (fractional or integer).

Accuracy limitations using %3 format:

1	8	7	4
2	8	8	4
3	8	9	0
4	8	10	0
5	4	11	0
6	4	12	0

Accuracy limitations using %6 format:

1	20	11	12
2	20	12	12
3	20	13	8
4	20	14	8
5	16	15	8
6	16	16	8
7	16	17	0
8	16	18	0
9	12	19	0
10	12	20	0

The %6 form is the most speed-efficient of all floating-point representations but also requires the most memory space.

Integers Stored in Floating-Point Variables

If, when a value is packed into %3, %4, or %6 form, the value is within the double-precision signed-integer range, word 0 is cleared and the value is instead stored into words 1 and 2 in %2 integer form If two such values are operated upon, integer arithmetic is used, which can be performed faster than floating-point arithmetic. If the result value is again within the %2 range, it will be packed as such when stored back into a variable.

Integer arithmetic is not performed if:

Either operand is in floating-point form

or

A divide operation is performed

The adjustment between integer and floating-point arithmetic is totally user-transparent. However, use of integer arithmetic greatly enhances the net speed of program calculations, many of which are integer-type operations (A=A+1, etc.).

String Data and Literals - "str.lit"

A string is defined as a sequence of zero or more ASCII characters. Strings range in length from 0 to 65534 bytes (characters). Strings within programs are enclosed in double quotes and referred to as string constant *str.lit*. A zero byte is used internally to denote the logical end of a string.

Each *str.lit* is governed by the following rules:

1. The *str.lit* must begin and end with double quotation marks (").
2. Any character may be expressed using its octal ASCII value enclosed within backslashes, for example "\215\". Non printable and special control characters that perform an immediate keyboard function (such as backspace) must be entered in this fashion to be included as data.
3. All printable characters represent themselves except \ (backslash).
4. Each \334\ is replaced with a single backslash.
5. Each pair of single quotes (' ') are replaced by a single double quote (").

See also: ASCII Codes and Input Character Processing

CRT Mnemonics and Expressions - crt.expr

CRT mnemonics and expressions, *crt.expr*, are used in conjunction with a CRT term file to provide control of video terminal functions such as *clear-screen*, *reverse-video*, etc. CRT mnemonics appear in one of two forms:

- A set of one or more 2-character codes enclosed in single quotation marks ('). Each code can be preceded by an optional count value.
- A cursor address in the form: @*num.expr*, *num.expr;*. Addresses are given in the form *column*, *row* from origin 0,0 home (upper-left of screen).

For example:

```
'CS'              Clear screen

'CS10ML'          Clear and move left 10 positions.

@5,5;'CL'         Position to column 5, row 5 and clear line

@10,L;            Position cursor to column 10, row L.

'BG'"\107\"'EG'   Output a graphics sequence.
```

See also: Using Dynamic Windows, Terminal Translation: CRT CODES $TERM Files for a complete discussion on defining your terminal for use with Windows, Mnemonics, Cursor Positioning and Extended Graphics.

Statements, Statement Numbers & Labels

All BASIC program instructions are called *statements*. They have the general form:

stn {label:} statement { \ statement }

where:	*stn*	is a valid statement number 1 to 99999999.
	label:	is a valid statement label followed by colon.
	statement	is any valid BASIC statement.
and	*{\...}*	is the separator for multiple statements.

Immediate Mode

Any BASIC statement entered without a stn is executed immediately. This type of operation is termed *immediate mode* and provides for interactive debugging, calculator, or single-step operations. Most statements may be executed in *immediate mode*; some cannot simply because of their nature. For example, **FOR** without a matching **NEXT** is prohibited. Each statement documented indicates whether it is available in *immediate mode*.

Statement Numbering

Each line begins with a statement number (*stn*) and ends with the **[EOL]** end of line character. The *stn* must be an integer in the range 1 thru 99999999 and is used to indicate where within the program to insert the line.

Following the *stn* may be a statement *label*. The *label* may be from 1 to 32 characters in length consisting of letters, digits, and underscore. A *label* must begin with a letter or underscore and end with a colon.

Throughout this guide, *stn* is used to indicate selection of either a statement number or label. If a *label* is not explicitly defined for a statement, the *stn* is considered both the statement number and label.

A *statement* is one instruction to be executed by the computer, such as printing a list of values. A program line is a line consisting of one or more BASIC *statements*.

Program lines may only be entered while in BASIC *program mode*. Program lines may be entered in any order. They are sorted automatically into ascending *statement number* order. A stn is always required when entering or changing a statement, even if the statement includes a *label*.

For example, the following lines assign values to variables. Spacing between keywords and around variable names is required if **LONGVARS** or **VARIABLE** modes are set to accept long variable names. If long variable names are not enabled, the system will accept statements without regard to spacing:

```
5A=0

10 LET A=10

20LETA=10
```

```
30 ASSIGN_VALUES: LET ZERO_VALUE=7
```

Let is assumed if not given, as in example line 5. If long variables is not enabled, line 20 is identical to line 10. If enabled, the variable name "LETA" is assigned the value of 10. The actual statement in this case would be "LET LETA=10."

See also: LET statement

Multiple-Statement Lines

Several BASIC statements may appear following a single *stn*.. Each statement is separated by a \ and termed a *sub-statement* . Sub-statements are numbered on each line starting with 1 and are identified as a *sub-stn*. For example:

```
100 PRINT TOTAL;J \ GOTO 140
```

When using multi-statement lines, certain programming effects must be noted. Conditional branching (**GOTO**, **GOSUB**, **ON**) may only select the first sub-statement of any line. Branching to sub-statements (other than the first) is only provided by the **JUMP** statement. Refer to the following statements for further considerations:

DATA	**ERRSTM**	**ESCSTM**	**GOSUB**
IF ERR	**JUMP**	**ON**	**GOTO**
REM	**RETURN**	**IF**	

Inserting, Changing & Deleting Statements

Insertion of new program lines is accomplished by selecting a new *stn* between two existing *stn's* . For example, to insert a new line between 10 and 20 above, select a *stn* from 11 to 19 such as:

```
14 LET Q=16
```

Fractional *stn's* are not allowed. The entire program may be renumbered as necessary using the **RENUMB**er command.

To **replace** an existing statement, simply enter the *stn* to replace followed by the new BASIC statements. The new line replaces the existing.

```
30 LET Z=7

30 LET Z=6 replaces LET Z=7
```

To **modify** part of an existing line, use the **EDIT** command. Simple changes, insertions or deletions are easily performed without re-typing the entire line. In addition, **EDIT** may be used to correct a line entered with an error.

To **delete** an existing program line, type the *stn* only, and press **[EOL]** (usually return). This process deletes one program line at a time:

```
20
```

Multiple lines are removed using the **DELETE/ERASE** commands. To delete all lines of a program, use the **NEW** command.

Examples:

```
OPEN #0, "$LPT", #3, "PAYROLL"

DIM A$[100],R$[100],3%,DATA_ARRAY[32]

SEARCH #3,3,1;A$,R1,E \ PRINT R1,E

READ #3,R1;R$ \ MAT READ #3,R1,104;DATA_ARRAY
```

See also: Statements and Calls

Variables

BASIC is an algebraic language, with data values operated upon and stored in storage areas called *variables* or *vars*. In UniBasic there are two types of *variables*. The first is a numeric variable and the second is a string variable.

Variable Naming Conventions

In UniBasic there are two types of variable names; a *short var* and a *long var*. The default is the short form: *letter* or *letter+ digit* for numeric variables, and *letter* or *letter+ digit +$* for a string variable. Any variable ending with a dollar sign is automatically recognized as a string variable.

To use the long variable names, the global environment variable **LONGVARS** is set, or you may issue the command: **VARIABLE +**. A long variable is named by a *letter* followed by up to 31 additional characters which may be *letters*, *digits* or *underscore*.

Lower-case letters are equivalent to their upper-case counterparts. Some examples of variable names include:

```
A    B0   DATA_VALUE

A$   B0$  PHONE_NUMBER$
```

By default, up to 348 different variable names may be used within each program. This value may be restricted or increased through the use of the environment variable **MAXVARS, which defaults to 348**. When you enter a program statement that includes a previously unused variable name, the variable count is compared to **MAXVARS**. If the definition of this new variable will exceed the limit, the following error is displayed:

```
Too many variables defined
```

Once a variable name is in the internal variable table, it is not removed even if all occurrences of its use are removed. A program must be dumped to ASCII form and re-loaded (see the **DUMP/LOAD/GET** commands) in order to release unused variable names. To increase the number of variable names beyond 348 (to 1113), the **MAXVARS** environment variable must be set to "extended". The number of variables will only be increased in newly created programs. To increase the number of variables in an existing program beyond 348, the program must be dumped to text and then reloaded while **MAXVARS** is set to "extended".

If you exceed the number of variable names allowed, use the **VARIABLE** command to locate one or more variables that could be removed from the program. Manually (or using an editor), remove all occurrences of the deleted variables. Next, **DUMP** the program to text, perform a **NEW**, and finally reload (**LOAD** or **GET**) and resave the program.

Subscripted Variables

Certain variables permit the use of a numeric subscript. In the second example below, *subscript* defines the beginning byte of a variable, while *subscript2* defines the ending byte of the subscript. A subscript is given in the form:

[subscript]

or

[subscript1, subscript2]

These *subscripts* may be any numeric expressions which, following evaluation, are truncated to integers. *Subscripts* are allowed on numeric variables, arrays and matrices, and string variables. An error is generated if a supplied *subscript* is outside the range of the variable referenced.

Arrays and Matrices

An array is a list of numeric data elements. A matrix is a two-dimensional table. Array and matrix elements are numbered origin zero for selecting individual elements. Therefore, an array dimensioned [10] actually contains the 11 elements [0] thru [10].

Matrices also have row and column zero. The 4 X 4 matrix shown above contains the 25 elements:

```
[0,0]   [0,1]   .   .   .   [0,4]
•
•
[4,0]   [4,1]   .   .   .   [4,4]
```

The example below shows a four element array (list) and a 16 element matrix (4 by 4):

Array[4]	Matrix [4,4]				
0	0	0	0	0	0
1	0	1	2	3	4
2	0	5	6	7	8
3	0	9	10	11	12
4	0	13	14	15	16

Note: Most **MAT** statements do not operate on row and column zero elements; they use origin one. So, for the purposes of matrix arithmetic, a 4 X 4 matrix actually has 16 usable elements. The **MAT READ** and **MAT WRITE** statements do transfer row and column zero.

Numeric, Array and Matrix Variables

A numeric variable is one of three types: simple, array, or matrix.

A simple numeric variable *var* or *num.var* is one that will store a single numeric value.

For example: A B4 INPUT

An array variable *array.var* may contain many values, which are operated upon either as a whole (**MAT**), or individually by selecting a single element or *subscript*. The *subscript* addresses a single array element by its number (0-n).

For example: A[3] B4[36] INPUT[0]

A matrix variable *mat.var* may also contain many values, which are operated upon either as a whole (MAT), or individually by selecting two separate *subscripts*. The two *subscripts* together address a single matrix element by its position, i.e. row and column number (0-x,0-y).

For example: X[9,2] B4[15,28] INPUT[0,10]

All *subscripts* are origin zero. If an *array.var* or *mat.var* is referenced without subscript, each missing *subscript* defaults to zero (excepting **MAT** Statements defined to operate upon the total variable).

For example: If A is an array, then A = A[0].

If B is a matrix, then B = B[0,0] and B[x] = B[x,0].

In most other contexts, the terms *array* and *matrix* are used interchangeably. In this guide, we will restrict the usage of *array* to indicate one-dimensional and *matrix* to indicate two-dimensional.

Automatic Dimensioning Numeric Variables

A variable's type and precision are selected when dimensioned, either explicitly (**DIM** or **COM** statements), or implicitly by its initial usage, termed Auto-Dimensioning. All auto-dimensioned variables take on the default or last specified precision from a **DIM** or **COM** statement. A simple num.var is auto-dimensioned to hold a single value. An *array.var* is auto-dimensioned to hold 10 elements, and a *mat.var* to hold [10,10] elements. All numeric variables are initialized to zero when dimensioned.

```
LET A=0
```

performs an automatic dimension of A to a simple variable at the current precision.

```
LET A[6]=0
```

performs an automatic dimension of A[10] at the current precision.

```
LET A[6,3]=0
```

performs an automatic dimension of A[10,10] at the current precision.

Re-Dimensioning Numeric Variables

Once any *num.var*, *array.var*, or *mat.var* is defined through explicit **DIM** or **COM**, or automatic

dimensioning, its precision cannot be changed. When a matrix variable used in a **MAT** statement includes *subscripts*, the *subscript* values are interpreted as a new working size for the selected matrix. This new size can not require more total elements than the original dimension. For example, a matrix originally dimensioned as [10,10] has 121 elements. Some examples of legal new working sizes would be:

```
[50,1] [2,40] [40,2] [20,4] [3,3] [7,6] . . .
```

The new working space will now remain in effect for the remainder of the program, or until changed again. A change in working size does not affect variable precision, or file access statements.

If you attempt to re-dimension a two-dimensional array (matrix), to (-1,-1) a subscript error is reported.

String Variables

Variables used for string data are denoted by a dollar sign following the variable name.

A$ D5$ X0$ DATA_VALUE$

A string variable *str.var* must be explicitly dimensioned before it may be referenced in statements in a program. A *str.var* can be dimensioned only once, by using a **DIM** or **COM** statement . The dimensioned size represents the maximum size in bytes (characters) allowed for the variable. A *str.var* may also be passed from one program to another using **CHAIN READ**, in which case it may not be included within a **DIM** or **COM** statement.

A *str.var* is initialized with all zero bytes when dimensioned, and so has a logical length of zero.

See also: LEN function

A *str.var* may contain any ASCII Characters. Each *str.var* is terminated by the ASCII character \000\. The logical length of any *str.var* is equal to the number of characters from a starting position up to, but not including the terminator.

Subscripted Strings

String *subscripts* are used to access certain portions of a string by position. String positions are numbered starting at 1. String *subscripts* may be any numeric expressions that, when truncated to integers specify character positions between and including 1 and the dimensioned length of the *str.var*.

A *str.var* given by its name alone (B$) refers to the entire string, from the first position up to the first zero byte.

A *str.var* given with a single subscript (B$[14]) refers to all bytes from the starting position up to the first zero byte.

A *str.var* given with two subscripts (B$[14,22]) refers to all bytes between and including the two positions selected up to the first zero byte. Therefore, two equal subscripts (B$[8,8]) specify a single byte position within the string.

A *str.var* may also contain binary information including zero-byte terminator characters. Certain statements are provided for manipulation of binary strings, **CALL $STRING**, **MAT**, **CONV**, **ASC** and **CHR**. These functions and statements may be used to operate upon an entire string or *substring*. The **LEN** function may not be used with binary strings since the first zero byte is considered a terminator.

String Arrays

String arrays are not directly supported, but can be emulated using formulated subscripts. For example, if a string array, A$, is to contain N strings of L characters each, the required dimension is:

```
DIM A$[N*L]
```

and any given element E of the array can be accessed by:

```
A$[E*L+1,E*L+L]
```

Dimensioning String Variables

String variables must be declared in a **DIM** or **COM**, or **CHAIN READ** statement. Attempting to use a string variable not previously dimensioned produces an error. No auto-dimensioning of string variables is supported.

Re-Dimensioning String Variables

Once a *str.var* is defined, its size may not be changed. Any attempt to dimension the variable to a smaller or larger size results in an error. A re-dimension of the same size is permitted, without an error.

Expressions

There are two types of expressions: numeric expressions and string expressions. A numeric expression ***num.expr*** is considered any group of numeric variables, constants, functions, and/or operators returning a numeric result. A string expression *str.expr* is considered any group of string variables, constants, functions, CRT expressions and/or operators to be concatenated (linked together) returning a string result. Any statement may incorporate the use of string or numeric expressions as long as the final result matches the format of the statement, or the variable chosen to store the result.

Operator Precedence

Expressions are evaluated according to the precedence documented in the Operator Precedence Table. Operators on the same level are evaluated from left to right in the expression, however parentheses can be used to override this hierarchy.

Operator Precedence Table

(highest)	+ - and Functions	Unary + - and FUNCTIONS evaluated R-L
	^	Exponentiation Left-to-Right
	* /%	Mult, Divide, Modulo Left-to-Right
	+ -	Add, Subtract Left-to-Right
	TO	String TO string Left-to-Right
	USING	number USING string Left-to-Right
	, +	String concatenation Left-to-Right
	< <= > >= <>	expr relation expr Left-to-Right
	AND	relation AND relation Left to Right
(lowest)	OR	relation OR relation Left to Right

For example:

EXPRESSION	EVALUATES AS	RESULTS WITH
3+4*5	3+(4*5)	23
(3+4)*5	(3+4)*5	35
14/7*10/2	((14/7)*10)/2	10
3^2*4	(3^2)*4	36
"3"+"B"	3 concatenate B	3B
'CSBP'+'BU'	CS BP BU	Clear Screen, begin protect & underline

Functions are evaluated before any arithmetic operations are performed.

Predefined BASIC Functions

Many built-in functions are included which can be used within numeric or string expressions. Functions produce a result based upon a given value, termed an *argument*. The result, as well as the *argument* can be string or numeric depending on the function in question. A function's general form is:

FUNCTION *argument*

where **FUNCTION** is the three-letter function name, and argument is the variable or expression to be operated upon.

Note that the *argument* may or may not be enclosed within parentheses. Parentheses are only required when the argument is itself an expression, as functions are evaluated on a higher precedence than other arithmetic operations.

For example:

```
100 LET A=ABS X+2
```

In this case, the ABS function is evaluated before the 2 is added. The statement:

```
100 LET A=ABS(X+2)
```

performs the addition before applying the function.

The function itself can appear within another expression, provided its result is compatible with the surrounding expression, e.g. a function producing a numeric result is invalid within a string expression.

```
B$+INT(X)+C$
```

is by itself invalid unless the numeric result of the function **INT** is cast into a string result, for example:

```
B$+STR(INT(X))+C$
```

All pre-defined functions are documented below in alphabetical order. The first column identifies the function name (**ABS**, **TAN**, etc.), the second defines the argument type (string/numeric), and the third the result type. The function's operation is then described at the right.

Name	Arg	Res	Operation
ABS	num	num	Absolute Value of the argument.
ASC	str	num	ASCII value of specified character in string. Characters are toggled and returned in BITS/IRIS 8-bit format unless Binary Input mode is enabled by **SYSTEM** statement or '**IOBI**' mnemonic.
ATN	num	num	Arctangent in radians.
CHN	num	num	Same as **CHF**.
CHF	num	num	Various parameters of an open file or device. The argument must be the channel number (0-99) of an open channel plus a constant greater than 100 to select mode. The Channel Modes are shown in the following table, and 'xx' refers to the desired channel number.
	0xx	num	Total number of records contained within the file. This value can be used also as the first record number not contained in a file. For Contiguous files, this is the larger of the initial number of records specified in **BUILD/CREATE** or the current number of records. If the file has a *First Real Data Record*, that value is included in this size.
	1xx	num	Record number of current file position. For an item file, mode 100 yields the last record number written.
	2xx	num	Byte displacement into record of current file position.
CHF	3xx	num	Record Length in words for IRIS Applications, or (0) representing the channel status word for BITS Applications.
	4xx	num	Memory location of UniBasic **T_chan** structure for this channel.
	5xx	num	Open File's record length in bytes.
	6xx	num	unused, returns 0.
	7xx	num	unused, returns 0.
	8xx	str	Filename of file opened on channel.

CHR	num	str	Supplies the ASCII character selected by the argument value for BITS applications. The argument is supplied in IRIS/BITS 8-bit format and toggled to conform to the internal character representation. If Binary Output is enabled **SYSTEM** statement or '**IOBO'** mnemonic is in effect, no toggling is performed.
CHR	num	num	Returns the 'characteristic' value for IRIS applications. This is an integer exponent X such that: 10^{X-1} <= argument < 10^{X}.
COS	num	num	Cosine in radians.
DET	---	num	Determinant of the last matrix inverted. See the **MAT INV** statement.
ERR	num	num	Various values pertaining to error, ESCape and interrupt branching. When using this function within IRIS programs, the *argument* must be parenthesized to prevent misinterpretation as an **IF ERR** statement. The argument selects:
	0	num	Last error number in BITS error format
	1	num	*stn* of last BASIC error.
	2	num	*stn* of last ESCaped statement.
	3	num	*stn* of last interrupted statement.
	4	num	*sub.stn* of last error, ESC, or interrupt.
	5	num	*sub.stn* of last BASIC error.
	6	num	*sub.stn* of last ESCaped statement.
	7	num	*sub.stn* of last interrupted statement.
	8	num	Last Index File Structure error identifier.
ERM	num	str	Supplies the selected message from the user message file currently selected Returns null if no user message file is selected. See **CALL 40**.
EXP	num	num	Exponential, the constant e to the power given (e^{X}).
FRA	num	num	Fractional portion of argument. For example: FRA(4.5) yields 0.5.
INT	num	num	For a **num.arg** returns the greatest integer less than or equal to the argument. For example: INT(4.5) yields 4, while INT(-4.5) yields -5.
INT	str	num	For a *str.arg*. returns the ASCII value of the first character in the string. This is functionally identical to the **ASC** function.
IXR	num	num	Integer radix base 10 of the argument. For example: IXR(1000) returns 3.
LEN	str	num	Length of string in characters. Length is computed from optional starting *subscript* to first zero-byte terminator.

LOG	num	num	Logarithm base e of the argument. Logarithm in any base B can be achieved using the theorem: $\log_B X = \log_e X / \log_e B$.
MAN	num	num	Decimal mantissa of the argument in base 10.
MEM	num	num	Supplies data from the selected location in main memory; presently this function returns 0.
MSC	num	num	Miscellaneous numeric functions. The argument selects the value returned; -1 returned for unimplemented functions:
	0	num	Your current Port number.
	1	num	Logical input element last accepted.
	2	num	UniBasic revision level.
	3	num	*stn* of last **GOSUB** instruction. Value is returned and removed from the stack.
	4	num	Reserved for future use.
	5	num	Current screen tab column counter.
	6	num	Current unused variable space.
	7	num	Returns the environment variable **MSC7**, or the Unix Group number * 256 + User number if **MSC7=65535** or is undefined.
	8-17		Reserved for future use.
	18	num	The constant PI (3.141592653589793).
	19	num	The constant e (2.718281828459045).
	20	num	Maximum channels per user; returns 64.
	20-29		Reserved for future use.
	30	num	*stn* of current BASIC statement.
	31	num	*sub.stn* of current BASIC statement.
	32	num	*crt_type* value from current *term*. file.
	33	num	Number of columns in the open window.
	34	num	Number of rows in the open window.
	35	num	Size of environment variable **INPUTSIZE**.
	36	num	Reserved for future use.
	37	num	Maximum port number supported.
	38	num	Number of Total Users.
	39	num	European date flag.
	40	num	*max-x* value from current term. file; Number of columns for your CRT
	41	num	*max_y* value from current term. file; Number of rows for your CRT

	42	num	Window nesting level. Number of open Windows. On an ANSI monitor, a default of 1 window is always opened.
MSF	num	str	Miscellaneous string functions. Argument selects the value returned:
	-1	str	UniBasic revision 8-character string. 5.8.2.3 returns 05080203, 5.3 returns 05030000.
	0	str	System date and time in international format: dd mon year hh:mm:ss
	1	str	Current working directory path
	2	str	Text description of last BASIC error.
	3	str	System date and time in IRIS/US format: mon dd, year hh:mm:ss
	4	str	Path and filename of the current BASIC program loaded into memory. If the returned string does not begin with '/', the program name is relative to your current working directory. The full name must be assembled by concatenating MSF(1) and MSF(4).
	5	str	Returns the name of the parent BASIC program, when the current program was invoked by SWAP.
NOT	any	num	Logical NOT. Returns 1 if argument is zero or null, or zero if not.
RND	num	num	A pseudo-random number X is generated in the range 0 < X < argument.
			See also: RANDOM statement for more on pseudo-random numbers.
SGN	num	num	**Signum** function. Returns the sign of the argument, where:
		-1	if argument < 0
		0	if argument = 0
		1	if argument > 0
SPC	num	num	Special numeric functions used by IRIS applications. The argument selects the value returned, or a -1 is returned for unimplemented functions:
SPC	0	num	CPU time used this session in tenth-seconds.
	1	num	Connect time used this session in minutes.
	2	num	Hours since a base date of 1980. This value is computed assuming all months have 31 days.
	3	num	Current tenth-second of the hour.
	4	num	UniBasic revision level.
	5	num	Returns the environment variable **SPC5** or the Unix Group number * 256 + User number if **SPC5=65535** or is

			undefined.
	6	num	Your current Port number.
	7	num	Returns the environment variable **SPC7**.
	8	num	Last BASIC error number in IRIS format.
	9	num	Current *stn* being executed.
	10	num	*stn* where last BASIC error occurred.
	11	num	Current Logical Unit number. The last directory name in the current working directory is returned as a number.
	12	num	Logical Unit number of the current program. The last directory name in the current programs pathname is returned as a number.
	13	num	*crt_type* value from current *term*. file.
	14	num	*stn* of last **GOSUB** instruction. Value is returned and removed from the stack.
	15	num	Return and clear the last BASIC error number in IRIS format.
	16	num	*stn* of last **GOSUB** statement. Value is returned and left on the stack; non-destructive read, whereas **SPC 14** is destructive.
	17	num	Length of last character-limited **INPUT**.
	18	num	System base year; always returns 1980.
	19	num	UniBasic License Number in decimal.
	20	num	System base year; Returns the default 1980 or the value of the Environment Variable **BASEYEAR**.
	21	num	Length of the input buffer environment variable **INPUTSIZE**.
	22	num	Available program space in words. Returns a large constant to reflect virtually unlimited space.
	23	num	Return the current library logical unit number. A -1 is returned if no current library, or if it is non-numeric.
	24	num	Statement number stn of last **END**, **STOP** or **SUSPEND** statement.
SQR	num	num	Square root function. Returns the square root of the argument. An error is generated if the argument is negative.
STR	num	str	Convert the numeric value into a string result. No leading or trailing spaces are provided.
TAB	num	str	Return the required number of spaces terminated by a zero byte to move the terminal to the column specified by the argument.
TAN	num	num	Tangent of the argument returned in radians.

TIM	num	num	Returns various Time functions as numeric values. The argument specifies the function to perform:
	0	num	CPU time used this session in seconds.
	1	num	Connect time used this session in minutes.
	2	num	System real-time hours since base date. Normally adjusted using a base year of 1980. To change the value returned, see the environment variable **BASEYEAR**.
	3	num	Current tenth-second of the hour.
	4	num	Current date in the form: MMDDYY where MM is the month (1-12), DD is the day of the month (01-31) and YY is the year such as 89.
	5	num	Current date in the form YYDDD where DDD is the day of the year (1-366).
	6	num	Number of days since 0 January 1968.
	7	num	Current day of week (0=Sunday, 6=Saturday).
	8	num	Current year in the form YY, such as 89.
	9	num	Current month; 1=January, 12=December.
	10	num	Current day of the month ; 1-31.
	11	num	Current hour of the day; 0-23.
	12	num	Current minute of the hour; 0-59.
	13	num	Current second of the minute; 0-59.9.
	14	num	Current date in the form: MMDDYYYY where MM is the month (1-12), DD is the day of the month (01-31) and YYYY is the year such as 2001.
	15	num	Current date in the form YYYYDDD where DDD is the day of the year (1-366) and YYYY is the year such as 2001.
	16	num	Current year in the form YYYY, such as 2001.
VAL	str	num	Convert the string argument to a numeric value. An error is generated if the argument is null or does not contain a valid numeric value.

See also: DEF FNx for information on custom User-Defined functions within a program

Operators Used in Expressions

Several classes of operators are provided for use within expressions. Operators are evaluated either right to left, or left to right and have a strict evaluation precedence. Parenthesis may be used to change the precedence of an operation.

Unary operators + -

Arithmetic operators	^ * / % + -
Relational operators	< <= > >= = <>
Concatenation operators	+ ,
Boolean operators	AND OR
String operators	USING TO

Parenthesis may be used to override the default evaluation order of any expression.

Unary Operators + -

The unary operators (+ -) are used to change the sign of an *argument*. They are evaluated Right-to-Left and have the highest precedence. The + is a non-operation, and the - changes a negative value positive or a positive value negative.

Arithmetic Operators ^ * / % + -

Arithmetic operators follow unary operators in the precedence of an expression. The highest precedence is given to **(^)** invoking exponentiation, which is essentially repeated multiplication. A value y^x is read, "take the value y raised to the power x." In simpler terms, multiply y by itself x times. Exponentiation has the highest precedence of all of the arithmetic operators and is evaluated Left-to-Right.

Next, **(* / %)** which selects multiplication, division and mod. The mod operator **%** returns the remainder of a division of the two operands. This is calculated as (x - INT(x/y)*y). 10%2 yields 0, 10%3 yields 1, etc. These operators are evaluated from Left-to-Right after exponentiation.

Finally, **(+ -)** addition and subtraction are the lowest precedence of the arithmetic operators. These are also evaluated from Left-to-Right.

Concatenation Operators + ,

Concatenation operators are used to link string expressions together. The result of concatenating two string expressions is the combination of both expressions into a single string expression. Each concatenated string is appended to the end of the current expressions result. "This" +" That" results in the string: "This That", etc.

The concatenation operator **(+)** may be used in any expression involving strings, and IRIS programs may also use the **(,)** concatenator in **LET** and **IF** statements.

Relational Operators = <> > >= < <=

All relational operators are evaluated on an equal precedence and all group Left-to-Right. Their result is said to be True (one) if the relation is true, and False (zero) if the relation is false. Relational operators can be used in **IF** statements or as part of a boolean expression. The format is: *expression* ***relation*** *expression*, where ***relation*** can be any of the following:

=	Equal
<>	Not Equal
>	Greater Than
>=	Greater Than or Equal To
<	Less Than
<=	Less Than or Equal To

When relationals are used for numeric comparisons, it is easy to understand that the comparisons are strictly based upon the numeric values compared. All comparisons are made using the same 20-digit significance as printed. No additional hidden digits interfere causing printed values to differ from internal representation as is typical with systems utilizing binary instead of decimal floating point operations.

When relationals are used in Boolean expressions, they result in a numeric result of one if the relation is true, and zero if the relation is false.

String variables and literals are compared using the ASCII code of each character, one character at a time. If the strings are not subscripted to control their length, then they are evaluated using the current logical length (from any optional starting position up to the first zero-byte terminator). Strings are equal only when they are exactly equal in length and contents. When a shorter string is compared to a longer one, and they are equal up to the length of the shorter string, the shorter string is said to be less than the longer string. If, during comparison, two characters do not match, the left string is said to be less than the right string if the ASCII code of the mismatched character is less than the ASCII code of the right strings character.

See also: Appendix A for a complete list of ASCII codes and their numeric values

Boolean Operators AND OR

The Boolean operators **AND/OR** are processed Left-to-Right and are used to compare several relational expressions together. **AND** has a higher precedence than **OR**. The format of these operators is: *expression* **AND** *expression*, or *expression* **OR** *expression*. The result is true (one) for AND if both *expressions* are true, or true (one) for OR if either *expression* is true.

String Operator USING

The **USING** operator groups Left-to-Right and results in a formatted string result from a numeric expression. The format of this operator is:

numeric expression **USING** *string expression*.

The *numeric expression* is evaluated first. Next the *string expression* is evaluated and used to 'format' the *numeric expression* into a string result.

The format string is scanned, and any characters which are not *field descriptors* are copied to the destination until a *format field* is seen. Characters which can begin a format field are: **$ # + - & ***. Other field descriptors (, ! CR DB) are treated as text and are copied until a starting character is seen. After formatting a result, the remaining characters in the *format* string (up to the start of another format field) are copied to the destination.

Each *format* field is made up of certain characters describing the formatting to be done. These

are called field descriptors. Numeric items are formatted according to the rules governing each descriptor. If an item cannot be formatted according to the field given, the field is output filled with asterisks (*). This generally occurs when a number is too large to be expressed with the number of digits available in the field.

Field Descriptors

Field descriptors for a *format* field fall into seven categories:

- Leading characters
- Floating characters
- Numeric Characters
- Commas
- Decimal Points
- Post Sign
- Numeric Split

Leading Characters

A field can begin with one or two leading characters. The available leading characters are:

LEADING	OUTPUT	
$	$ always	
+	+ if item >= 0;	- if item < 0
-	space if item >= 0;	- if item < 0

The **$** can be combined with either + or - for a two-character leading group. Note that all three leading characters are also valid as floating characters. A group of two or more identical characters is considered a floating character designation. You can change the character output of the **$** leading character by setting the environment variable **CURRENCY** to any printable character. You will still use the **$** (or its ASCII equivalent) for your programming.

Floating Characters

A field can contain groups of floating characters. This character 'floats' and is eventually executed just before the first digit output. The available floating characters are the same as the leading characters ($, +, -) and are processed the same.

Note: Numeric formatting outputs a sign (+ or -) only if one is specified within the format field. If none is given in the format, all items are output as positive, regardless of sign.

One extra floating character should be given in the format field in addition to the number given for the highest digit count desired. One space is required for the execution of the floating character itself. The remaining floating characters can be occupied by digits. For example, the format string "$$$$" can accommodate no number larger than 999, as one space is required for the dollar sign itself.

Numeric Characters

A field can contain groups of numeric characters The available numeric characters are:

SYMBOL	CHARACTER
#	Digit or space if leading zero
&	Digit, leading zeroes not suppressed
*	Digit or "*" if leading zero

Every numeric character given in a format field can contain a digit. For example:

```
Format:      ####    &&&&    ***#
               17    0017    **17
             247     0247    *247
             6140    6140    6140
               0     0000    ***0
```

Commas

A field can contain one or more commas which are output when significant. For example:

```
Format:      ##,###    #,###,###    &,&&&,&&&
                768          768    0,000,768
              2,147        2,147    0,002,147
             ******    1,034,957    1,034,957
```

Both the programming and output of commas and decimal points is controlled by the environment variables: **EURINPUT**, **EUROUTPUT**. These parameters let you change the programming and output style respectively of comma and decimal point fields. You may set either or both parameters for your desired effect.

See also: the Environment Variable: **CURRENCY**.

```
EURINPUT=1     ##.###    #.###.###    &.&&&.&&&
EUROUTPUT=1    2.147        2.147    2.034.957
```

Decimal Points

A field can contain a period for an item's fractional portion. The fractional portion will then follow and be truncated to the number of digits specified. Only numeric descriptors (#&*) can follow the period, and all are processed as a character. For example:

Format:	##.###	##.#	##.&&	**.**
	74.000	74.0	74.00	74.00
	16.408	16.4	16.40	16.40

Both the programming and output of commas and decimal points is controlled by the environment variables: **EURINPUT**, **EUROUTPUT**. These parameters let you change the programming and output style respectively of comma and decimal point fields. You may set either or both parameters for your desired effect.

See also: the Environment Variable: **CURRENCY**

EURINPUT=1	##,###	##,#	##,&&	**,**
EUROUTPUT=1	16,408	16,4	16,40	16,40

Post Signs

Post signs are only applicable to BITS programs. A field can be terminated with a post sign designator. The post signs are:

Sign	output if item >=0	output if item <0
+	+ if item >= 0	- if item < 0
-	space if item >= 0	- if item < 0
DB	DB if item >= 0	CR if item < 0
DR	DR is item >= 0	CR is item < 0
CR	two spaces if item >= 0	CR if item < 0

Format:	+##.##+	##.##-	##.##CR	##.##DB
	+47.24+	47.24	47.24	47.24DB
	- 6.27-	6.27-	6.27CR	6.27CR

A sign can be output before and after an item. Page numbers using the field ---&- are output as -#- if the page numbers are made negative. For example, page number can be -7- or -10-.

Numeric Split

Numeric Split is only applicable to BITS programs. A numeric item, such as a part number, date or government Social Security Number, can be separated automatically (without dividing into

separate numerics). The descriptor ! causes a - to be output when significant. For example:

```
Format:   &&&!&&!&&&&     ##!##!##      &&&!&&&&&&!&&
          130-42-1427      3-21-85       047-000065-24
          000-06-1217     12-24-86       050-000036-03
```

String Operator TO

The **TO** operator is evaluated Left-to-Right and is used to specify part of a string expression. The general form is:

string expression **TO** *string expression*

The *string expression* on the left is evaluated first and referred to as the *source*. Next the right *string expression* is evaluated and shall be referred to as the *pattern*. The resulting *string expression* is generated by copying all characters from the *source* up to and including the *pattern* string. If the *pattern* is not found within the *source*, then all characters of the *source* become the resulting *string expression*.

For example, if you have a large block of text and wish to find the first sentence, you might use this operator to find the result of:

str.var **TO** ". " (Locate first period followed by 2 spaces).

Numeric Expressions

Numeric expressions are performed in either integer or 6-word decimal floating point. Each argument is unpacked into the floating point register where all operations occur. The final result is maintained in the highest precision until the full expression is computed. The result is finally converted into the format requested by the operation. This may include truncation to an integer, or converted to the precision of a variable for storage of the result. An error can occur if the destination precision is not large enough to store the final result.

For example:

```
5 + 4

P * 10

VAL (A$)
```

String Expressions

A string expression *str.expr* is considered any group of string variables, literals, functions, CRT expressions and/or operators to be concatenated (linked together) returning a string result. For example:

```
T USING "#####"

A$ + B$
```

```
"Processing element: " + A$
```

All statements may incorporate the use of string or numeric expressions as long as the final result matches the format of the statement. What this means, is that any statement (not just **IF**, **PRINT** and **LET**) that previously required a *str.lit*, or *str.var*, may now contain any legal string expression. A variable is required only when a statement returns data into the variable such as **LET**, **INPUT**, **READ**, etc. Numeric and string conversion is performed across an equal sign of the **LET** statement, or through the **VAL**, **STR**, **ASC** and **CHR** functions.

The exact interpretation of the + operator is determined by the operand that precedes it, so that:

```
LET B$=A$+A
```

implies string concatenation, and the *num.var* A is converted to string automatically.

However, the reverse is not true:

```
LET B=A+A$
```

implies addition but is an invalid expression. In this case, the conversion of A$ must be done explicitly, e.g.:

```
LET B=A+VAL(A$)
```

String concatenation converts a numeric operand on the right from numeric to string. Numeric expressions do not perform automatic conversion of string elements.

Note: The IRIS string concatenator ',' may only be used in **IF** and **LET** statements. To utilize string expressions in all other statements, use the concatenator '+'.

Examples:

```
100 OPEN #0, P$+F$+"."+STR(SPC(5))

200 ON VAL(A$) GOTO 100,200,300

300 PRINT USING A$+".##"; D, E, F
```

Rules Governing String Processing

When using string items within a program, that is any *str.var*, *str.lit*, *crt.expr*, functions returning string values or *str.expr*, the following rules are applied to operations:

- A string may contain any of the ASCII codes listed in Appendix A.
- A zero ASCII byte is used to terminate any string segment.
- *str.lits* using the form \xxx\ to represent ASCII characters perform an automatic toggle of the high-bit to insure compatibility with IRIS and BITS applications externally, and Unix internally. When Binary Input or Output is enabled, this toggling is disabled for use in communications and raw binary processing.
- String variables must be **DIM**ensioned, **COM**mon, or **CHAIN READ** prior to use. They may not be re-dimensioned to other than the original declared size.
- String variables may be subscripted to select a starting and ending character position within

a string. A single *subscript* selects a starting point only. All strings terminate upon the occurrence of a zero-byte terminator, the second *subscript*, or the physical dimension of a string.

- A *full string* is defined to be any reference to a string variable in which a single or no subscripts are supplied.
- A *sub-string* is defined to be any reference to a string variable using 2 *subscripts*.

String Assignment

When assigning data to a string using **LET**, the following rules are applied when using *full strings*:

- The source is truncated to the size of the supplied destination.
- A zero-byte terminator is inserted in the destination if the source is shorter than the destination.
- A zero-byte terminator may be placed within a string by specifying a single *subscript* in the form: *str.var[x]* = "".

When assigning data to a string using **LET**, the following rules are applied when using *sub-strings* in IRIS applications:

- When the source is shorter than the destination, the remaining characters within the *subscripts* are deleted. Characters following the subscripted portion are shifted down to immediately follow the shorter source. (IRIS Mode).
- When a zero-byte is overlaid in the destination, it is pushed forward to the first character position following the length of the source copied. This may cause a zero-byte to be placed into the first character position beyond the second *subscript* if the source exactly fills or is larger than the destination.

When assigning data to a string using **LET**, the following rules are applied when using *sub-strings* in BITS applications:

- When the source is shorter than the destination, the second *subscript* is ignored. Only the number of characters supplied in the source are copied to the destination.
- When a zero-byte is overlaid in the destination, it is pushed forward to the first character position following the length of the source copied if and only if the source string does not completely fill the destination. No bytes outside the supplied *subscripts* are altered.

When assigning data to a string using **LET**, the following rules are applied when using *sub-strings* in IRIS applications running with the environment variable **STRING=HAGEN** set:

- When the source is shorter than the destination, the second *subscript* is ignored. Only the number of characters supplied in the source are copied to the destination. No shuffling down or overlaid zero-byte operations are performed.

Other special string functions are available to the application:

- Concatenated strings are evaluated and treated as a single source string for **LET**. IRIS programs concatenate strings in **LET** or **IF** statements by placing a comma between each *str.var*, *str.lit*, string function or *crt.expr*. For BITS applications, the concatenation operator + is used. The + operator may also be used for IRIS applications in statements other than

LET or **IF**.

- A string may be completely filled with a single character (or group of characters) except zero-byte terminators using the form:
- *str.var* = *str.expr* (+|,) *str.var*, i.e.: A$=" ",A$ to space fill.
- A zero-byte terminator is placed into a *str.var* by supplying a single *subscript* for the destination, and a null *str.lit* as the source, i.e. *str.var* = "". To fill a *str.var* with zero-byte terminators.

  See also: CALL 57 and CALL 60
- Characters beyond the zero-byte terminator may be operated upon by specifying a starting *subscript* beyond the zero-byte. Use the **LEN** function to determine the length of any *sub-string*.
- A number of special **CALL** Statements are available for string processing.
- Numeric data may be converted to string using the **LET** Statement, or in some cases the functions **STR** and **CHR**.

UniBasic Files

This section documents the types and usage of data files within UniBasic applications.

UniBasic differs from IRIS and BITS in its internal representation of numeric and string data within variables and files. These differences, once understood, provide the user a totally compatible platform for moving IRIS and BITS programs and data files without sacrificing the new features of Unix.

ASCII characters stored internally conform to 7-bit ASCII industry standard. 8-bit ASCII characters are reserved for graphics, and *crt mnemonics*.

IRIS and BITS store characters as 8-bit strings in exactly the reverse format. All printable characters have bit-8 set, and 7-bit codes (less than 200_8) are used for printer (or CRT) functions. A carriage return is represented as \215\ and code \015\ represents CRT function $\#15_8$.

Character processing is performed as follows:

Characters input from the terminal port are passed exactly as received. Most systems are configured to strip the parity bit which, in effect returns 7-bit characters to the system. Unless you are sending/receiving binary data, verify that the port is configured to strip this parity bit. The Unix command: **stty -a** command will display *istrip* if parity is being stripped, or *-istrip* if 8-bit data is allowed.

Program statements, commands and filename comparisons must be performed using 7-bit characters for consistent operation.

When a *str.lit* is entered, printable characters are stored as received. Characters entered using the \xxx\ octal representation form are high-bit toggled except \0\ and \200\; i.e. \201\ is stored internally as \001\, and \001\ as \201\. During display (such as **LIST**), the data is again toggled for display in the familiar form.

During output, printable characters less than \200\ are displayed directly. CRT translation is performed on all bytes greater than \200\ sent to the screen. When these characters are

transmitted to a *file* or *device*, no translation is performed. Later screen display of this data performs the CRT translation, or a supplied **lptfilter** is available to provide translation for device independence.

Since input characters are stripped and *str.lits* toggled internally, the application runs unmodified. Any \215\ in a program is stored and output as a [RETURN], and \015\ is stored as \215\ invoking CRT function #15.

When obtaining the decimal ASCII code of a character using **ASC** or **CALL $STRING**, the internal value is again toggled to match the IRIS/BITS format. A [RETURN] is the value 141_{10}. CRT codes are returned as codes less than 128. This facility permits most applications which check the ASCII range of a character to operate transparently.

When generating ASCII data using the **CHR** or **CALL $STRING** functions, your supplied code is toggled to the new internal format. In this way, the code 141 still generates a [RETURN] for your application.

This internal toggling is virtually transparent to all Business Application Programs. All normal comparisons of strings, input and records work as before. String comparison is always performed in 8-bit format to ensure compatibility when operating upon binary strings.

To facilitate operation with true binary data, the toggling feature for **ASC**, **CHR**, and **CALL $STRING** is automatically disabled when Binary Input or Binary Output modes are enabled. Binary Input and Output modes are available using **SYSTEM** and the **IO mnemonics**.

Note: System or special applications that manipulate binary data using **CHR**, **ASC** or **$STRING** may yield unpredictable results when Binary Input/Output is not enabled since the resulting top bits will be toggled.

To pack or unpack binary data when not operating in Binary mode, use the **CONV** statement. If this statement is not acceptable, a **CALL** is provided to toggle data within a string according to the same rules described above to minimize changes to these special system programs. For example:

A$ contains a binary string:

```
CALL 60,3,A$ !Toggle the top bits
```

...proceed as normal, processing the data with ASC/$STRING

A$ contains binary data built from $STRING or CHR function

```
CALL 60,3,A$ !Toggle data into actual binary
```

Introduction to Files

A *file* is a pre-selected area of the disk to be considered a single data storage entity. Files allow data to be stored and retrieved by programs, and retain their data indefinitely. A *device* is an external storage medium such as a hardcopy printer, magnetic tape, or terminal screen.

Maximum file size is limited by the host operating system. Usually, a file may contain a maximum of 2^{31} bytes. On many operating systems, files can be created as "huge" files to exceed this limit. Some systems may have a limit set upon the number of blocks a file can contain. This value is available using the command **ulimit**. Following installation, verify that this value is not restrictive for your applications.

All files are logically divided into equal sections called *records*. Record division allows data to be accessed via its *record number*. Each record is made up of a selected number of bytes (characters), and all records in a given file are of equal length. When a file is created, the creator specifies this record length in bytes or words (byte-pairs). Data records may be any even length from 2 to 65534 bytes. All files have a *record length*, whether accessed by record or not. Saved BASIC programs, for example, are given an arbitrary record length of 65534 bytes.

Record numbers usually start at zero, meaning a file with five records has record numbers 0, 1, 2, 3, and 4. Individual bytes within each record are also numbered from zero. BASIC statements allow access to specific bytes within any record by giving a *byte displacement*.

To access a *file*, a link is made to the file using a *channel number* in the range 0 to 99. All communication is via the *channel number* linked to the *file* or *device*. The link is made using one of the statements **BUILD**, **CREATE**, **EOPEN**, **OPEN**, or **ROPEN**.

Several types of data file structures are supported, each with its own rules governing access and modification. The types of files available to UniBasic are:

Universal Data Files

Contiguous Data Files

Tree-Structured Data Files

Formatted Item Files

Indexed Keyed Files

Saved BASIC Program Files

UniBasic can also read and write dL4 Portable Indexed Contiguous, Contiguous, or Formatted files. All other files are assumed to be Text Files and are accessed according to the rules contained herein.

Filenames and Pathnames

A *filename* is the name given to a *file*, and is made up of lower-case letters, digits, dash (-) or periods (.). Upper-case characters are converted to lower-case automatically. Other characters, although allowed by Unix are not permitted in standard UniBasic *filenames*.

A *pathname* is a series of Unix *directory* names separated by /, terminated with a *filename*, such as: /usr/ub/23/payroll.

Standard *filenames* are converted to a series of *pathnames*, appended one at a time to the entries of the **LUST** (Logical Unit Search Table) until a match is found.

Filenames beginning with / are assumed to be full *pathnames* and are passed directly to Unix. **LUST** is not used, and no conversion is performed.

The form *pack:file* is converted into *pack/file*. Account branch characters (%&#, etc) and account

[grp-usr] suffixes are discarded.

Filenames in the form *0/filename* are converted into *sys/filename*; files in the form *lu/filename* remain unchanged excepting the omission of leading zeros in the lu number, i.e. *023/filename* becomes *23/filename*.

To replace an existing *filename*, append an **!** character to the *filename*.

File Attributes, Protection and Permissions

Access to files on the system is controlled by the file *attributes* or *permissions* given by the creator for access to a file by other users on a system. The default *attributes* under Unix are made up of 3 octal digits. The first digit affects the owner/creator of the file. The second digit controls other users in the same group, and the third digit controls access to all other users. The digits are as follows:

- 4 Allows reading of a file
- 2 Allows writing to a file
- 1 Allows execution of a file (for shell scripts and C programs)

The digits are combined to select the desired protection. A 6, for example permits reading and writing to a file; 666 allows reading and writing by all levels. The default *permission* (when none are specified) is 666 permitting reading and writing by all users. To facilitate a different default protection (such as IRIS <77> protection against all but owner), change the **umask** setting in */usr/ub/.profile*. This mask is a 3-digit mask that removes *permission* digits passed on **CREATE** and **BUILD**. The first digit should be 0 to allow the owner unlimited access to the file. The second and third digits control masking for other users in the same group and other users in different groups as follows:

- 4 Remove read permission.
- 2 Remove write permission

To simulate IRIS default <77> protection, set **umask** to 066.

Using IRIS Protections

IRIS protections <pp> are processed as follows. A 6 is selected for the owner/creator, the first digit is applied to users in other groups, and the second digit is applied to users in the same group. Note that privilege levels are not supported in Unix; the same group equates to the same privilege, and other users in other groups applies to users at lower privilege levels.

The IRIS digits are mapped as follows:

- 4 Remove Read permission
- 2 Remove Write permission
- 1 Ignored

A <77> protection results in the Unix protections <600>, <70> maps to <660> and <33> to

<644>.

Using Unix Permissions Directly

A 3-digit *permission* value may be passed directly to Unix. The **BUILD** and **CREATE** statements as well as the **CHANGE** utility provide for specifying a full 3-digit protection value. The *permissions* are supplied using the format: <ppp> as defined above.

See also: File Attributes, Protection and Permissions.

BITS Attributes

BITS attributes <PRWEO> may be specified and are converted into the appropriate Unix permission.

- P Set default 666 protection code
- R Remove Read permission at all levels except owner
- W Remove Write permission at all levels except owner

Other BITS attribute letters, such as: D, S, G, A, and B are accepted and ignored.

Supplemental Protection Attributes

Additional letter attributes are supported and must be placed before any numeric selections within the <> brackets.

- U Build a Universal data file which contains IRIS style BCD data. Unlike other IRIS BCD files, these data files are the only ones that are platform independent.

  See also: Universal Data Files, **PREALLOCATE** environment variable and IRIS BCD Data Files.

- H Build a "huge" Universal data file. A "huge" file is a Universal data file that supports data or index parts larger than 2 gigabytes in size. Huge files are not supported on some older operating systems.

- Q Build the file to contain IRIS style BCD data. Valid for data files only. Forces numeric data to be stored in IRIS Binary-Coded Decimal form. Q is used for files transferred from IRIS without record conversion.

  See also: **PREALLOCATE** environment variable and IRIS BCD Data Files.

- K Build the file to contain 8-Bit IRIS/BITS style binary keys. Data is toggled to 7-Bit format whenever a key is retrieved into a string variable, and into 8-Bit format when new keys are inserted. This attribute is required when a file has mixed key values both above and below \200\. Normal ASCII keys do not require this special attribute. When converting files from IRIS, options are available to force this condition.

  See also: **PREALLOCATE** environment variable and IRIS BCD Data Files.

- F The program is an *IRIS BASIC* program. This attribute causes the program to obey

IRIS rules for encoding syntax of BASIC Statements and Runtime considerations. This attribute is set automatically during SAVE commands, and has no effect if set on data files. IRIS rules are applied for all runtime and file-access statements.

E The program is *execute-only* and cannot be listed. Valid for saved BASIC programs only. The program may be executed, but all channels are closed and the program is erased from the user's partition when aborted or completed. This attribute is used for system command programs written in BASIC, such as **LIBR**.

O The program is an *overlay*. When an *overlay* program is executed from *command mode*, UniBasic is forked creating a *child process* to run this command. Upon termination for any reason, the *child process* dies, and remaining type-ahead is returned to the original program. The original program is restored as if the Overlay program was never called. Specifically, *overlay* protection is used for BASIC program processors such as **LIBR**, **QUERY**, **SCAN**, etc.

J Build the file to contain IMS style BCD data. Valid for data files only. Forces numeric data to be stored in IMS Binary-Coded Decimal form. J is used for files transferred from IMS without record conversion.

Y Flag the file as an IRIS polyfile. Perform functions in bytes instead of words, and set a first real data record of zero.

Z Force usage of BITS numeric and string data in a "Huge" file. Warning: files using BITS data are not portable between platforms.

See also: **PREALLOCATE** environment variable and IMS BCD Data Files.

Accessing Data Files Through a Channel

Once a *channel* link is established, file access may be performed. The following statements are used to control *channel* links, and transfer data to and from *files*.

BUILD #	Build a new data or Text File.
CLEAR #	Clear an open channel (same as CLOSE).
CLOSE #	Close an open channel.
CREATE #	Create a new data file.
EOPEN #	Exclusively open a file for single access.
INDEX #	Maintain the index portion of a file.
INPUT #	Input ASCII input from a channel; BITS only.
MAT READ #	Read {lock} a matrix / binary string.
MAT WRITE #	Write {lock} a matrix/binary string.
OPEN #	Open an existing file for reading and writing.
PRINT#	Redirect normal PRINT format to a channel.
RDLOCK #	Read and lock a record.
RDREL #	Read a relative 512-byte block from a channel.
READ #	Read {lock} data from a channel.

REWIND #	Reset the channel to the first record and byte.
ROPEN #	Open a file for Read-only, ignore locks.
SEARCH #	Maintain the index portion of a file.
SETFP #	Set the file position for sequential transfers.
UNLOCK #	Unlock any locked record on a channel.
WRITE #	Write {lock} data to a channel.
WRITE #x;;	Unlock any locked record on a channel.
WRLOCK #	Write and lock a record.
WRREL #	Write a relative 512-byte block to a channel.

Note: Data transfer is governed by the file type for IRIS applications, and by the statement used for BITS applications. Mixing statement types can have adverse effects on an application. Before using any class of statement, refer to BASIC Statements and Appendix D CALLS in this guide for additional information.

Channel Expression - chn.expr

SYNOPSIS:

STATEMENT *#channel* {,*record* {,*byte displacement* {,*time-out*}}};*expr.list* {;}

DESCRIPTION:

STATEMENT specifies any BASIC statement that performs an operation to a *file* or *device*, as described previously.

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select one of 100 possible open files. The *channel* must be in the range 0 to 99. Special *channels* are reserved for system use. Channel (-1) contains the open BASIC program currently loaded. There is no open *channel* if this is an unsaved program. Channel (-2) is used for special operations such as **DUMP**, **LOAD** and **MERGE**.

The *channel* may be the only parameter if it is followed by a semi-colon, i.e. **#3;**. Additional parameters are parsed until the first semi-colon is seen. An error occurs if more than (4) parameters are supplied and a semi-colon terminator for the *channel expression* is not specified.

The optional *record* is any *num.expr* which, after evaluation is truncated to an integer and used to select a starting record number for the transfer. If the *record* expression is omitted, transfer will be sequential based upon the file type, statement and emulation (IRIS/BITS) in force. Sequential access is always from the last byte transferred for BITS applications.

When sequentially accessing records in IRIS applications, the following rules apply:

RECORD	ACTION PERFORMED
omitted	The *record* number used for the last access to this channel is incremented and used to select the record. This mode reads sequential records of a file.
-1	Performs identically to 'omitted' except that it serves as a place holder so that a *byte displacement* may be specified.
-2	The *record* is reset to the same record number used during the last access to this channel. This, in effect re-transfers the same record.

The optional *byte displacement* is any *num.expr* which, after evaluation is truncated to an integer and used to specify the starting point in the *record* for the transfer. If the *byte displacement* is omitted, transfer begins with byte 0 of the selected *record*.

The optional *time-out* expression is any *num.var* which, after evaluation is truncated to an integer and used as the maximum time (in tenth-seconds) to wait for the selected *record* to become unlocked. If, after the specified *time-out* the *record* is still locked, the error Selected Record is Locked is returned to the program. If the *time-out* is (-1) or omitted, default record lock retry is governed by the environment variable **LOCKRETRY**. If this value is zero, retry continues indefinitely. A non-zero value specifies the number of five-second periods to wait prior to issuing the Selected Record is Locked error. Any *time-out* is terminated immediately upon the *record* becoming available.

The *expr.list* may contain a list of variables or expressions for the operation.

If the *statement* is terminated with a semi-colon, and the running program is an IRIS program, the selected *record* is unlocked at the termination of the statement. Otherwise the *record* remains locked until another operation is performed unlocking the *record*.

ERRORS:

Channel is not opened

Channel is already opened

Illegal Channel Number

Selected Record is Locked

See Also:

CHF function, CHN function, Accessing Data Files Through a Channel, Introduction to Files

Record Locking

Record locking is a feature of the file structure to restrict access of a given *record* to a single user. Under Unix, this is accomplished by first checking whether any other user has a lock on

the same *record* on the same file. If not, the *record* is locked while the statement performs its transfers. Upon completion of the transfer, the *record* is unlocked unless the statement requested a continuing lock.

Record Locking is essential in applications where two or more users are trying to update the same information simultaneously. The first user might be performing an inventory receipt, while the other is taking stock to fill an order. Applications must be written to ensure that all updating operations are performed using Record Locking. When two or more users attempt access, the first is given access, and additional users are suspended (or an error is given) until the *record* is available.

For example, the first user is updating stock received into inventory. The part number is entered and its *record* is locked. The second user entering that part number for an order is suspended. The first user enters the amount received and the *record* is updated and unlocked. The second user continues unaware of the dual access. This assumes of course that the first user didn't leave the *record* locked indefinitely.

A *deadly embrace* may occur when two or more users are attempting to access a *record* which is locked by the other. Both users wait indefinitely for the other to unlock the *record*. For example, user 1 has locked the ABC Company customer record and is attempting to read the parts file record for wool carpet. Meanwhile, user 2 has already locked wool carpet and tries to read ABC Company. Each waits indefinitely for the other. Some Unix systems return a system error (negative BASIC error) when a *deadly embrace* is detected.

You can avoid infinite suspension of a program by specifying a *time-out* or period of time (in tenth-seconds) to wait for a locked *record*. If, after that amount of time the *record* is still locked, an error is generated to the program. For older applications, set a system-wide *time-out* default selected when no individual *time-out* is specified in the statement.

The Environment Variable **LOCKRETRY** specifies this delay. If the value is undefined (or zero), programs wait indefinitely for locked records (IRIS 7 style). A non-zero value indicates the number of five second intervals to wait before generating an error to the application.

To perform an operation and lock a *record* in IRIS mode, simply omit the optional ';' at the end of the statement. To perform the operation and unlock the *record*, include the trailing ';'. To unlock any previously locked *record* on a channel without performing a transfer, issue the statement: **WRITE #channel;;**

In BITS mode, the statement controls Record Locking (**READ**, **WRITE**, **PRINT**, **INPUT**) for operations without locking, and **RDLOCK/WRLOCK** for operations requiring locking. To remove any outstanding locks on a channel, the **UNLOCK #** statement is used.

Note: Any locked *record* on a channel is automatically removed on any of the following:

Closing the channel.

Trailing semi-colon on the last operation (IRIS).

Access to the same record without again locking.

Attempted access to any other record.

Only a single *record* may be locked on any given channel. If you need to lock several at once, you must open the file on separate channels.

Text Files

A Text file is a file comprised of ASCII characters terminated by a zero-byte. For purposes of random access, Text Files are assumed to have a record length of 512 bytes. Data begins in the first byte of the file and there is no special UniBasic header. Lines of text are separated by the Unix new-line (\12\) character. When Text Files are created, the data is stored in Unix 7-bit ASCII format to ensure compatibility with all other Unix text editors, word processors or other programs.

Creating Text Files

Text files are created using the **BUILD** statement. Standard Unix files are built using 7-bit data without any special UniBasic header information. All Text files are accessible to any Unix text processor or command.

Accessing Text Files

Text files are typically accessed sequentially. When data is written to a Text File, carriage returns are converted into new-line characters. A column count is maintained for the channel. Printable characters increment the column; return, new-line or form-feed resets the counter to zero.

When **TAB** functions are used to the open channel (i.e. writing to a *device* such as a printer), the column is kept separately from the column count of the screen. If writing to a file, a zero byte terminator is always maintained at the end of the file. A zero byte is written and the file pointer is decremented such that each subsequent write operation overwrites the trailing zero byte, and appends a new zero-byte at the end-of-file.

When reading data from a Text File, End-of-File is signified by the occurrence of a zero byte, regardless of whether data exists beyond the zero-byte. BITS programs generate an End of File error (88), and IRIS applications simply receive a null (empty) string.

When BITS applications read from Text Files, the normal statement used for sequential access is **INPUT**. Input terminates on new-line or form-feed. No terminator is placed into the string. An empty string is simply a blank line in the Text File. Carriage returns are stripped from the file and ignored.

When IRIS applications **READ** from Text Files, a null string indicates the end-of-file. Otherwise, carriage returns are stripped from the file, and new-lines terminate the **READ**. All new-lines are converted into the string as \215\ carriage returns. Additional special modes are available for IRIS applications reading Text Files. The optional *record* controls the type of operation to perform:

Record	Action Performed
omitted	Access the next sequential byte of the file up to the first new-line character or size of the string (whichever is smaller). Replace the new-line with \215\.
-1	Same as 'omitted'.
-2	Transfer characters up to the **DIM**ensioned size of the string variable. Convert new-lines to \215\ but do not terminate the transfer until end-of-file or filling the string.

Text files may also be accessed using **MAT READ** and **MAT WRITE** statements.

Saved BASIC Program Files

A SAVED BASIC program file contains p-code compiled BASIC programs stored by the **SAVE** or **PSAVE** commands. Each program is stored with several flags indicating the type of program (IRIS or BITS), and encryption status. For further information on application program protection and encryption, see the **PSAVE** command.

Newly created programs are of the type IRIS or BITS based upon the default **BASICMODE** environment variable or command (**NEW**, **NEWB**, or **NEWI**) issued. This option controls statement syntax and run-time operation and cannot be changed for the life of the program file.

A program file is converted to a Text File using the **DUMP** command.

When converting a Text File into a program file, verify that your default program mode (IRIS or BITS) is set via the **BASICMODE** environment variable, the proper **NEW** command, or by issuing the proper **GET** command for BITS mode.

Contiguous Data Files

Even though the Unix systems do not support Contiguous files in the traditional internal sense, compatibility is provided for applications designed to use these files.

Contiguous files utilize a fixed-length *record*, specified during creation. Each *record* contains the identical number of bytes. The total number of records to be within the file is stored within the file's header during creation.

The value of the **PREALLOCATE** environment variable is used during file creation and globally during execution of programs performing Contiguous file access. Refer to this documentation in order to define the options properly for your applications. **PREALLOCATE** provides features including pre-writing all records to null, limiting expansion, and eliminating system file structure gaps in the file.

A Contiguous file will return as its number of records (**CHF/CHN** functions), the greater value of its current physical size, or the size in records specified during creation.

Access to any *record* within the valid **CHF/CHN** range with either **READ** or WRITE statements is permitted. If the record is beyond the current physical size, the file is extended unless this feature is restricted using **PREALLOCATE**. To expand a Contiguous file, simply write to any record higher than the current size.

During expansion of the file, all intervening records are written (with zero bytes) from the current physical size up to and including the new record. To prevent the writing of all intervening records, set **PREALLOCATE** accordingly. This automatic filling in of records is to prevent Unix from reporting the file as sparse (gaps). Sparse files are usually considered corrupted when the file system is checked, although they are perfectly valid.

Creating Contiguous Files

Contiguous files are created using the **BUILD** or **CREATE** statements. In addition, the **FORMAT** command may be used from command mode to create the file. A Contiguous file may have any number of records with a maximum record length of 65534 bytes (32767 words). Contiguous files may be built as Universal files if the **PREALLOCATE** 8192 (16384 for "huge") flag is set or if the <U> or <H> attributes are specified.

Accessing Contiguous Files

Contiguous files are accessed by supplying the **record** and *byte displacement*. Access may cross a logical record boundary. Care must be taken to ensure that your transfers are within the specified *record* or data in subsequent records may be damaged.

When transferring data to a Contiguous file, the *record*, and *byte displacement* are used to specify the starting point for the transfer. All items in the *var list* are transferred sequentially. The following table illustrates the optional use of the supplied *record*.

Record	Action Performed
omitted	The *record* number used for the last access to this channel is incremented and used to select the *record* if the program is an IRIS program. BITS programs resume transfer at the first byte not transferred by a previous operation. This mode reads sequential records of a file.
-1	The *record* number used for the last access to this channel is incremented and used to select the *record*. This mode permits the selection of a new *byte displacement* within the incremented *record*.
-2	The *record* is reset to the same *record* number used during the last access to this channel. This, in effect re-transfers the same record.

Tree-Structured Data Files

Tree-structured files utilize a fixed-length **record**, specified during creation. Each *record* contains the identical number of bytes. These type of files are preserved for compatibility with BITS applications. They provide a free-format record in a dynamically expandable structure.

A Tree-structured file will return as its number of records (**CHF/CHN** functions), its current physical size in records.

Access to any *record* within the valid **CHF/CHN** range with either **READ** or **WRITE** statements is permitted. If the record is beyond the current physical size, the file is extended.

During expansion of the file, all intervening records are written (with zero bytes) from the files current physical size up to and including the record being accessed. This automatic filling in of records is to prevent Unix from reporting the file as sparse (gaps). Sparse files are usually considered corrupted when the file system is checked, although they are perfectly valid.

Creating Tree-Structured Files

Tree-structured files are created using the **CREATE** statement. A Tree-structured file may have any number of records with a maximum record length of 65534 bytes (32767 words).

Accessing Tree-Structured Files

Tree-Structured files are accessed by supplying the *record* and *byte displacement*. Access may cross a logical record boundary. Care must be taken to ensure that your transfers are within the specified *record* or data in subsequent records may be damaged.

When transferring data to a Tree-structured file, the *record*, and *byte displacement* are used to specify the starting point for the transfer. All items in the *var list* are transferred. The following table illustrates the optional use of the supplied *record*.

Record	Action Performed
omitted	The *record* number used for the last access to this channel is incremented and used to select the *record* if the program is an IRIS program. BITS programs resume transfer at the first byte not transferred by a previous operation. This mode reads sequential records of a file.
-1	The *record* number used for the last access to this channel is incremented and used to select the *record*. This mode permits the selection of a new *byte displacement* within the incremented *record*.
-2	The *record* is reset to the same *record* number used during the last access to this channel. This, in effect re-transfers the same record.

Formatted (Item) Data Files

Formatted ITEM files are sequential data files utilizing a fixed-length record and fixed record format. Each record is pre-defined with respect to the data record definition. The format is initialized through creation and is maintained for the duration of the file's existence. When initially created, only a single record (Record 0) is within the file.

The record length can be up to 65534 bytes in length. A null record is returned when access is made to a record below the maximum record number, but not physically in the file.

The value of the **PREALLOCATE** environment variable is used during file creation and globally during execution of programs performing Formatted Item file access. Refer to this documentation in order to define the options properly for your applications.

A Formatted file will return as its number of records (**CHF/CHN** functions), the first record not contained within the file. If your files grow dynamically using this function, no empty records exist in the file. If you **READ** a record beyond the current number of records in the file, an error is generated (Illegal Record or End-of-file). When you **WRITE** a record beyond the current number of records, the file is expanded automatically.

During expansion of the file, all intervening records are written (with zero bytes) from the file's current physical size up to and including the record being accessed. To prevent the writing of all intervening records, set **PREALLOCATE** accordingly. This automatic filling in of records is to prevent Unix from reporting the file as sparse (gaps). Sparse files are usually considered corrupted when the file system is checked, although they are perfectly valid.

PREALLOCATE may be set to return a Record-Not-Written error if required by the application. When defined, each read operation is checked for a null record. If the record contains all zero-bytes, the Record Not Written error is returned. When not defined, null records are returned. This function slightly degrades read access to Formatted files. Set this option only when your application expects the Record Not Written error in the middle of the file.

Creating Formatted ITEM Files

Formatted ITEM files are created using the **BUILD** or **CREATE** statements. In addition, the **FORMAT** command may be used from command mode to create the file. A Formatted Item file may have any number of records with a maximum record length of 65534 bytes (32767 words). Formatted ITEM files may be built as Universal files if the **PREALLOCATE** 8192 (16384 for "huge") flag is set or if the <U> or <H> attributes are specified.

To create a Formatted Item file within an application, write to record zero a list of variables to sequential item numbers. The type and **DIM** of each variable is recorded in the format map. When a numeric variable is written, its precision is also stored in the format map. When a string variable is written, its **DIM**ensioned size is incremented and then rounded up to an even number of bytes. If a **MAT** operation is performed, a Binary Item is created using the actual **DIM**ensioned size. Strings are rounded up (not incremented first), and numerics occupy the entire size of the specified variable, array or matrix. The actual data within the variables is also written to the record after the item is defined in the format map.

An error is generated if items are written in other than sequential item number order starting at 0, or when you exceed 128 items. Once an item is defined, its type, precision or length may not be changed.

Accessing Formatted ITEM Files

Formatted files are accessed by supplying the *record* and *item number* (*byte displacement*). Access cannot cross a logical record boundary.

When transferring data to a Formatted Item file, the record and *item number* are used to specify the starting point for the transfer. All items in the *var list* are transferred, and each must match the pre-defined record layout in the format map.

If an Item is defined as string, only a *str.var* may be transferred. If the Item is numeric, a conversion is performed when the variable precision does not match the item's definition. Data is converted to the precision of the destination; *var* when reading, *item* when writing. An error occurs if the destination precision is too small to hold the numeric value.

Binary items are accessible using **MAT** statements. You can, however transfer any *str.var*, *num.var*, *mat.var* or *array.var* into a binary field. No conversion is performed. Care must be exercised to ensure that numeric data is transferred into variables of the same precision used when written or the resulting data will be indistinguishable to the application.

The following table illustrates the optional use of the supplied *record*.

Record	Action Performed
	The *record* number used for the last access to this channel is incremented and

omitted	used to select the *record*. This mode reads sequential records of a file.
-1	Performs identically to 'omitted' except that it serves as a place holder so that a *byte displacement* may be specified.
-2	The *record* is reset to the same *record* number used during the last access to this channel. This, in effect re-transfers the same record.

Indexed Data Files

An Indexed Data File is any Contiguous Data File which is defined to contain a companion ISAM Key file. Access to data records is identical to a standard Contiguous Data File, except write operations may not cross a *record* boundary unless enabled by **PREALLOCATE**. The companion ISAM (Indexed Sequential Access Method) file holds keys and pointers to data within the Contiguous Data File. The use of an Indexed file allows an application to rapidly locate data in a large database. Even when a file contains several hundred thousand data records, a specific record can be located instantly.

The environment variable **PREALLOCATE** options affecting Contiguous Data Files also apply to Indexed Data Files. In addition, four options are provided specifically controlling Indexed Files:

PREALLOCATE option 128 prevents dynamic expansion beyond the number of records specified at creation. This limits the number of active records a user may insert using the **SEARCH** or **INDEX** statements. When enabled, a new record is not allocated when the first available record is greater or equal to the number of records specified during creation. To expand the file, **WRITE** to a higher limiting record number. The record number must be greater or equal to the value returned by **CHF** or corruption of the delete list may occur! Note, this operation is prevented if **PREALLOCATE** option 2 is enabled for this session.

PREALLOCATE option 256 forces a check prior to deleting a record using **SEARCH** or **INDEX**. If the record is already deleted (not in use), an exception status is returned. This option is required to simulate BITS and Polyfiles bit-map ability to delete records whether they are in use or not.

PREALLOCATE option 512 allows a **WRITE** to cross an ISAM record boundary. Normally, an error is generated when a **WRITE** to an indexed file crosses a record boundary. Setting this option should only be done when the application can be certain that all records to be written are already allocated, otherwise the file's deleted record list will be corrupted.

PREALLOCATE option 4096 prohibits writing to a deleted record. An examination of a record's status (deleted or in-use) is made prior to performing a **WRITE**. An error is generated if the record is already deleted, preserving a file's free record chain.

Indexed files, consisting of optional data records and keys, are maintained by the application program. When new data is to be added to the file, you request a new record. Automatically, the system expands the file if there are no unused records. After writing your new data to the supplied record of the file, you insert a *key*, that is a unique piece of information tagged to the new record. The *key* could be a customer name, number; any unique information about the record. Later, you retrieve the record by simply asking for the record that contains the key.

Each file can have from 1 to 62 separate indices, and each index may have a different sized *key* (up to 122-bytes). This allows multiple *keys* (e.g. name, account number, etc.) to access the same

data. Each different index provides a different way to locate a *record*.

Any given *record* may be located by its specific *key*. When the entire *key* is not available, a group of *records* matching a partial *key* may be displayed for final selection under program control.

Data records may be read from the file sequentially (in key order), forward or backward for as many different indices as are in the file. For example, a file keyed by customer name and number could produce a sorted (ascending or descending) report by those fields without any resorting.

When information is no longer needed in a file, the keys are deleted, and the record is returned to the system for later reuse before extending the file.

Indexed Files are not required to contain data records. A Contiguous Data File is always present with a single data record, but may be unused. This allows indices to exist separately from the data referenced, or to build key-only files into existing data bases.

Indexed files utilize the FairCom, release 4.3C c-tree™ file structure, widely accepted in the Unix community for its reliability, industry standard approach, and extended features.

c-tree 4.3C provides for index node deallocation and b-tree compression when keys are deleted; but only in the single-user or server environment. Using the full locking capabilities of Unix, compression is allowed in the multi-user environment.

This is accomplished by granting a user who is deleting a key exclusive index access for the duration of the delete. This is a requirement of the b-tree compression algorithm. Users performing searches and/or insertions can still access an index concurrently.

This type of compression has the following benefits:

- Unused space in an index is kept to a minimum. When an index block becomes empty, it is placed on the delete list. It therefore be can be reused elsewhere in the index when required.
- An index that has keys systematically added to the end and deleted from the beginning does not require the file to grow continuously.
- Since overall index size is reduced, overall access performance to the index is proportionally increased, with very large indices benefiting the most.

The added locking permits implementation of the fast search-next and search-previous function in c-tree. Sequential mode 3 or mode 6 searches through an index now do not require a complete b-tree search. The current key position is always saved and the next key in sequence is returned, if possible. Concurrent changes to the index are detected, and a full search is only performed if necessary.

Indexed Data Files are maintained within 2 separate Unix files. These are a standard Contiguous Data File utilizing a lower-case name (as built), and the ISAM (key) portion in a companion file with the same name using upper-case characters (i.e. payroll and PAYROLL). In the case of Universal Data files, the ISAM portion companion file has a .idx extension (i.e. payroll and payroll.idx).

Indexed File Creation

Indexed files are created using the **BUILD**, **CREATE**, **INDEX** and **SEARCH** statements or

using one of the supplied utilities **BUILDXF** or **MAKEIN**. They are initially created with a single data record. The actual number of records supplied to the statements or utilities is stored in the file header. Indexed files may be built as Universal files if the **PREALLOCATE** 8192 (16384 for "huge") flag is set or if the <U> or <H> attributes are specified.

Note: An ISAM file is made up of (2) separate files; the lower-case filename holds the data portion and an uppercase filename is created to hold the ISAM portion. If the file is Universal, the ISAM portion will have the data file name with a .idx extension rather than an uppercase filename. Filenames that do not contain at least one letter cannot be used for ISAM data files.

During initial creation, you may specify the type of B-Tree balancing to apply to each index. Proper selection increases performance and minimizes the disk space required to hold keys. The default is to assume random key insertions into each index. This results in a well balanced tree-structure with nodes split when half full. If your insertions into a specific directory are sequential (ascending or descending), you may change this parameter to suit your application. An example of a sequential index is an order/invoice number file keyed by an increasing (decreasing) number or date. By setting the proper parameter, as much as 25% performance and a 50% reduction in disk space may be realized; See Summary of SEARCH/INDEX Modes.

When allocating new records, the system first checks for any deleted records that can be reused. If found, they are used first. When no deleted records exist in the file, the file is expanded using the number of records specified by the environment variable **DXTDSIZ**. This value is set to a default of one for the best overall performance. Setting this to a higher value may increase performance of certain applications.

Similarly, when the ISAM portion of the file is full, it is expanded by the value specified in bytes of the environment variable IXTDSIZ. This value must be a multiple of 512 or the file may grow erratically. The default value should never have to be changed.

To maintain a dynamically expandable file structure, c-tree maintains a linked list of deleted records in the data portion of the file. When records are returned to the system, c-tree checks that you have not returned the same record twice in a row. It does not normally check to see if you have returned the record in a previous operation. It is therefore possible to corrupt the Deleted Record Chain if you arbitrarily return records not actually allocated. To prevent this, you can set the environment variable **PREALLOCATE** option 256 to force c-tree to check for a record already deleted.

Deleted records are flagged with a single-byte *delete-character* (ff hex, 3778). Next, a 4-byte pointer is written linking deleted records together into a *delete-list*. The top of the *delete-list* is maintained in the header. It is possible to corrupt this pointer system if you perform a **WRITE #** operation to a *record* following its release as a free record. Many applications write their own delete-flag into unused records. If your applications require this capability, set the environment variable **ISAMOFFSET** to a byte location other than zero (default) such that c-tree has 5 contiguous bytes available for *delete-list* maintenance.

C-tree requires internal arrays of data to maintain fast key operations such as search next. For each Indexed file your application opens, one array element is required for the data portion of the file, and one element for each Index in the file. A typical application opening 10 files with an average of 3 indices requires (3 + 1) * 10 or 40 positions. If your application errors trying to OPEN too many ISAM files, change the default value of the environment variable

ISAMFILES.

Indexed files dynamically expand to meet the requirements of your application. Over a period of time, continuous expansion and contraction of data occurs in your files. For example, at month or year-end, applications typically delete a large number of keys and records. The Unix system does not provide for a reduction in a file's size. The **ubcompress** utility is provided to rebuild the ISAM portion of the file and release unneeded space back to the system. The data portion is not compressed to insure that all records maintain their positions in the file. Additionally, since not all applications have the keys within the data records, the process of sorting and rebuilding all indices to point to the compressed file would be very time consuming.

Accessing an Indexed Data File

An Indexed File is accessed using the **SEARCH #** and/or **INDEX#** statements. The parameters are identical and select operation mode, index to operate upon, and data values passed both ways.

SYNOPSIS

SEARCH #*channel , mode , index ; key var, record var, status var*

INDEX #*channel ; mode , index, key var, record var, status var*

channel is any *num.expr* which, after evaluation is truncated to an integer and used to specify an opened channel currently linked to an Indexed Data file. A semi-colon may follow the *channel* or *index*.

mode is any *num.expr* which, after evaluation is truncated to an integer and used to specify a mode of operation for the statement. The following pages provide a detailed list of *mode* operations.

index is any *num.expr* which, after evaluation is truncated to an integer and used to specify an Index or Directory (list of keys) for the operation.

key.var is any **DIM**ensioned *str.var* which must be **DIM**ensioned large enough to hold the key being operated upon. An error is generated on search type operations if a *key* from the file cannot fit into the supplied *str.var*.

record.var is any *num.var* and contains (or returns) a value for the statement *mode*.

status.var is any *num.var* used to return a status (exception) value to the program. Generally, a zero indicates a successful operation; non-zero for an exception error. When issuing *mode* 1 functions, the *status.var* is set before the statement to select the miscellaneous information to be returned.

Mode 0 - Index Definition

Generally, Indexed Files are created and structured using the **MAKEIN** or **BUILDXF** utility .

SEARCH/INDEX *mode* 0 is used to create an Indexed File during program execution.

Each index in the file is defined using a *mode* 0 statement specifying the *key* length. Indices must be defined in sequential order, beginning with 1, up to a maximum of 62. The *index* is selected with the index expression.

The *record.var* defines the key length (2-122 bytes) of the selected *index*. Key length is expressed in bytes for BITS Applications and IRIS Polyfiles where a **CALL $VOLLINK** is issued, or in words (byte pairs) for standard IRIS Indexed files.

status.var is set upon completion as follows:

0	Operation successful.
4	File is not a data file (type Data or Contiguous).
6	Selected index number is out of sequence.
8	File already indexed (May not be changed once defined).
9	Illegal parameter specified. Key length can be 2-122 bytes.
10	Too many indices specified. Maximum is 62.

To create an Indexed File with two indices of key lengths (bytes) of 6 and 24 requires two *mode* 0 statements. The first to *index* 1 with *record.var* containing 6; the second to *index* 2 with *record.var* equal to 24.

As each *index* is defined, a *mode* 8 may be issued to the same index with *record.var* set to 0 for random insertions, 1 for increasing keys, and 2 for decreasing keys. If this step is omitted, random insertions are performed.

The data portion of an Indexed File begins with data *record* zero. If the creation program is an IRIS program, or the **BUILDXF** utility is used, the file begins with record one; that is no record zero is logically within the file. To force the first data record to be other than zero, issue a *mode* 1, with *record.var* set to the desired first record number and *status.var* set to 6. Setting a First Real Data Record other than zero does not occupy space within a file. The system simply stores a starting record constant which is added or subtracted from all file operations. If the First Real Record is set to 200, then logical record 200 equals physical record 0; 210 record 10, and so on. This feature is included for compatibility when moving existing data files from a live IRIS system in order to keep the record numbers and key pointers consistent.

Once all indices have been defined, the file structure must be locked. This is accomplished by issuing a *mode* 0 statement with *index* equal to 0 and *record.var* set to the desired number of data records. This number of records is placed into the file header and used by **CHF/CHN** functions and to limit automatic expansion during record allocation; see **PREALLOCATE**.

Once all indices are structured according to the information supplied, the file is available for key insertion, record allocation and other operations.

(Release 9.4) An index is only encrypted if the index was created by SEARCH mode 0 with 16384 added to the key length.

Note: The encryption status of an index can be determined using **SEARCH** mode 12 which returns the key length of an index with 16384 added if the index is encrypted. **SEARCH** mode 12 is otherwise identical to **SEARCH** mode 1 when used to determine key length. The encryption status of indexes can also be determined by using the **QUERY** utility.

No further mode 0 statements may ever be issued to this file without an exception status occurring.

Mode 1—Miscellaneous Index Information

SEARCH/INDEX *mode* 1 is used to access structure information about an open Indexed File. When the *index* expression is non-zero, the *key length* of the selected index is returned in *record.var*. If the running program is an IRIS program and the file was not structured as a polyfile using **CALL $VOLLINK**, the size is returned in words using the formula INT (size in bytes/2). BITS indexed files, or those created by **MAKEIN** with an odd size key length, will appear to IRIS programs as having 1 less byte.

Specify *index* zero and set *status.var* to select one of the functions listed below. The value (if any) yielded by the function is returned in *record.var*.

0 Return in *record.var* the First Real Data Record as defined during creation.

1 Return in *record.var* the available record count. This is either the value of the environment variable **AVAILREC** if defined, or computed by taking the current size of the file and subtracting the actual number of active records.

2 Allocate a new record in the file returning its value in *record.var*. Possible exception status:

3 = No free records remaining. This condition is only returned when you have set the environment variable **PREALLOCATE** option 128 restricting automatic expansion.

3 De-allocate (return) a record to the file. Available record count is incremented, active records is decremented. *record.var* supplies the record number to mark as 'available'. Possible exception status:

1 = Record number already de-allocated. If you attempt to return the same record twice in a row, this condition is returned. To check the records status before returning it to the Delete List set the environment variable **PREALLOCATE** option to 256.

4 Return in **record.var** the number of physical records within the file for IRIS applications only. Does not include the addition of the First Real Data Record value. Error for BITS programs.

5 Same as mode 4; for IRIS or BITS applications.

6 Set the First Real Data Record to the value supplied in *record.var*. This function is used by the Conversion Programs, and whenever having a record zero is undesirable. This option may only be set prior to freezing the structure with *mode* 0.

7 Return in the current (actual) number of records in use within the file in *record.var*. This number is maintained as records are allocated and de-allocated (See 2 and 3 above).

Mode 2—Search for a Specific Key

SEARCH/INDEX *mode* 2 is used to search an index for an exact match to the supplied *key.var*. If found, *record.var* receives the data record number associated with the *key*, and the *status.var* is set to zero. If no match is found, *record.var* is unchanged and *status.var* is set to one.

A match is indicated when the supplied *key.var* is equal to an entry in the index up to the end of *key*, even if the entry in the file is longer. When the entry is longer, its value is returned in *key.var*.

For example, a search for *key* **ABC** produces a match with the first entry whose first three characters are **ABC**. If the first such entry is **ABC Company East**, then a match is indicated, *key.var* is set to contain **ABC Company East**, *record.var* is set to the associated record number, and *status.var* is set to zero. A match is not produced if the entry in the index is shorter than the key supplied. For example, the entry **AB** is not considered a match.

Note: The actual keys are case-sensitive. This means that "ABC" does not equal "abc."

Mode 3—Search for the Next Highest Key

SEARCH/INDEX *mode* 3 is used to access data records alphabetically, or to search forward from a selected point in the index. The selected *index* is searched for the first entry logically greater than the supplied *key.var*. If found, *record.var* receives the data record number associated with the *key*, and *status.var* is set to zero. When no more entries are found, *record.var* is unchanged and the *status.var* is set to two (End of Index).

For example, a search with *key* **ABC** returns the first entry logically exceeding **ABC**, such as **ABC Company East**. Subsequent mode 3 searches using the same *key* might yield entries such as **ABC Company West**, **Dynamic Concepts**, and **Dynamic Conversions**.

To search an entire *index*, start by setting *key.var* to a null string, and perform *mode* 3 commands until *status.var* is set to 2.

Note that a *mode* 3 search yields the first entry greater than *key*; a *mode* 3 with the key **ABC** does not return **ABC** itself if it exists. It is best to perform a *mode* 2 search first when you want to include the starting *key* in your search.

Mode 4—Insert a New Key into an Index

SEARCH/INDEX *mode* 4 insert new *keys* into an index. The selected *index* is first searched for an entry exactly matching *key.var*. If found, *record.var* is set to the record number associated with the key and *status.var* is set to one.

If no match is found, and sufficient space exists within the selected *index*, *key.var* is inserted in the index using the record number supplied in *record.var* as a pointer to the data record. Successful insertion is indicated by a zero in the *status.var*. If no space exists within the selected index, the *status.var* is set to two (End of Index).

Mode 5—Delete an Existing Key from an Index

SEARCH/INDEX *mode* 5 deletes existing entries from an index. The selected *index* is searched for an entry exactly matching *key.var*. If found, the *key* is removed from the index, *record.var* is set to the record number associated with the key and the *status.var* is set to zero (successful deletion).

If the exact entry is not found, the *record.var* is unchanged and *status.var* is set to one.

Following successful deletion of a *key*, the *record* should be returned for re-use using *mode* 1 with *status.var* set to 3.

Mode 6—Search for a Previous Lower Key

SEARCH/INDEX *mode 6* is used to access data records in descending order, or backward from a selected point in the index. The selected *index* is searched for the first entry logically less than the supplied *key.var*. If found, *record.var* receives the data record number associated with *key*, and *status.var* is set to zero. If not found, *record.var* is unchanged and *status.var* is set to two (End of Index).

For example, a search with the *key* **XYZ** returns the first key found logically less than **XYZ**, such as **Solution Systems**. Subsequent *mode 6* searches using the same *key* might yield keys such as **Solution Concepts**, **Resources International**, etc.

Note that a *mode 6* search yields the first entry less than *key.var*, so a *mode 6* executed with **XYZ** will not yield the **XYZ** itself if it exists. It is best to perform a mode 2 search first when it is desirable to include the starting *key* in your search.

To search an entire *index*, start by setting *key* to "\377\", and perform *mode 6* commands until 2 is returned in *status.var*.

Mode 7—Reorganize Index

SEARCH/INDEX *mode 7* provides for compatibility with older IRIS applications performing an index reorganization. This mode is a non-operation and always returns a *status.var* of zero indicating success and allowing the older program to run without error.

Mode 8—Specify B-Tree Insertion Algorithm

SEARCH/INDEX *mode* 8 retrieves or changes the B-Tree insertion algorithm for an index. If *record.var* is greater or equal to zero, it's value is truncated to an integer and used to select the new insertion method for *index*. If successful, the file's header is changed, and *status.var* is set to zero. If the *record.var* is outside the accepted range, *status.var* is set to one, and no change is made.

If *record.var* is any negative value, the current insertion algorithm used for *index* is returned in *record.var* and *status.var* is set to zero.

Value	Type of Insertion Algorithm Invoked
0	Default. Selects random insertions and is used when keys in the index are inserted in any order.
1	Selects increasing insertions and is used when each key inserted in the index is greater than the previous insertion. Types of keys in this category include sequential order numbers or date keys.
2	Selects decreasing insertions and is used when each key in the index is less than the previous insertion.

Changes are stored in the file's header and become effective immediately for the user storing the

change. Other users must first **CLOSE** and **OPEN** the file before the change takes effect.

The standard **BUILDXF** and **MAKEIN** utilities do not have options for setting the insertion algorithm.

By default, files are created for random insertions. Random insertions split B-Tree nodes when they are half full. This provides a better balancing and room for future insertions.

When sequential keys are inserted (ascending or descending), the nodes should be split only when full. Extra space is not required for later insertions between sequential key values.

The benefits of adding a *mode* 8 to your Application code include saving up to 50% on disk space; 25% increase in performance on insertions, deletions and searches; and less need to run the **ubcompress** utility to release unused space to the system.

Mode 12-Determined encryption status

The encryption status of an index can be determined using **SEARCH** mode 12 which returns the key length of an index with 16384 added if the index is encrypted. **SEARCH** mode 12 is otherwise identical to **SEARCH** mode 1 when used to determine key length. The encryption status of indexes can also be determined by using the **QUERY** utility.

Indexed File Errors & Recovery

If you accidentally delete the ISAM portion of an Indexed file, you can rebuild the file by the following steps.

1. Create a new Indexed file with a different name using the same parameters for number of Indices and Key Lengths.
2. Write a small program to rebuild and insert the keys into the new temporary file. Only insert keys and records, do not copy the existing data.
3. Use the Unix **mv** command to move the new temporary files ISAM portion as the old files ISAM file, i.e.: **mv TEMPFILE MYFILE** or **mv tempfile.idx myfile.idx**. This command must be performed at the shell. Do not use any utilities designed to operate on both portions of ISAM files, such as **COPY** supplied with UniBasic.

If an error is encountered during ISAM file access, an exception (V2=5) status or BITS error #110 (Index file structure error or svar dim < Key Length) may be printed. First, check to see if your string DIM is at least the size of the Key. If so, Print the value of ERR(8) and check the following table for additional information. This table includes all of the c-tree error codes. When using standard Indexed files, only a few of these errors are possible.

Code	Explanation of c-tree Status
0	Successful Operation, No error.
10	Initialization parameters require too much memory.
11	Illegal Initialization parameters: Either **ISAMBUFS** < 3, **ISAMSECT** <1 or **ISAMFILES** < 1.

12	Could not **OPEN** the file. The Index portion is missing, protected or locked by another process.
13	Cannot determine the file type - Corrupted file or Reversed Keys.
14	File appears corrupted and should be checked.
15	Data file has been compacted (CTCMPC), but not cleared through CTRBLD. Rebuild data file (but do not force rebuild).
16	Not enough space to create file or invalid ISAM filename. ISAM filenames must have at least one letter in the filename.
17	Could not create data file. Either no space exists or filename is an improper name.
18	Tried to create existing index portion filename.
19	Tried to create existing data portion filename.
20	Key length too large for node size. There must be room for at least 3 key values per node. The node size is given by **ISAMSECT** *128. Default **ISAMSECT** is 4 resulting in 512-byte nodes.
21	Cannot create data file with record length smaller than 6 bytes.
22	File number out of range; Increase **ISAMFILES** environment variable.
23	Illegal Index Number specified.
24	Could not close file.
26	File number is not in use.
28	Trying to insert a key with a file byte pointer of zero.
29	High level c-tree function called with zero file byte pointer.
30	Selected file byte pointer is beyond the logical length of the file. If the pointer is correct, it is possible that the ISAM header is damaged.
31	Next Record in delete chain does not have 1st byte set to ff (hex). Data File header may be corrupt, or free records were overwritten by the application.
32	Attempt to delete the same record twice in a row. The record being deleted is already the top record on the delete stack. Attempting to return records onto the delete stack more than once may corrupt the file unless the **PREALLOCATE** option is set to 256.
33	File byte pointer is zero using high level c-tree function.
34	Could not find correct predecessor node. Indicates that an index insertion was interrupted before completion. Rebuild index using ubcompress utility.
35	Cannot seek in the file - possibly out of disk space.
36	Cannot read in the file - possible cause: corrupted record position in file.
37	Cannot write to file - possibly out of disk space.
39	Record or node pointers have exceeded 2^32.
40	**ISAMSECT** environment variable was larger when this index was created. Buffers are too small for nodes.

41	Could not unlock data record.
42	Could not obtain a data record lock. Probably the Unix number of locks is too small for the system. Re-configure system.
43	Current configuration parameters inconsistent with the parameters at time of creation. File created under a different Byte swap (Reverse), or file came from an incompatible machine.
Code	**Explanation of c-tree Status**
45	Key length too large.
46	File number is already in use.
48	A function has been called for the wrong type of file, e.g.. a variable length record function used for a fixed length file.
49	Could not write file directory updates to disk during file extension.
50	Could not lock index file node. Probably the Unix number of locks is too small for the system. Re-configure the system.
51	Could not unlock an index file node.
52	Variable length and/or floating point keys disabled in CTOPTN.H.
108	Key number is out of range for the file.
113	Internal Lock Table overflow.
114	First byte of fixed-length data record found by ISAM routine equals delete flag. Attempt to write to a non-allocated record. This exception only occurs when operating with **PREALLOCATE = 4096**.
124	Number of indices in index file does not match information stored in the UniBasic data file header. Either the UniBasic header or the ISAM file is damaged.

Accessing non-UniBasic Files and Devices

Any Unix file may be opened for read/write access by a program. Access is limited by the permissions granted to the user by the creator of the file. If the file is other than a Text File, certain programming precautions must be taken.

If the file is a *character devicename*, data may be read or written from/to the device usually a character at a time. If the file is a *block device*, data must be read or written from/to the device a block at a time. A typical *character device* is a terminal port such as */dev/tty01*; a *block device* might be a magnetic tape drive such as */dev/rct0*.

If the file contains other than ASCII data, process the file as binary using **MAT READ**, **MAT WRITE**, **RDREL** or **WRREL** statements. Use the **CONV** statement to view or alter data within a binary file.

IRIS BCD Data and Key Files

IRIS BCD Data files are standard Contiguous, Formatted or Indexed files whose records conform to IRIS data types. String fields contain IRIS 8-bit strings, and numeric fields are in IRIS BCD

precision.

IRIS BCD Key Files are Indexed Data Files whose keys conform to IRIS 8-bit form. Toggling is performed in and out of each index to guarantee the proper insertion order when binary keys are used.

A file is an IRIS BCD data file if the attribute <Q> has been set. Access to IRIS BCD files adds a small amount of overhead (4%) during access of string fields.

A file is an IRIS BCD Key file if the attribute <K> has been set.

Creating IRIS BCD Data Files

IRIS BCD files are created using **BUILD** or **CREATE** statements. The Supplemental Protection Attributes <Q> and <K> force the new data file to be maintained using IRIS BCD records and/or IRIS 8-bit keys respectively.

Setting the environment variable **PREALLOCATE** option 32 forces all newly created data files to contain IRIS BCD data records or keys.

The **IRIS** Conversion Utilities automatically create IRIS BCD Data files for all converted Contiguous, Formatted, Indexed or Polyfiles where a record definition is not given. IRIS style keys may also be preserved during the conversion.

During conversion, **PREALLOCATE** options must be cleared. Perform the conversion and then set the options as desired.

The following conditions might be reasons to force the creation of new files in IRIS BCD data record or key format:

1. Conversion of an existing end-user's system when the application is totally unknown. Set both <K> and <Q> conditions globally in **PREALLOCATE** after converting all files. Assume all files contain Binary Keys. You may omit binary key conversion and setting <K> if you are sure binary keys are not used.
2. The application makes use of **MAT READ / MAT WRITE** statements to expand files or copy records to history files without regard to the record format. This condition is not supported between mixed Base 10000 and IRIS BCD files. Required toggling and/or conversion is performed one-sided resulting in corrupted data in the destination file. These types of operations are permitted only when both files are of the same class (BCD/ Base 10000). Set global <Q> BCD Data using **PREALLOCATE** options if some of the files have the <Q> attribute following an IRIS Conversion.
3. The application moves data between *num.vars* and *str.vars* using a specially designed **CALL**. Again, this condition is not supported between mixed Base 10000 and IRIS BCD files. Required toggling and/or conversion is performed one-sided resulting in corrupted data in the destination file. Set global <Q> BCD Data using **PREALLOCATE** option if some of the files are <Q> following an IRIS Conversion. In rare cases, Dynamic Concepts may recommend the use of the environment variable **BCDVARS** if the special **CALL** does not support mixed file operations.

Accessing IRIS BCD Data Files

Accessing IRIS BCD Data files is identical to normal Contiguous, Formatted, or Indexed file access. Most applications require no modifications to run with a mixture of IRIS BCD and normal data files.

To preserve the record integrity of both standard and IRIS BCD data files, incoming data (read operation) is converted (if necessary) to match the variable format used by the program. Outgoing data (write operation) is converted (if necessary) to the format of the destination file.

For *str.vars*, incoming data is bit-8 toggled from an IRIS BCD file. Outgoing data is again toggled if written to the same or another IRIS BCD file. Transfer to a normal Base 10000 file does not require toggling. String access to/from IRIS BCD files add about 3% overhead to the transfer.

For *num.vars*, *array.vars* and *mat.vars*, incoming data is placed directly into the variable, and its internal type is changed to the corresponding BCD precision. For example, a variable dimensioned to 4% may internally switch back and forth between precision %4 and %10; See also Numeric Variable Precision. No overhead is required for these operations.

If you transfer a single element of an *array.var* or *mat.var*, that element is converted instead of converting the entire *array.var* or *mat.var*. This operation is negligible, consisting of a load and store of the variable from one data type to another.

Outgoing *num.vars*, *array.vars* and *mat.vars* are converted to the format of the file; that is, Base 10000 or IRIS BCD. Conversion is only performed when a variable's current precision does not match the type of file. This conversion is negligible, simply changing the storage format and not the size occupied by the data.

Each Base 10000 precision has a corresponding BCD precision occupying the same size. Base 10000 does provide additional digits of significance and extra digits may be lost converting from Base 10000 to IRIS BCD. Typical IRIS programs are not affected since they are designed for this lower number of significant digits.

If the environment variable **BCDVARS** is set, all *num.vars*, *array.vars*, and *mat.vars* are allocated and stored internally using BCD precisions (%7-%10). In this mode, conversions are eliminated when all files are IRIS BCD format.

When a file contains IRIS style 8-bit keys as indicated by the Supplemental Attribute <K>, keys are inserted and maintained in the indices in 8-bit form. Toggling is performed to and from the index and is negligible. This condition is required when an application utilizes binary keys and the internal toggling corrupts your programmed insertion order.

An example is when keys contain data both below and above \200\. For example the IRIS Key: \177\\200\\201\ is stored in UniBasic as the string \377\\300\\001\ which alters its order when a sequential search of an index is performed.

See also: PREALLOCATE option 64, Creating IRIS BCD Data Files

Universal Data Files

Universal Data files are IRIS BCD style Contiguous, Formatted or Indexed files which are platform independent. The files are accessible across all Unix platforms. In addition, they are

usable on a Windows system with version 3.1 and higher of dL4 for Windows. Packed data should be avoided for maximum platform independence.

Text files are essentially platform independent, except Unix uses a 'LF' and Windows/DOS uses a 'CRLF' as the line terminator.

A file is a Universal data file if the attribute <U> has been set. The size of a Universal data file is limited to 2^{31} bytes or by the size of the file system. On most operating systems, a Universal file can be created as a "huge" file if the attribute <H> is set. The size of a "huge" file is limited only by the available space on the file system.

It is not necessary to run **ubrebuild** as Universal data files do not use deleted record lists. The **ISAMOFFSET** environment variable is ignored and the user may write data at any location in the record. The **ISAMSECT** environment variable should be set to a value of 8 or less (8 is recommended).

Creating Universal Data Files

Universal data files are created using **BUILD** or **CREATE** statements. The Supplemental Protection Attribute <U> forces the data file to be maintained using Universal records and/or Universal keys.

Universal Indexed Data Files have their keys stored in a companion Index file that has the data file name with a .idx extension, as opposed to the traditional method of using the uppercase of the filename.

Setting the environment variable **PREALLOCATE** option 8192 forces creation of a Universal data file. Setting **PREALLOCATE** option 16384 forces creation of Huge Universal data files.

Accessing Universal Data Files

Accessing Universal Data files is identical to normal non-Universal Contiguous, Formatted, or Indexed file access. Applications can run with a mixture of both Universal and non-Universal data files.

The **ubcompress** utility may be used to reduce the size of the index portion of a Universal Indexed file.

Encrypted Data Files

Encrypted Data files are IRIS BCD style Contiguous or Indexed files in which some or all record fields are encrypted. In an Encrypted Indexed file, some or all of the indexes can also be encrypted. Like Universal files, Encrypted Data files are accessible across all Unix platforms. In addition, they are usable on a Windows system with version 7.1 or higher of dL4 for Windows.

A file is an Encrypted data file if the attribute <N="keyname"> has been set. The key "keyname" (case insensitive) identifies a key definition in the current key list that specifies the encryption

algorithm and the passphrase. The size of a Encrypted data file is limited to 2^{31} bytes or by the size of the file system. On most operating systems, an Encrypted file can be created as a "huge" file if the attribute <H> is set. The size of a "huge" file is limited only by the available space on the file system. Encrypted files must also be Universal files so the <U> or <H> attributes should be used when creating the file unless the **PREALLOCATE** environment variable has been set to restrict all files to universal format.

It is not necessary to run ubrebuild as Encrypted data files do not use deleted record lists. The **ISAMOFFSET** environment variable is ignored and the user may write data at any location in the record. The **ISAMSECT** environment variable should be set to a value of 4 or 8 (8 is recommended and larger values may be needed for very large key sizes).

Creating Encrypted Data Files

Encrypted data files are created using **BUILD** or **CREATE** statements. The Supplemental Protection Attribute <N="keyname"> forces the data file to be maintained using Encrypted records and/or Encrypted keys. The use of quotation marks around the key name is mandatory. The case insensitive key name must exist in the current key list which is loaded from a key file (see **UBKEYFILE**) or defined with **SYSTEM** 100 statements. Encrypted files must also be Universal files so the <U> or <H> attributes should be used when creating the file unless the **PREALLOCATE** environment variable has been set to restrict all files to universal format.

Records in an Encrypted file can be either fully encrypted or they may consist of a mix of encrypted and non-encrypted fields. By default, the entire record is encrypted. A file with fully encrypted records cannot be opened unless the key used by the file is defined in the current key list. A file with partially encrypted records can be opened for reading without the key, but all encrypted fields will be returned as zero filled, space filled, or asterisk filled depending on how the file was created.

To create an Encrypted file with partially encrypted records, the encryption attribute must use the format <N=fld1,fld2,...,fldN,"keyname">. The field definitions "fldN" specify which byte ranges in the record are encrypted. An encrypted record can have up to 16 field definitions (each encrypted field definition may contain as many actual data fields as will fit in the encrypted field). Each field definition has the format S-E where S is the zero based starting byte offset in the record and E is the ending byte offset. The byte range can optionally be followed by the character "Z" to zero fill the field when the record is read without the key, by the character "S" to space fill the field, and "*" to asterisk fill the field. The default behavior is to zero fill encrypted fields.

The following example builds an encrypted contiguous file with 80 character records in which the first 10 characters and the last 6 characters are encrypted:

```
BUILD #1,"<UN=0-9Z,74-79S,CUSTDATA> [1:40] filename"
```

Encrypted Indexed files have their keys stored in a companion Index file that has the data file name with a .idx extension, as opposed to the traditional method of using the uppercase of the filename. By default, the indexes of an Encrypted Indexed file are not encrypted. To create an encrypted index, specify the key length in words plus 16384 in the **SEARCH** or **INDEX** statement that defines the index. The following **IRIS** style program creates an Encrypted Index file with 10 character keys in the non-encrypted index 1 and 14 characters keys in the encrypted index 2:

```
DIM K$[14],R,S
BUILD #1,"<UN=''keyname''>[1:40]filename"
LET R=5 \ SEARCH #1,0,1;K$,R,S \ IF S STOP
LET R=7 + 16384 \ SEARCH #1,0,2;K$,R,S \ IF S STOP
SEARCH #1,0,0;K$,R,S \ IF S STOP
CLOSE #1
```

Setting **PREALLOCATE** option 16384 forces creation of Huge Encrypted data files whenever Encrypted data files are built.

Accessing Encrypted Data Files

Accessing Encrypted Data files is identical to normal non-Encrypted Contiguous or Indexed file access. Encrypted Data files do not require or use any special syntax to open the files. The encryption keys are supplied from the current key list which is loaded from a key file (see **UBKEYFILE**) or using **SYSTEM** 100 statements. Applications can run with a mixture of both Encrypted and non-Encrypted data files.

Special UniBasic Files

Special files are maintained and referenced during a UniBasic session. These files are:

errmessage	Error Messages; BASIC Error descriptions.
sys/term.xxxx	Terminal Input/Output CRT Translation.

Error Message File: errmessage

All BASIC and system error messages are stored in the system Text File *errmessage*. This file must be in the directory search path specified by the Environment Variable **LUST** and is a simple Text File with the format:

```
n:i:Text String for Message
```

The *n* indicates the default error number as defined in Appendix C of this guide. The optional *i* field specifies the IRIS error number to be returned by **SPC(8)** whenever the error is indicated. A default table of IRIS error numbers may be found in Appendix C.

Error codes above 256 correspond to internal Unix errors returned to BASIC. When a system error has no equivalent, a negative error number is returned for **SPC(8)** and **ERR(0)**. The negative number corresponds to the actual Unix system error. For further information on Unix errors, refer to ***errno*** values in your Unix system documentation.

$TERM Files: term.xxxx

Each terminal under UniBasic is assigned a Terminal Type as defined by the environment variable **TERM**.

When UniBasic is launched, or following the execution of the **!** command, a *term* file is opened and read into memory to affect input and output translation for the terminal. The *term* file is located within the **LUST** using the *filename* **term.$TERM**, where **$TERM** is the value of the environment variable **TERM**. For example, if the value of **$TERM** is tvi925, the file *term.tvi925* is located.

An error is generated at startup, or following the **!** command if the *term* file cannot be opened. No output translations are performed and standard input translation characters are not defined.

See also: Terminal Translation File $TERM Files, Input/Output

Device Input and Output

Port Numbering

The Unix system does not provide Ports or Port Numbering in the traditional sense. Each process may or may not have a *tty* character device opened for input and output. When signing onto the system, your process has a system *tty* channel opened which is connected to your terminal.

A *port number* is a unique value from 0 to 4095 assigned to your terminal when launching a UniBasic session. The *port number* is required for communication between applications and users by the **SIGNAL**, **SEND** and **RECV** statements.

Upon initial entry, a *message queue* is created, a *port number* is established. When the session terminates, the *message queue* is deleted and the associated *port number* becomes available again. A *port number* is established by the successful completion of one of the following steps:

1. If the Environment Variable **PORT** is defined, its value selects the *port number* for this session. If another UniBasic process is already established as the same port number, your session terminates with an error message.
2. If the Environment Variable **PORTS** is defined, the list is searched for the system *tty* name and, if found establishes the *port number* for this session. The system *tty* name assigned to your terminal is available using the Unix command: **who am i**.
3. The system *tty* name is interpreted searching for trailing digits to use as an identifying *port number*. For example, *tty23* selects *port number* 23. Many systems use system *tty* naming conventions such as *tty1a*, *tty1b*, etc. These usually require definition of the environment variable **PORTS** to ensure consistent selection of a *port number*.

If a *port number* cannot be established using one of the above steps, the *message queues* are scanned backwards from the value of the Environment Variable **MAXPORT** (default 999) for the first unused *port number*.

An error is generated, and the session is terminated if you attempt to utilize a *port number* already signed on and in use.

You may initiate multiple UniBasic sessions, with different port numbers, from the same terminal..

Suppose you have an application error and wish to launch another session without going to another terminal. While in *BASIC program mode*, issue the command:

```
!PORT=nnn ; UniBasic
```

where: 'nnn' is an unused *port number*.

To cancel secondary session(s), issue a **BYE** command.

Phantom Ports

A *phantom port* is any UniBasic session which is not connected to a character *tty*. That is no *tty* is opened for the process. All input for the session must be transmitted from another *port number*, and output must be re-directed to a *file* or *device* or it is discarded.

Communication to a *phantom port* is restricted to the statements **SEND**, **RECV**, **PORT**, **CALL $TRXCO**, **CALL 98**, and **SIGNAL**. An application may control any active UniBasic process whether or not it has an opened system *tty* device.

A *phantom port* is initialized using either **CALL $TRXCO** or the **PORT** statement. A *port number* is supplied for these operations. The active *message queues* are interrogated to determine whether an active process is already assigned to that *port number*. If so, an error status is returned to the parent and no process is launched. If the *port number* is not in use, a phantom port initialization proceeds.

Commands may be transmitted to a *phantom port* or an interactive *port number* which has an active *message queue* entry. When transmitted to an interactive *port number*, commands become input to the interactive process. Data is echoed on the terminal as if it were entered on the keyboard.

If the *port number* does not have an active *message queue*, a copy of your process is forked (duplicated). The new child process immediately severs its connection to you as the parent. It assumes your environment and default working directory, but closes the system *tty* channels re-directing all input and output to the */dev/null* file. A new *message queue* is created for the specified *port number* which now becomes a *phantom port*. It is available to all other users for communication and transmission of commands.

Note: Commands may only be transmitted to a *port number* which is actively running a UniBasic process and has an assigned Message Queue.

Simply defining */dev/tty23* to be *port number* 23 does not provide for communication until *port number* 23 actually launches a UniBasic process. To send commands to an interactive port, first login to Unix and launch a UniBasic process.

When connecting modems or other non-keyboard devices that you wish to control using **CALL $TRXCO** or **PORT** statements, configure your Unix system to automatically launch a UniBasic process on a known *port number* for communication. You might also communicate with a modem or *other device* by directly opening the device, and using standard **READ** and **WRITE** statements.

See also: Launching UniBasic Ports at Startup.

Accessing Drivers ($LPT) and Pipes

A *pipe* may be opened for input to or output from a BASIC program. An output *pipe* is the mechanism whereby another Unix *process* is started and your output to a channel *pipes* the data as standard input to the new process. An input *pipe* is the mechanism whereby another Unix *process* is started and its output becomes your input on the opened channel.

To open a pipe, the filename must be the name of an *executable* Unix program or shell script, that is, the permissions of the file must include 'x'. To open an input pipe, the filename is preceded by two dollar-signs ($$name); an output pipe is preceded by a single dollar sign ($name).

Unlike IRIS or BITS, the $ character is not part of the *filename*. It is a flag indicating the desire to access another process using a *pipe*. The *filename* does not select a data file, but instead selects the name of a Unix executable command or shell script. If you must specify a full Unix *pathname*, the $ or $$ must be the first character in the filename string, such as $/bin/ls. When opening pipes to processes found within your defined directory search list, as defined by the Environment Variable **LUST**, the $ or $$ may be the first character of the string, or the first character of the filename, such as $23/lpt or 23/$lpt. In general you may establish a *pipe* to any command accepted by the shell, such as ls, cat, or lpt.

Printer driver scripts ($LPT) are examples of output *pipes*. Your application opens $LPT. The **LUST** is searched for the *filename* "lpt". If the file is executable, it is started as a process and a *pipe* is established on the specified channel. As you **PRINT** to the channel, "lpt" receives the data and processes the output. It may re-direct the output to a physical device, or through the spooler. When you **CLOSE** the channel, "lpt" receives an end-of-file and terminates.

An example of an input *pipe* might be to read a list of all filenames stored in a directory. By opening the file "$$/bin/ls {pathname}", you read the output of the Unix **ls** command as if it were a Text File. A null string (IRIS) or end-of-file error (BITS) is generated when the *pipe* is empty.

If a *filename* specified with a $ or $$ is not within the paths specified by **LUST**, the entire *pathname* must be specified to **OPEN**.

When creating your own C or shell scripts to be used as *pipes*, always make a backup copy, ensure that the *file* is executable and does not have write permission enabled. This prevents accidental overwriting if the $ is omitted.

Note: The $ and $$ are only flags used during **OPEN**. You may still **OPEN** any Unix file, according to the rules of Text File access, regardless of executable attributes.

When processing from/to a *pipe*, many systems buffer data in blocks of 4096 characters, or until the appropriate process terminates. You cannot use pipes for simple communication between processes. To transmit data between application programs, use the **SIGNAL** or **SEND/RECV** Statements.

When an **OPEN** is requested using the $ prefix, the following operations are performed by UniBasic:

1. The file is checked for execute permission, and that the file is not a directory. If these conditions are met, a pipe operation is attempted. If not, the file is opened

as a standard Text File and those rules apply.

2. The file is opened and the first 5-characters are read from the file.
3. A check is made for a string with the first 5-characters as '#lock'. Other characters on the line are ignored. Any preceding spaces, tabs or blank lines cause a failure of the test.

   If '#lock' is seen, the system checks the */tmp* directory for a *filename* in the form **/tmp/lk.inode**, where 'inode' is the Unix *i-node* number of the executable script being opened.

   If the file is not found, it is created to prevent other users from opening the same printer script. This file contains the process ID (PID) and UniBasic port number of the process requesting the lock. Its permissions are set to allow reading and writing by all levels, but it should neither be written to by a user nor its permissions changed. This file may be read by a user to determine what process or port has the printer locked.

   If the file exists, the system reads the port number and verifies that the port is an active port.

   If the port is active, the error 'Device is in use and Locked' is generated to the BASIC program, and the operation fails.

   If the port is not active, as in the case of a lock file not being removed because of a previous system failure, the system overwrites the existing lock file with the current requesting process.

   If the '#lock' string is not seen, a lock file is not created. The script itself must guarantee against multi-user access, and most likely will rely on the Unix Spooling System.

Note: The Unix Spooling System is the preferred way to use a printer.

---

4. The file is now launched as a process, and a pipe is opened on the users channel to it.

Printer Drivers

A printer driver is nothing more than an executable Unix shell script opened from your application as a *pipe*. Whenever an opened filename begins with a $, and is executable, a *pipe* is established and the selected filename (without the $) is started as a process reading as its input the data you **PRINT** to the *pipe*.

The supplied **lpt.iris** and **lpt.bits** are sample printer shell scripts for IRIS and BITS applications respectively. The main difference is that the IRIS driver utilizes locking (only a single user may access the device until closed), whereas the BITS driver passes its data to the Unix spooling system.

The sample **lpt.iris** driver may be modified using the Unix editor. It is designed to operate using the bourne shell. Once complete, it is copied and given different names such as LPT, LPT1, etc. as

required by your applications. For information on the supplied printer scripts and configuring serial ports for printers, see also Configuring Printer Drivers and Configuring Serial Printers.

Mail Drivers

A mail driver is nothing more than an executable Unix shell script opened from your application as a *pipe*. Whenever an opened filename begins with a $, and is executable, a *pipe* is established and the selected filename (without the $) is started as a process reading as its input the data you **PRINT** to the *pipe*.

The supplied **email.mail** and **email.sendmail** are sample shell scripts that allow UniBasic applications to send email. The main difference is that the **email.mail** driver utilizes the Unix **mail** command, whereas the **email.sendmail** driver passes its data to the Unix sendmail facility.

The sample mail drivers are designed to operate using the bourne shell. To use these drivers, their path must be in the **LUST** environment variable and have read and execute permissions. Read and execute permissions must be set prior to use.

The driver scripts may be renamed and/or modified for your application using any Unix editor.

The following is an example of using the **email.mail** script in a UniBasic application:

```
10 OPEN #1,"$email.mail -s test -t recipient"
20 PRINT #1, "Test Message"
30 CLOSE #1
```

The UniBasic **OPEN** statement opens the **email.mail** pipe driver script with arguments to specify the subject and recipient of the email. The -s option is followed by the subject and the -t option is followed by the recipient list. All data **PRINT**ed to the channel will be sent to the email pipe until the channel is **CLOSE**d, which closes the pipe and sends the mail.

The mail program on some Unix systems may not require and/or accept the -s option. Consult your system documentation and the man pages for your particular requirements. The driver script may be modified as necessary to work with the mail program. For example, **mail** will not accept the -s option with a subject on the command line, while **mailx** requires the -s option and a subject on the command line or it will interactively request a subject. UnixWare has both **mail** and **mailx** programs, whereas SCO only has the **mailx** program; the **mail** command is a link to **mailx**.

The **email.mail** driver script may be tested outside the UniBasic environment by issuing the following command, modified as necessary, at the Unix prompt:

```
$ echo body of mail |email.mail -s test -t your_userID
```

Verify that the mail is received.

The **email.sendmail** pipe driver script was written to interface with the **sendmail** program. If used, the user must identify the location of sendmail on the system and modify the assignment to the variable 'EMAILPATH' in the **email.sendmail** script. The command line arguments to **email.sendmail** must include a -f option with the sender specified. For example, the UniBasic **OPEN** statement above becomes:

```
10 OPEN #1,"$email.sendmail -s test -f sender -t recipient"
```

A -r option to specify a 'Reply-to:' field in the arguments is optional.

The **email.sendmail** driver script may be tested outside the UniBasic environment by issuing the following command at the Unix prompt:

```
$ echo text | email.sendmail -s test -f your_userID -t your_userID
```

Verify that the mail is recieved.

Terminal Translation File $TERM Files

Each terminal under Unix is assigned a Terminal Type as defined by the environment variable **TERM**.

When UniBasic is launched, or following the execution of the **!** command, a *term* file is opened and read into memory to affect input and output translation for the terminal. The *term* file is located within the **LUST** using the *filename* **term.$TERM**, where **$TERM** is the value of the environment variable **TERM**. For example, if the value of **$TERM** is *tvi925*, the file *term.tvi925* is located.

An error is generated at startup, or following the **!** command if the *term* file cannot be opened. No output translations are performed, and standard input translation characters are defined.

Terminals operate in two distinct modes: Normal and Window Tracking. Normal mode allows the terminal to control the operations based upon sequences and data transmitted. The effect of wrap-around from line to line, scrolling, behavior of protected regions is terminal dependent. Applications transmitting mnemonics (such as **CS** to clear screen) retrieve a replacement string unique to the terminal for the mnemonic function selected from the *term* file.

Window Tracking is a feature whereby software maintains a tracking map of all characters and attributes on the screen. Little or no overhead is caused by the maintenance of this map. Its purpose is to simulate features not supported by most terminals, such as multi-screen windows. In addition, Window Tracking is effective for PC Monitors and other dumb terminals lacking features such as Protected Fields. For a complete discussion of Window Tracking, see Using Dynamic Windows.

The *term* file is a standard Unix Text file. The file contains four types of information:

$TERM Flags and Switches

Mnemonics Translated for Output

$TERM Extended Graphic Mnemonics

$TERM Input Character Processing

$TERM Flags and Switches

Flags and switches control the formatting of cursor addressing strings sent to the terminal.

When a cursor address is specified **@x,y;**, the system first outputs the mnemonic **PC** as the lead-in. This mnemonic is defined to be the codes required by a terminal to expect a cursor addressing sequence. The following steps are then performed on the supplied cursor address:

1. The coordinates are converted to integer and checked against the limits as specified by the *max_x* and *max_y* definitions.
2. If the *xy-direc* flag is zero, then the integer X and Y values are added to the *base_x* and *base_y* parameters. A non-zero value in *xy_direc* causes a subtraction of the *base_x* and *base_y* parameters.
3. The X and Y values are then readied for transmission to the screen. If *xy_ascii* is zero, no further processing is performed. The single characters X and Y are ready for output. If, however, *xy_ascii* is non-zero, the X and Y values are converted to ASCII digits for output. For example if X is 23, the resulting outputs are the digits '2' and '3'. This feature is required on most ANSI terminals and PC monitors.
4. If *xy_order* is non-zero, the Y value is output first, otherwise X is assumed to be first.
5. If *pc_leadin* is non-zero, then the mnemonic **PC1** is sent following the transmission of the first coordinate (X or Y from step 4).
6. The second coordinate is transmitted.
7. If the mnemonic **PC2** is defined as other than null, it is output to terminate a cursor addressing sequence.

Flag name	Description / Effect on operation
xy_order:n	Selects the order that the coordinates are sent to the terminal. A zero specifies XY order (column/row as in the BASIC application), and non-zero selects YX (row/column) order.
xy_direc:n	A zero (default) selects that the coordinates are to be added to the base_x or base_y character specifications. A non-zero flag causes the coordinates to be subtracted from the base_x and base_y ASCII characters.
xy_ascii:n	A zero specifies that the coordinates sent to the terminal are the result of simply adding/subtracting the coordinate from the base_x or base_y specification. A non-zero flag causes the coordinates to be output as ASCII digits (i.e. ANSI terminals).
base_x:n	The base ASCII character in decimal to be added to (or subtracted from) a cursor positioning operation to a column (x coordinate). This value represents X coordinate zero.
base_y:n	The base ASCII character in decimal to be added to (or subtracted from) a cursor positioning operation to a row (y coordinate). This value represents Y coordinate zero.
max_x:n	The maximum number of columns on the terminal in all modes.
max_y:n	The maximum number of lines on the terminal in all modes.
pc_leadin:n	A non-zero flag specifies that a special lead-in is required between the two coordinates. If this flag is set, then the mnemonic **PC1** is transmitted after the first coordinate.
crt_type:n	Define the terminal type to be returned to the BASIC program by the

MSC(32) and **SPC(13)** functions. Set the type to 23 for ANSI monitors and other ANSI terminals to force them to behave more like conventional terminals with respect to protected fields. When set, Window Tracking is always enabled.

crt_flags:n. Flags controlling character transmission and interpretation of mnemonics for Dynamic Windows. Multiple options are set by adding the values together.

Mode	Description
1	Convert \200\ characters to \0\ on output. Required on older NCR Tower systems, and other conditions where you always want \200\ nulls transmitted as true zero bytes before transmission to a serial port.
2	Interpretation of Format Mode using Dynamic Windows. Set for BITS programs and applications relying on only **BP/EP** to set both format and write-protected. In this mode, **BP** is defined to perform both **FX** and **BP** functions, and **EP** performs both **EP** and **FM** functions. Clear for IRIS programs and applications using **BP/EP** to write protect and **FM/FX** to control Format Mode separately.
4	Interpretation of Dimmed Mode using Dynamic Windows. Set for BITS programs and applications relying on dimmed characters to be normally unprotected. Clear for IRIS programs and applications using **BD/ED** as replacements for **BP/EP**. In this mode, the **BD/ED** mnemonics cannot be defined with a size greater than zero; See Defining TERM Mnemonics.
8	Controls the function of the **CL** and **CE** mnemonics. Set to clear up to the End Of Line or until the first protected character. Normally, **CL** and **CE** clear unconditionally to the end of the line or screen, skipping over protected fields.
48	When set, **MD** does not cause the terminal to scroll on the last line. When clear, **MD** causes the terminal to scroll on the last line.
192	When set, **MR** does nothing at the last position of the screen. When clear, **MR** moves the cursor forward either to the first position of the new last line after scrolling the screen or, if the terminal is in Format Mode (see **FM/FX**), home without scrolling.
768	When set, the terminal does not reposition the cursor after it receives an **IL** or **DL**. When clear, the terminal positions the cursor to the beginning of the current line after it receives an **IL** or **DL**.

The default value for *crt_flags* is 0. Values of *crt_flags* in the term file can be expressed in decimal, octal, or hexadecimal form. For example, to represent the decimal value 16, simply enter 16. To represent an octal value, proceed the value with a zero, i.e. 020. To represent a hexadecimal value, proceed the value with a zero+x, i.e. 0x10.

Note: Because screen behavior differs from terminal to terminal, it is not recommended that you rely on any specific behavior when developing applications. This can lead to incompatible and/or non-portable software in the future.

The behavior of the mnemonics mentioned above is undefined within UniBasic at this time, however future versions may clarify their operation. Proposed future changes more clearly define the interface between software and the terminal screen or window. As a result, do not rely on *crt.flag* settings when developing applications. Rather, they are included to provide varying degrees of compatibility for older applications running within in Window Tracking mode.

Note: Because Window Tracking mode is a simulation and many applications are coded to a specific behavior, not all applications will behave identically in Normal mode and Window Tracking mode.

All printable characters are maintained with their attributes for Protect, Reverse, Underline, Blink, Graphics and Dimmed. Printable characters overflow the right edge of a window and wrap automatically to the first column of the window on the next line.

The following examples illustrate use of *crt_flags* with Wyse 50 terminals:

Programs using **BD** and **ED** to protect characters and **FM** to turn on format mode (IRIS style):

```
1020 PRINT 'BD' "This is protected!"'EDFM'
```

term.wyse50 file settings:

```
crt_flags:0
BP:\33\)
EP:\33\(
BD:\33\)
ED:\33\(
FM:\33\&
FX:\33\'
```

Programs using **BP** and **EP** to protect characters and turn on format mode (BITS style):

```
1020 PRINT 'BP'"This is protected!"'EP'
```

term.wyse50 file settings:

```
crt_flags:2
BP:\33\'\33\)
EP:\33\(\33\&
BD:\33\)
ED:\33\(
FM:
FX:
```

Programs using **BP** and **EP** to protect characters and **FM** and **FX** to turn on and off format mode, retaining dimmed characters (**BD/ED**) as unprotected:

```
1020 PRINT 'FXBP'"Protected,"'EPFMBD'"Dim!"'ED'
```

term.wyse50 file settings:

```
crt_flags:4
BP:\33\)
EP:\33\(
BD,1:\33\GP
ED,1:\33\G0
FM:\33\&
FX:\33\'
```

Note: Because a Wyse 50 terminal treats dimmed mnemonics as embedded, the BD and ED must be defined with a size of 1. See Defining $TERM Mnemonics.

Unfortunately, not all terminals are created equal. For example, the Wyse 60 uses a bit map to store every possible attribute combination for each character. When a Begin Underline is sent, it places the cursor into Underline mode and sets an underline bit for each successive character until Underline is turned off.

In contrast, the Wyse 50 terminal uses a physical screen position to store the actual attribute. When a Begin Underline is sent, it is stored at the position of the cursor and the cursor is advanced one position. From that position to the end of the screen is immediately underlined. When Underline is turned off, underlining is cleared from that position to the end of the screen.

Defining $TERM Mnemonics

The standard format within a *term* file for defining a mnemonic replacement string is:

```
MNEMONIC {,size} :replacement string
CS:\33\*
BPW:\33\Gw
G1:[
BU,1:\33\g
```

The MNEMONIC is any 2-character supported mnemonic (in either upper or lower case) used by an application. In addition, the three-letter mnemonics **BPW EPW** may be defined as special replacements for **BP/EP** used exclusively during Window Tracking operations.

Extended Graphics characters are defined using the form: **Gn** where **n** selects one of 28 (**G1-G28**) special graphics sequences.

IOxx mnemonics control internal system flags and are not defined as output replacement strings.

The optional *size* specifies the number of screen positions required to hold the mnemonic. The default is zero positions for all attributes other than **BH**, **BV**, **PI** and Extended Graphics Characters which assume one position. Include a *size* definition for specific mnemonics when:

1. Applications assume, or the terminal itself outputs a space as part of the mnemonic - i.e. certain Wyse 50 mnemonics.

2. You modify attributes, adding a space to compensate for assumptions made with older terminals.
3. You are using user-defined mnemonics **SA**, etc., or color mnemonics for graphics or other functions that physically require one or more spaces on the screen for display.
4. Your *replacement string* contains one or more printable characters.

The colon and *text* define the replacement string generated when the data is transmitted to a terminal. The *text* may be made up of printable characters or octal characters in the form \xxx\.

Sequences are transmitted directly to the system without toggling. Printable characters should be less than \200\. Only use codes greater than \200\ when a local printer or screen calls for a special graphics function and the terminal / printer manual specifies use of an 8-bit code.

Mnemonics Translated for Output

As discussed in the beginning of this section, printable characters are stored and operated internally using 7-bit ASCII codes. For programming ease, and compatible program source, octal values used within *str.lits* are toggled internally such that \215\ is toggled to the printable \015\, and \015\ is toggled to be \215\; same as the mnemonic **CR**. In the same way, entry of \301\ toggles to \101\ producing the ASCII character 'A'. Mnemonics are converted to an 8-bit character such that **CS** has the same representation as an IRIS \020\ *str.lit* after toggling.

In simpler terms, *str.lits* and mnemonics are toggled internally from their IRIS/BITS form to match the industry-standard without re-programming.

Mnemonics use the same coding sequence as IRIS, allowing octal code representations of crt functions.

When data is processed for output, a stream of characters is produced containing codes less than \200\ for printable characters, and greater than \200\ for crt mnemonics, functions or special printer/device control. To transmit codes above \200\, program *str.lits* using codes less than \200\ and vice-versa.

The **IOBO** mnemonic and **SYSTEM 16** statements enable Binary Output Mode and prevent toggling with the **ASC**, **CHR** and **CALL $STRING** functions.

All mnemonics required by an application must be defined within the *term* file. Use of an undefined mnemonic results in the transmission of the internal code used to represent the mnemonic. These codes are in the range \200\ to \377\. Some terminals may interpret these codes leading to undesirable results on the screen.

When outputting through a *channel*, the internal code is passed through. This facilitates device independence and the ability to write text containing mnemonics to a file. Later retrieval and transmission to a terminal substitutes the *replacement strings* required. When mnemonics are sent to through a *pipe* to a *device*, the supplied **lptfilter** utility may be included within the *pipe* to substitute *replacement strings* for a device.

See also: Configuring Printer Drivers, lptfilter Utility

CRT Mnemonics

CRT mnemonics are used in conjunction with a CRT *term* file to provide control of video terminal functions such as clear-screen, reverse-video, etc. CRT mnemonics appear in one of two forms:

- A set of one or more 2-character codes enclosed in single quotation marks ('). Each code can be preceded by an optional count value.
- A cursor address in the form: @*num.expr*, *num.expr;*. Addresses are given in the form *column*, *row* from origin 0,0 home (upper-left of screen).

For example:

```
'CS'            Clear screen

'CS10ML'        Clear and move left 10 positions.

@5,5;'CL'       Position to column 5, row 5 and clear line

@10,L;          Position cursor to column 10, row L.
```

See also: Using Dynamic Windows, Terminal Translation

Appendix B shows these mnemonics and their octal replacement value when used within *str.lits*, and their internal representation in files or when sent to a *device* or *pipe*.

This section lists the output mnemonics within their general functional area. The functional areas are:

- Keyboard and aux port
- Clear & reset the terminal
- Cursor position
- Control attributes
- Control color
- Transmit data
- Miscellaneous functions
- I/O control
- I/O mnemonics not supported

Mnemonics for Keyboard and Auxport

Mnemonic	Description
AE	Enable the Auxiliary port on the terminal. This mnemonic enables the Auxiliary Printer port until the **AD** mnemonic is sent.
AD	Disable the Auxiliary port on the back of the terminal.
BA	Begin Transparent output to Auxiliary printer port. Enabling Transparent

output causes all output characters (and input echoing) to be directed to the Auxiliary Port of the terminal until the mnemonic **EA** is sent.

BO Begin non-Transparent output to Auxiliary printer port. This mnemonic operates similar to the 'BA' mnemonic except that data is transmitted to both the Auxiliary port and the screen until an **EO** mnemonic is sent.

EA End Transparent output to Auxiliary port.

EO End non-Transparent output to Auxiliary port.

EF End Function Key Definition. This code terminates all characters being sent to down-load function keys using the mnemonics **P1** through **P8**.

LK Lock Keyboard. The keyboard is locked and no further characters are accepted from the terminal. All keys are locked out until the **UK** mnemonic is sent or until the terminal is reset.

P1 Begin Programming down-loadable function key 1. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P2 Begin Programming down-loadable function key 2. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P3 Begin Programming down-loadable function key 3. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P4 Begin Programming down-loadable function key 4. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P5 Begin Programming down-loadable function key 5. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P6 Begin Programming down-loadable function key 6. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P7 Begin Programming down-loadable function key 7. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

P8 Begin Programming down-loadable function key 8. All further characters are sent to the terminal's function key until the mnemonic **EF** is sent.

RF Reset Function keys to their default values.

UK UnLock Keyboard. Characters and functions may now be entered from the keyboard.

Mnemonics to Clear & Reset the Terminal

Mnemonic	Description
CE	Clear from cursor to end of screen. All unprotected characters from the current cursor position up to the end of the screen are cleared.
CL	Clear from cursor to end of line. All unprotected characters from the current cursor up to the end of the line are cleared.
CS	Clear the entire screen. All characters both protected and unprotected are cleared.

CT	Clear all TAB Stops set by the **ST** mnemonic.
CU	Clear all unprotected characters on the screen. This mnemonic is used to clear data from the screen while leaving any protected mask intact. Also, performs a Move Home (**MH**), if window tracking is on. The cursor is moved to position 0,0 of the current window.
ES	End Write Status Line. Characters output and echoed are no longer displayed in the status line of the terminal (See also: **WS**).
K0	CURSOR Set no cursor to be displayed on the terminal.
K1	CURSOR Set Blinking Block.
K2	CURSOR Set Steady Block.
K3	CURSOR Set Blinking Underline.
K4	CURSOR Set Steady Underline
NR	Narrow Display. Set 132 column mode and display further output and echoed characters in narrow format.
NV	Normal video. Display reverse video as dark on lighted background.
RS	Reset Terminal. Send the commands to reset the terminal to its power-up parameters. This normally resets baud, protocols, translations, function keys and clears the screen.
RV	Reverse video. Display reverse video as lighted characters on dark background.
SF	Status Line OFF. Turn off the optional status line at the bottom (or top) of the screen.
SO	Status Line ON. Turn on the optional status line at the bottom (or top) of the screen.
WD	Wide Display. Set the terminal into 80 column mode and display further output and echoed characters in normal format.
WS	Write Status Line. All further characters echoed or output are displayed in the terminals status line until the **ES** mnemonic is sent.
XX	Initialize Terminal. This mnemonic can define a series of functions such as Clear screen, Clear Memory, Clear Status Line, etc. required to reset the terminal; See also: **RS**.

Mnemonics Applied to the Cursor Position

Mnemonic	Description
BK	Cursor Back. A carriage return without line-feed is sent to the screen moving the cursor to the beginning of the current line. Since Unix output post processing normally converts a \215\ into \215\\212\, it may not be possible to send only a return.
CR	Perform a new-line operation. A carriage return and a line-feed are sent to the terminal. If the cursor is at the bottom of the window, the screen will scroll up

one line. Some terminals will not scroll if the screen window contains protected fields.

DC Delete Character. The character at the cursor is deleted and all remaining characters on the line are shifted left.

DL Delete Line. The line containing the current cursor is deleted from the window and all remaining lines are moved up.

FF Form Feed. Scroll to the next page. This mnemonic is used primarily for printers using the supplied **lptfilter**(u), when directing data through the Auxiliary printer port.

IC Insert Character. A space is added at the current cursor position by shifting the character under the cursor (and all remaining characters on the line) right one position.

IL Insert Line. A new line is added by shifting the line containing the cursor (and all following lines) down one line. Lines may disappear off the end of a window.

LF Perform a Line-Feed. This, in effect, is identical to a **MD** mnemonic. The cursor is moved down to the next line while staying at the same column.

MD Move Down. The cursor is moved down to the next line while staying at the same column. Some terminals will scroll if you are already on the last line of the screen.

MH Move Home. The cursor is moved to position 0,0 of the current window.

ML Move Left. The cursor is moved Left one character.

MP Use Memory Pointer instead of cursor for next positioning command.

MR Move Right. The cursor is moved Right one character.

MU Move Up. The cursor is moved up to the previous line while staying at the same column.

PC Position CURSOR; Lead-in sequence. This mnemonic is not used directly. **PC** as well as **PC1**/**PC2** are shown here for documentation purposes only. These mnemonics are output when a cursor address **@x,y**; is specified. The sequence sent is: **PC** lead-in, coordinate 1, **PC1** separator, coordinate 2, **PC2** trailer.

PC1 Position Cursor separator. Defined when a sequence is required between the X and Y coordinates in cursor addressing. Not normally output directly by the application program.

PC2 Position Cursor trailer. Defined when a sequence is required after sending the second coordinate position. Not normally output directly by the application program.

TB Tab Backward. The cursor is moved to the start of the previous TAB Stop as defined with the **ST** mnemonic.

TF Tab Forward. The cursor is moved to the start of the next TAB Stop as defined with the **ST** mnemonic.

VT Vertical Tab. Move the cursor Down in the window to the next preset Vertical Tab Stop. This mnemonic is normally used for printers using the supplied

lptfilter(u), or when you direct data through the Auxiliary printer port.

Mnemonics to Control Attributes

Mnemonic	Description
BB	Begin Blink Mode. All further output and echoed characters will blink until the **EB** mnemonic is sent.
BD	Begin Dimmed Intensity Mode. All further output and echoed characters will be displayed in Dimmed (half) intensity until the **ED** mnemonic is sent. Some terminals will treat dimmed intensity data as protectable and use of the **FM** mnemonic will cause dimmed fields to become protected.
BG	Begin Graphics Mode. All further output characters are translated upon the special Graphics Control Sequences defined the default *term*. file. Each of the 256 ASCII characters conform to the special graphics characters **GRnnn**. Normal character and CRT translation is disabled.
BP	Begin Protectable Field. Further characters echoed or sent to the terminal are flagged as protectable and are usually displayed in half-intensity. Similarly, half-intensity data printed using the **BD** mnemonic may also be protectable, depending upon your terminal. After you have painted your protectable fields on the terminal, you must issue the **FM** mnemonic to format and write-protect your protected field.
BPW	Display format for Beginning a Protected field when using dynamic windows. To simulate protected fields, normally, **BD** and **ED** mnemonics are used. This definition in the term file provides an alternate sequence, such as; reverse video, underlined or color, to denote protection. This mnemonic is not used within the program, rather it is output in place of **BP** when Window Tracking is enabled.
BR	Begin Reversed Video . All further output and echoed characters will be displayed in reverse video format. On most terminals, the background will become lit and the characters are shown as black. Color monitors and other terminals may permit control of the display.
BU	Begin Underline Mode. All further output and echoed characters will be underlined until the **EU** mnemonic is sent.
BX	Begin Expanded Print. All further output and echoed characters will be displayed in your pre-defined choice of double-high, double-wide or both.
EB	End Blink Mode. Characters output and echoed will no longer blink.
ED	End Dimmed Mode. Characters output and echoed will no longer be in half-intensity.
EG	End Graphics Mode. Normal terminal translation is restored. Printable characters represent themselves and CRT codes are processed normally.
EP	End Protectable Field. All further characters transmitted are not to be considered part of a protected field.
EPW	End Protected special display for Window Tracking. Used in conjunction with

BPW replacing **BP/EP** simulated use of **BD/ED** mnemonics.

ER	End Reversed Video. Characters output and echoed will no longer be in reverse video format.
EU	End Underline Mode. Characters output and echoed will no longer be underlined.
EX	End Expanded Print. Characters output or echoed will no longer be in expanded format.
FM	Enter Format Mode. Write protect is set on all characters previously sent using the **BP** mnemonic. The protectable fields are now protected preventing any overwriting of protected data. On some terminals, dimmed characters (**BD**) may also become protected.
FX	Exit Format Mode. All previously write-protected characters are now returned to their protectable state. Fields can be overwritten or changed until another **FM** is issued. Some terminals cannot overwrite protected characters once formatted by the **FM** mnemonic. A clear-screen (**CS**) is required to reset these fields.
ST	Set a TAB Stop at the cursor. To be used with the **TF** and **TB** mnemonics for presetting TAB stops on the screen.

Mnemonics to Control Color

Mnemonic	Description
RE	Color RED. All further output and echoed characters are displayed in Red.
GR	Color GREEN. All further output and echoed characters are displayed in Green.
YE	Color YELLOW. All further output and echoed characters are displayed in Yellow.
BL	Color BLUE. All further output and echoed characters are displayed in Blue.
BK	Color BLACK. All further output and echoed characters are displayed in Black.
MA	Color MAGENTA. All further output and echoed characters are displayed in Magenta.
CY	Color CYAN. All further output and echoed characters are displayed in Cyan.
WH	Color WHITE. All further output and echoed characters are displayed in White.

Mnemonics to Transmit Data

Mnemonic	Description
BT	Begin Transmission. Begin transmitting all characters from the terminals memory. This function is highly terminal dependent.

ET	End Transmission. Disable transmission of characters from the terminal's memory.
LU	Send Line Unprotected. All non-protected characters from the current cursor through the end of the line are transmitted from the terminal.
PS	Print Screen. Send the contents of the current screen through the terminal's Auxiliary/Printer port.
PU	Send Page Unprotected. All unprotected characters on the screen are transmitted from the screen to the system.
SL	Send Line All. All characters (including protected fields) on the line containing the cursor are transmitted from the screen to the system.
SP	Send Page All. All characters (including protected fields) on the screen are transmitted to the system.
TL	Transmit Line unprotected. All non-protected characters from the current cursor through the end of the line are transmitted from the terminal.
TP	Transmit Line protected. All characters (including protected fields) on the screen from the current cursor to the end of the screen are transmitted to the system. NOTE: **TP** may also be used by BITS Applications to Toggle Pages of screen memory.
TR	Transmit Screen unprotected. All non-protected characters from the current cursor through the end of the screen are transmitted from the terminal.
TS	Transmit Screen protected. All characters from the current cursor through the end of the screen are transmitted from the terminal.

Miscellaneous Mnemonics

Mnemonic	Description
AS	Print String in ASCII. This mnemonic is not defined in the normal *term*. file. Instead it sets a flag for PRINT. The next PRINT of a string variable will be in ASCII output format. The entire DIMensioned size of the string is sent, including nulls. The internal (non-toggled) information is displayed representing the actual data that would be sent. All codes greater than \200\ are displayed as \xxx\ octal. Printable characters represent themselves, and control characters (001-031) display in ^x format.
BH	Box Horizontal character. This mnemonic is used to draw horizontal box characters using **WINDOW**. If undefined, the '_' character is printed.
BV	' character is printed.
HX	Print String in Hex. This mnemonic is not defined in the normal *term*. file. Instead it sets a flag for PRINT. The next PRINT of a string variable will be in Hex output format. The entire DIMensioned size of the string is sent, including nulls. The internal (non-toggled) information is displayed representing the actual data that would be sent. All codes are represented as hex digits 00 to ff.
OC	Print String in Octal. This mnemonic is not defined in the normal term. file. Instead it sets a flag for PRINT. The next PRINT of a string variable will be in

	Octal output format. The entire DIMensioned size of the string is sent, including nulls. The internal (non-toggled) information is displayed representing the actual data that would be sent. All characters are displayed in \octal\.
RB	Ring BELL. Sends the sequence causing the terminal to beep.
TP	Toggle Page. Switches the display to another page of memory in the terminal.
RD	Read Cursor. The terminal will transmit its current coordinate position to the program. This function is highly dependent upon the terminal.
PI	Position Indicator. This mnemonic is used by supplied utilities to display the requested number of input characters in a field. The form used by the program is usually 'nPInML' where n is the number of characters in the field. The default character for this mnemonic is _.
SA	User Defined mnemonic to contain any non-supported terminal function.
SB	User Defined mnemonic to contain any non-supported terminal function.
SC	User Defined mnemonic to contain any non-supported terminal function.
SD	User Defined mnemonic to contain any non-supported terminal function.
S1	User Defined mnemonic to contain any non-supported terminal function.
S2	User Defined mnemonic to contain any non-supported terminal function.
S3	User Defined mnemonic to contain any non-supported terminal function.
S4	User Defined mnemonic to contain any non-supported terminal function.

Special Mnemonics for I/O Control

Mnemonic	Description
IO	Special lead-in for an IO Control mnemonic. IO is followed by a 2 or 4-character IO mnemonic.
IOBC	Begin activate-on-control-character. The IOBC mnemonic enables XON/XOFF and CTRL+Q/CTRL+S are ignored. The terminating control character is placed into the last position of the INPUT string variable. INPUT continues to terminate on receipt of a control character until the mnemonic 'IOEC' is sent.
IOBD	Begin Destructive Backspace. When destructive backspace is enabled (default), pressing a BACKSPACE or CONTROL-H results in the sequence backspace, space, backspace being transmitted to the screen. Destructive backspace continues until the 'IOED' mnemonic is sent.
IOBE	Begin Input Echo. As characters are entered on the screen, they are displayed (normal default). Input echo continues until the **IOEE** mnemonic is sent. **CALL $ECHO** and the **SYSTEM** Statement provide additional ways to enable/disable echo. Any of the 3 methods can be used together or separately. A **CALL $ECHO** can enable echo disabled by IOEE, etc.
IOBF	Mnemonic accepted, but does not perform a function.

IOBI Begin input transparency. The **IOBI** mnemonic enables Binary Input mode resulting in no input translation of characters received until the **IOEI** is sent. Nulls, [ESC]s, and control characters are placed into the string exactly as received with and without the high-order bit set. When Binary Input is enabled, your INPUT statements must specify a time limit or character count or input continues indefinitely. See also **HALT Command** to unlock a port, and **SYSTEM Statement** Binary Input Mode.

IOBO Begin output transparency. The **IOBO** mnemonic enables Binary Output Mode resulting in no output translation of characters. All 256 ASCII characters are sent to the terminal directly. No graphics or CRT functions are performed. The format of this mnemonic is **IOBO**;"nnnnn\377\". 'nnnnn' is a one to five digit number in the range 1 to 65535 representing the number of characters to output in Binary Output Mode. This field may contain leading spaces or a zero byte. No trailing spaces are allowed. The digit field must terminate with "\377\". If the format is incorrect, Binary Output will not be enabled and the request is ignored. For example, to send the sequence ESCAPE=**, output: "4\377\\233\=**". After the specified number of characters are transmitted, Binary Output Mode is disabled automatically. The PRINT statement terminates strings on zero bytes. To output true zero/null bytes, you may use the **CHR()** function in BITS programming mode. Zero bytes cannot be sent in IRIS mode. See also **SYSTEM Statement** Binary Output mode.

IOBX Begin XON/XOFF protocol. The **IOBX** mnemonic enables Unix sending XON/XOFF protocol when communicating with a Host computer until the **IOEX** mnemonic is sent. The system will prevent overflow of the type-ahead buffer by sending an XOFF to a host when the buffer is full. This function is usually used when you activate a program on a port that is wired directly to another system. Normal keyboard XON/XOFF protocol is always enabled.

IOB Begin sending the \ character to the screen whenever [ESC] is pressed. The default operation is to always send the \ character for IRIS programs, and only for BITS applications without [ESC] branching in effect. The \ will be sent until the **IOE** mnemonic is sent.

IOCI Clear the contents of the terminal's type-ahead buffer. Any input entered but not processed as INPUT is discarded.

IOEC Disable activate-on-control-character. Normal INPUT (default) is restored, and XON/XOFF flow control are terminated. CTRL+Q and CTRL+S are recognized. Input is terminated by [EOL] (usually RETURN), length or time.

IOED End Destructive Backspace. Stop sending backspace, space, backspace. Send only a single backspace and erase the input character from the input buffer.

IOEE End Input Echo. Disable echo of input characters on the terminal. Identical to using **CALL $ECHO** or **SYSTEM** Statement. Input characters are not displayed on the screen until echo is enabled by **CALL $ECHO**, **SYSTEM** or an **IOBE** mnemonic is sent.

IOEF Mnemonic accepted, but does not perform a function.

IOEI End Input Transparency. Normal Input Mode is activated, and Binary Input is disabled. Special characters are processed and [EOL] (usually RETURN)

terminates INPUT.

IOEX End XON/XOFF Protocol. Normal overflow of the type-ahead buffer is allowed. This is the default condition whereby type-ahead buffer overflow outputs a bell to the terminal, and input is discarded.

IOE End sending the \ character to the screen whenever [ESC] is pressed. This function disables the **IOB** mnemonic and system default. The \ character is never sent to the terminal when [ESC]is pressed.

IOIH Setup for special Input Handling. This mnemonic is followed by a byte defining the type of Input processing to be performed. In a future release, custom tables may be defined within the default *term*. file.

IORS Reset the I/O parameters for this terminal. Echo is enabled as is the output of "\" on [ESC]. All other IO modes are turned off.

IRIS Mnemonics Not Supported

Mnemonic	Description
IOIHIR	Set the input handler type to standard processing as defined in your default *term*. file.
IOIHSM	Set the input handler type to SM BASIC standard.
IOIHSR	Set input handler to SM BASIC Read Record format.
IOIHSI	Set input handler to simple format. All characters are input except CONTROL S and CONTROL Q.

$TERM Extended Graphic Mnemonics

To define graphics sequences, you may define the mnemonics **BG EG** to send starting and ending graphics sequences required for the terminal. You then define unused mnemonics, such as the Special Mnemonics (**SA, SB, SC, SD**), color mnemonics, etc. as your own defined graphics sequences. For example, **SA** may be used to draw a left pointing T.

The second (recommended) method involves the definition of special mnemonics for Extended Graphics Characters, or EGC . These are a set of 28 octal characters that, when printed in between **BG** and **EG** mnemonics display graphics characters on the terminal. To enable EGC, define replacement strings for the mnemonics **BG** and **EG** only if your terminal requires a special sequence to switch between normal and graphics modes. It is not necessary to define **BG** or **EG** to use the EGC mnemonics.

Next, define up to 28 different graphics sequences listed below. The format in the term file for defining an EGC is **Gn: replacement** where 'n' is the graphics sequence (1-28), and 'replacement' is the string necessary to create the desired character. The first 11 have pre-defined meanings and should be defined accordingly. They are used for Windows when *replacement strings* for the first six sequences and **BG EG** are defined in the term file.

By defining any **Gn** mnemonics in the *term*. file, you enable EGC and change the method whereby graphic sequences are sent to a screen. No longer are **BG EG** treated as a simple mnemonics. Transmission of the **BG** mnemonic switches translation to the EGC table providing

the 28 possible sequences listed0 in the following table. The **EG** mnemonic resets translation to the standard mnemonic table.

When EGC is enabled, the following \octal\ and mnemonics output graphics sequences. When disabled, the \octal\ or either mnemonic outputs the *replacement string* defined for the first mnemonic below. For example: AE, G3 or \110\ normally enable the auxiliary port. When EGC is defined, these codes output a Lower Left Corner of a box when sent between **BG** and **EG** mnemonics.

Table of Extended Graphics Octal Codes

term	Octal	Mnemonics		Description
G1	\106\	'CT'	G1	Upper left corner
G2	\107\	'ST'	G2	Upper right corner
G3	\110\	'AE'	G3	Lower left corner
G4	\111\	'AD'	G4	Lower right corner
G5	\112\	'SL'	GH	Horizontal bar
G6	\113\	'LU'	GV	Vertical bar
G7	\114\	'SP'	GL	Left pointing T
G8	\115\	'GR'	GR	Right pointing T
G9	\116\	'TB'	GU	Upward pointing T
G10	\117\	'PI'	GD	Downward pointing T
G11	\120\	'RE'	GC	Cross (+)
G12	\121\	'PU'	none	User defined
G13	\122\	'YE'		User defined
G14	\123\	'BL'		User defined
G15	\124\	'MA'		User defined
G16	\125\	'CY'		User defined
G17	\126\	'WH'		User defined
G18	\127\	'XX'		User defined
G19	\130\	'SA'		User defined
G20	\131\	'SB'		User defined
G21	\132\	'SC'		User defined
G22	\133\	'SD'		User defined
G23	\134\	'BV'		User defined
G24	\135\	'BH'		User defined
G25	\136\			User defined

G26	\137\	User defined
G27	\140\	User defined
G28	\141\	User defined

Programs display graphic characters by printing the octal value associated with them. The following example displays a 3 by 3 box:

```
10 PRINT 'BG'"\106\\112\\107\"

20 PRINT "\113\\40\\113\"

30 PRINT "\110\\112\\111\"'EG'
```

Dynamic Windows use EGC to draw boxes using graphics mode. If the first six EGC are not defined, **BV** (if defined) or '|' becomes the vertical character, BH (if defined) or '- ' becomes the horizontal character, and '+' becomes the corner character as defaults. To use EGC with Dynamic Windows, the first six EGC (**G1** through **G6**) must be defined in the *term* file.

BG and **EG** need not be defined to display EGC unless the terminal requires initialization sequences for graphics. If the terminal is an ANSI monitor or one that displays graphic characters without an initialization sequence, **BG EG** need not be defined.

Note: The first eleven EGC (**G1** through **G11**) are reserved and should be used for the features described above. Failure to do so will render Dynamic Windows automatic box drawing useless. Any EGC reserved in the future will start at **G12**. When defining your own characters, start from the end of the list (**G28**) moving backwards.

$TERM Input Character Processing

Characters are processed in the form received from the Unix system. To avoid application problems, normal printable characters should be received in 7-bit form. To verify that your terminal is configured for 7-bit characters, issue the Unix command: **stty -a**. The option *istrip* should be displayed. If the option is displayed as *-istrip*, then the 8th bit is not being stripped prior to passing the data into the application. Refer to the Unix Terminal Information for additional information.

Input Character Definitions are included within the *term* file to define special functions for your applications. These characters are not passed as input unless Binary Input Mode is enabled using **IOBI** or **SYSTEM 14**. Some of these characters will be passed as input if Activate-On-Control-Character mode is enabled by using IOBC. Any given function may have one or more characters invoke its operation, however a single character may not perform two different functions. The format of this information in the term file is:

```
c:action          (printable character c)

^c:action         (control character c)

\ooo\:action      (octal value of character c)
```

In the following table, *action* is a number from 0 to 24 representing a function to be performed upon entry of a character tagged to the *action*. The **CODE** represents the name of the function invoked. Throughout this guide, the **[CODE]** format is used to specify an Input Character Function.

Action	Code	Function
0		Normal data, echo character and process as normal input characters.
1		Convert to space. These characters are to be converted to spaces whenever they are used.
2		Ignore this character completely. Used to disable special keys not supported by an application.
3		Ignore this character and echo a BELL whenever the character is entered by the user.
4	[DBS]	^H: Destructive backspace. Erase one character from the input buffer and echo backspace, space, backspace to erase the character from the screen. If no characters are in the buffer, the terminal bell is sounded.
5	[BS]	DEL: Echoed backspace. Erase one character from the input buffer and echo the erased character.
6	[ESC]	ESC: ESCape. Send the application an ESCAPE. The application can elect to abort, ignore or process the [ESC] itself using IF ERR, ESCSET, ESCDIS or ESCSTM. A \ is sent to the terminal for BITS applications only if [ESC] branching is disabled. For IRIS applications, the \ is always sent unless the mnemonic IOE\ is enabled.
7	[EOBC]	^D: [ESC] override. Abort any running command or program. This character bypasses any program [ESC] handling. A \ is always sent to the terminal.
8	[CANCEL]	^X: Cancel input buffer. Erase all characters currently typed as input characters.
9		^O: Cancel output. Not implemented.
10		^S: Pause output. Temporarily suspend output. This character is set by Unix and cannot be changed.
11		^Q: Resume output. Any output stopped by ^S is resumed. This character is set by Unix and cannot be changed.
12	[ECHO]	^E: Toggle echo. If echo is enabled, disable further echoing until another ^E is entered.
13	[SIGNAL]	^B: Generate signal to your program. This character sends a SIGNAL with two (-1) values.
14	[INTR]	^C: BASIC program interrupt. Used for applications to have a second method of interruption. Requires the use of INTSET statement.
15	[EOL]	RETURN: Terminate input. Transmit any input to the

		program or system as a completed line.
16		Normal data but no echo. This allows the input of certain characters with an echo inhibit. Some characters may, for example, perform an unwanted screen function when entered.
17		Normal data but echo space. This allows the input of certain characters that may affect a terminal to be echoed as space. The cursor will then move reflecting the input of a character.
18		Convert to carriage return. Each input of this character is replaced by a [EOL] (Usually RETURN) in the buffer, however, this character does not terminate input. The default character ^Z performs this operation.
19	[HOT-KEY]	Perform a SWAP to the Executive Program chosen by CALL $SWAPF. When this character is entered at any INPUT statement, the current program is suspended, the Executive is loaded and run until it terminates. See WINDOW and Using Dynamic Windows.
20	[UP]	Cursor Tracking up character. Whenever this character is entered during Cursor Tracking Mode, the character \053\ is returned in the string variable.
21	[DOWN]	Cursor Tracking down character. Whenever this character is entered during Cursor Tracking Mode, the character \052\ is returned in the string variable. This character, if defined, also performs an [EOL] when Cursor Tracking mode is disabled.
22	[LEFT]	Cursor Tracking left character. Whenever this character is entered during Cursor Tracking Mode, the character \010\ is returned in the string variable. This character, if defined, also performs a non-destructive backspace [NDBS] when Cursor Tracking mode is disabled.
23	[RIGHT]	Cursor Tracking right character. Whenever this character is entered during Cursor Tracking Mode, the character \040\ is returned in the string variable.
24	[NDBS]	Non-destructive backspace. Erase one character from the input buffer and echo a backspace. If no input characters are in the buffer, the terminal bell is sounded.

For example, "^A:4" defines the CTRL+A key to perform a destructive backspace. This does not disable the CTRL+H key which is pre-defined to perform the same function. To change one of the default characters, first redefine the existing character, then set the new character for the function. For example: ^H:0 to set CTRL+H as normal data, ^A:4 to set CTRL+A for destructive backspace.

Note: Only one key each may be selected to perform functions 6 and 7. Functions 9, 10 and 11 may not be changed on Unix systems.

Cursor Tracking Mode

Cursor Tracking is an **INPUT** mode available to BASIC programs to monitor the motion of the cursor on the screen. When activated, the arrow keys on the terminal are intercepted and returned as CRT control characters to the BASIC program. The program then tracks the cursor by knowing its position and counting the number of up, down, left and right arrows entered.

To activate Cursor Tracking, output the character \001\ as the last character prior to **INPUT**. Any other character disables Cursor Tracking. Cursor Tracking continues during input until the [EOL] character is entered.

The characters sent by the actual arrow keys must be defined in the *term* file equated to input translation functions 20,21,22 and 23 All terminals behave differently when using the arrow keys past the end of a line or screen. Make sure your application handles the keys the same as the terminal would operate in local mode.

```
10 INPUT {@x,y; and mnemonics} "\001\" str.var

20 PRINT ...  ;"\001\";

30 INPUT "" str.var
```

Statement 10 enables Cursor Tracking for the **INPUT** of A$. Statement 20 enables Cursor Tracking (note the terminating ; preventing transmission of a [EOL] usually RETURN). Statement 30 then receives **INPUT** in Cursor Tracking Mode.

The actual **INPUT** received will be standard ASCII characters. The arrow keys are returned as octal values:

OCTAL	**Key Pressed**
\010\	Left Arrow
\040\	Right Arrow
\052\	Down Arrow
\053\	Up Arrow

Cursor Tracking operates on all data processed following the output of "\001\". Data typed ahead (but not yet received from Unix) will process correctly.

Using Dynamic Windows

A window is simply another page of information on the screen. It may be the entire screen, or a smaller region placed anywhere on the current screen. When a window is created, the underlying information is saved and then cleared. An optional box with heading may be created to highlight a window. When a window is deleted, it is cleared restoring any previous underlying data. See also **WINDOW**.

Each window behaves as a full screen of the dimensions specified. Data automatically wraps within the boundaries of the window and many of the mnemonics are supported. Cursor positioning is relative, such that position 0,0 is the first character of the window. Scrolling within a window is allowed.

The number of windows that may be opened concurrently is limited by the environment variable **WINDOWS** which must be explicitly assigned a value greater than zero (its default) to enable Dynamic Windows.

Window *Zero* is the full screen before any windows are open.

Window Tracking Map is the screen map maintained for the actual screen display present. As windows are created, the underlying information is copied into allocated memory and saved until the window is deleted. The Window Tracking Map holds characters and attribute information for each position on the screen. The map is also used to simulate protected data on ANSI monitors.

Window Tracking is the process whereby each character and its attributes is maintained in a display map. Each character along with attributes for Protect, Reverse, Underline, Dim, Blink and Graphics are maintained. Mnemonics, keyboard echo and cursor addressing are intercepted to prevent access outside the current window. When enabled, a Map is automatically created for Window Zero; the first full-screen window.

If the environment variable **WINDOWS** is not defined or set to zero, a Window Tracking Map is not created and all characters and mnemonics pass directly to the screen. Similarly, if the *crt.type* in the *term*.file is not defined or zero, Window Tracking will be disabled.

By default, when using an ANSI style monitor with *crt_type* in the *term* file set to 23, the environment variable **WINDOWS** defaults to one if undefined or zero. A Clear Screen automatically enables Window Tracking during RUN mode. This allows the monitor to simulate Protected Fields, a normally unsupported feature on most PC monitors.

Using Protected Characters & PC Monitors

Protected characters and fields are simulated whenever Window Tracking is enabled. This is done for the following reasons:

1. Protected fields are not supported on ANSI PC Monitors.
2. Creating a new window on top of already protected characters.
3. Repainting protected characters when restoring underlying data.
4. Limit Clear Unprotect **CU** from clearing characters outside the current window.

On most terminals, protected characters are not truly protected until format mode **FM** is enabled. To accommodate a wide range of terminals, protection is available using {**BP EP**}, {**BP FM EP**} or {**BD ED FM**}. See also $TERM Flags and Switches; *crt_flags*.

To overlay protected fields with a new window, the terminal is never placed into format mode. On most terminals, Format Mode prohibits placing characters over a protected area unless the entire screen is cleared. Window Tracking intercepts **FM**, **FX**, **BP** and **EP** mnemonics to maintain the Window Tracking *Map*. **BD** and **ED** (or **BPW** and **EPW** if defined) are sent to the screen instead of **BP EP** and **FM FX** are never transmitted. **BPW** and **EPW** mnemonic definitions are provided as an alternative to **BD** and **ED**

when using either embedded dimmed mnemonics or alternative attributes.

Because terminals behave differently, and applications have control defining mnemonic *replacement strings*, the following problems may occur:

1. The cursor is moved using a non-supported window mnemonic, or Unix command, corrupting the Window Tracking *Map*.
2. When closing a window, attributes are enabled and disabled as needed to repaint the screen. If the edge of a window overlaps an underlined word, the underline code is sent in the middle of the word in a futile attempt to restore the screen.
3. When a window is created on top of or between beginning and ending codes, the underline runs through the window, up to end of the screen, or disappears altogether.
4. When underlined text wraps from the right edge of a window to the left, underline does not obey a windows borders.

By default, all mnemonics are assumed to occupy zero character positions on the screen with the exception of **BH**, **BV**, **PI**, and *Extended Graphics Characters* which default to a single position.

For problems 1 and 2 above, modify the *term* file for the specific mnemonics inserting the number of characters required on the screen. For example if **BU** (Begin Underline) requires a screen position, modify the entry to read:

BU,1:\33\G8

For problems 3 and 4 above, there is no simple solution. This occurs on terminals requiring a screen position to flag start and end of an attribute. When opening a window thereby clearing the end-attribute, the terminal automatically extends the attribute to the end of line or screen. If the terminal uses an identical sequence to end all special attributes, try lining the left and right sides with these ending mnemonics and modify the window to occupy one less character on each side using **WINDOW MODIFY**.

Mnemonics Simulated During Window Tracking

When a window is opened and Window Tracking is enabled, characters and mnemonic codes are intercepted and forced within the boundaries of the window. If the cursor is moved into a protected region, the cursor is automatically advanced to the first non-protected position. Scrolling is permitted in full screen mode and within a window.

The behavior of Windows varies, depending on the state of the Format Mode (**'FM'**) and the *crt.flags* field in the **term** file. Format Mode controls whether or not protected characters can be overwritten or cleared. When Format Mode is enabled, characters with the protect attribute **('BP')** are immune to being cleared by **'CU'**, **'CE'**, or **'CL'** mnemonics. When Format Mode is disabled **('FX')**, protected characters are vulnerable. Format Mode can be changed either explicitly with **'FM/'FX'**, or implicitly with **'BP'/'EP'** by setting c*rt_flags*:**2**.

There are three additional flags used to describe the behavior of Windows with regards to the **'MD','MR'**, and **'IL'/'DL'** mnemonics on a particular terminal when windows is enabled. They can be modified in the *crt_flags* field of the term file. Window Tracking

needs this information in order to accurately represent the screen. These flags do not control how the mnemonics function, and are only meaningful when Window Tracking is in use. See also: *crt_flags*.

The following pages list the mnemonics supported during Window Tracking. Exercise caution using mnemonics not listed. Verify that the number of screen positions required is defined in the *term* file.

Mnemonic	Function Performed
\215\	Carriage Return. BITS mode sends the cursor to the first column of the window on the current line, whereas IRIS mode also performs an automatic Line-feed (See \212\ New-Line).
\212\	New Line. The cursor is moved to the first column of the open window on the next line.
\210\	Backspace. Backspacing is forced to stay within the boundaries of any open window. Backspacing off the left of the window places the cursor in the last window column of the previous line.
@x,y;	Cursor Addressing operates normally, but is restricted to the boundaries of any open window. If the x or y coordinate is out of range, it is reduced to within range. Cursor positions are origin (0,0) as the first position of the window.
\001\	Cursor Tracking is supported within the boundaries of any open window.
'CR'	Carriage Return. The actual characters defined by the mnemonic are sent and the screen is adjusted to be within the boundaries of the current window. If the mnemonic contains only \15\, then the cursor is moved to the beginning of the current line. If the mnemonic contains \15\\12\ or \15\ in IRIS mode, the carriage is advanced to the first column of the window on the next line.
'LF'	Line Feed. During Window Tracking, the LF mnemonic does not get interpreted from the UniBasic term file and always outputs a line feed.
'MH'	Move Home. The cursor is placed into the first unprotected character position of the window.
'MU'	Move Up. The cursor is moved to the line above in the window unless the cursor is already on the first line of the window , then it scrolls to the next line up.
'MD'	Move Down. The cursor is moved down to the next line in the window unless the cursor is already on the last line of the window, where the action is determined by the configuration, and the *crt_flags*.
'ML'	Move Left. The cursor is moved one position to the left. If the cursor underflows the first column of the window , the cursor is placed on the last character position of the previous line in the window , if any.
'MR'	Move Right. The cursor is moved one position to the right, If the cursor overflows the last column of the window, the action is determined by the configuration, and the *crt_flags*.
'BP'	Begin Protect. The Protect attribute is turned on and all further characters will be tracked with the Protect attribute. If you wish **BP** to turn off format

mode, see *crt_flags* ($TERM Flags and Switches). The CRT code **BD** (Begin Dim) is output to place the screen into half-intensity mode. The alternate sequence **BPW** may be defined to display protected data in other than dimmed intensity.

'EP' End Protect. The Protect attribute is turned off. If you wish **EP** to turn on format mode, see *crt_flags* ($TERM Flags and Switches). The CRT code **ED** (End Dim) is output to restore normal intensity. The alternate sequence **EPW** may be defined to display protected data in other than dimmed intensity.

'FM' Turns on format mode. Formats characters transmitted as protected or dimmed to become protected. Can be used with *crt_flags*:**2** set. See also *crt_flags* ($TERM Flags and Switches).

'FX' Turns off format mode. Can be used with *crt_flags*:**2** set. See also *crt_flags* ($TERM Flags and Switches).

'BU' Begin Underline.

'EU' End Underline.

'BD' Begin Dimmed intensity. By default, dimmed characters are protectable. See also *crt_flags* ($TERM Flags and Switches).

'ED' End Dimmed Intensity. See also *crt_flags* ($TERM Flags and Switches).

'BR' Begin Reverse Video.

'ER' End Reverse Video.

'BB' Begin Blink.

'EB' End Blink.

'BG' Begin Graphics (see $TERM Extended Graphics Characters).

'EG' End Graphics (see $TERM Extended Graphics Characters).

'CU' Clear Unprotected data. All data in the window that is not protected is cleared. The attributes currently enabled are not disturbed by this mnemonic.

'CS' Clear Screen. The entire window is cleared and all attributes are turned off. Note that Clear Screen does not delete the window, rather it performs a Clear Screen within the boundaries of the window.

'XX' Initialize terminal. Same as Clear Screen.

'IC' Insert Character. This action is simulated when the window border does not extend to the right edge of the screen, or a protected character is to the right of the character. The action is dependent on the setting of the *crt_flags*.

'DC' Delete Character. This action is simulated when the window border does not extend to the right edge of the screen, or a protected character is to the right of the character. The action is dependent on the setting of the *crt_flags*.

'IL' Insert Line. An open line is inserted at the current position. This action is simulated when in a window. The action is dependent on the setting of the

crt_flags.

'DL'	Delete Line. Deletes the current line. This action is simulated when in a window. The action is dependent on the setting of the *crt_flags*.
'CL'	Clear Line. All unprotected characters from the current cursor position through the last character of the window line are cleared to space. See also *crt_flags* ($TERM Flags and Switches) to control the operation of **CL** with protected data.
'CE'	Clear End of Screen. All unprotected characters from the current cursor position through the last line of the window are cleared to space.
'BV'	' if undefined. Unused if Extended Graphics are defined. See also $TERM Extended Graphics Characters.
'BH'	Box Horizontal. Default character used for horizontal line around a window. Defaults to '-' if undefined. Unused if Extended Graphics are defined. See also $TERM Extended Graphics Characters.
'PI'	Position Indicator. Used to display a field width for a user INPUT prompt. Normally, PI is '_' (underline), and is used in the form: '20PI' to display for the user a 20-character input field. The cursor is not repositioned to the start of the field.

Note: No other mnemonics are supported during Window Tracking. Others are sent directly to the terminal. If a mnemonic moves the cursor or transmits data, the Windows Tracking Map may be compromised! Define other mnemonic's number of screen positions.

UniBasic Commands

Commands include those functions built within the UniBasic process. Certain other familiar commands (such as **LIBR** or **DIR**) are external system programs and are documented as Utilities.

Commands are issued in either *program mode* or at *command mode*. Commands restricted to *command mode* are signified by the **SCOPEPROMPT '#'**. BASIC *program mode* commands are shown without a prompt. When setting the environment variable **BASICMODE=BITS**, both modes are combined into a single prompt, **BITSPROMPT**, where both types of commands may be issued.

For example:

#**SAVE** {<*attributes*>} *filename* {**!**}

SAVE must be issued from *command mode*.

LIST {*parameters*}

LIST must be issued from BASIC *program mode*.

Some commands are restricted to a single **BASICMODE**. For example, **BITS** uses the **DELETE** command to delete files from command mode, while IRIS uses the **DELETE** command to remove statements.

Note: At the top of each page in this section, if a command is restricted to one mode or the other, it is so indicated. In addition, if a command is restricted to **BITS**, the command format is preceded by the prompt '*'.

For example:

***DELETE** *filename*

Starting & Ending Statement Numbers

Statement Numbers (or labels), are referenced as *stn*, *starting stn*, or *ending stn*. Commands which allow both a *starting stn* and *ending stn* behave differently when selecting **BASICMODE=IRIS** or **BASICMODE=BITS**:

When operating under the environment selected by **BASICMODE=IRIS**, the following rules apply:

1. Supplying a starting *stn* without an *ending stn* selects those statements greater or equal to *starting stn* through the end of the program: `10 LIST`
2. Supplying an *ending stn* without a *starting stn* selects those statements from the beginning of the program up to and including any statement equal to the supplied *ending stn*: `LIST 1000`
3. Supplying both a *starting stn* and *ending stn* selects all statements greater or equal to *starting stn*, and less than or equal to *ending stn*:`1000 LIST 2000`
4. Supplying an identical *starting stn* and *ending stn* selects only that single statement, or the first *stn* greater: `100 LIST 100`

When operating under the environment selected by **BASICMODE=BITS**, the following rules apply:

1. Supplying a *starting stn* followed by a comma and no *ending stn* selects those statements greater than or equal to *starting stn* through the end of the program: `LIST 1000,`
2. Supplying a comma followed by an *ending stn* selects those statements from the beginning of the program up to and including any statement equal to the supplied *ending stn*: `LIST ,1000`
3. Supplying both a *starting stn* and e*nding stn* selects all statements greater than or equal to *starting stn*, and less than or equal to e*nding stn*: `LIST 1000,2000`

4. Supplying a single *starting stn* without a preceding or trailing comma selects only that single statement, or the first *stn* greater: `LIST 1000`

labels may be specified wherever a *stn* is required. Some commands, such as **RENUMBER** or **ENTER**, rely on integer statement numbers for their operation. If a *label* is supplied, its current statement number is substituted.

Processing in Command Mode

When in *command mode*, the following steps are performed for each input line:

1. If the command is internal to UniBasic, it is executed immediately. This includes commands such as **BYE**, **BASIC**, etc., otherwise -
2. A search is made through the directories defined in the environment variable **LUST** for a BASIC program or utility (such as **LIBR**, **QUERY**). If found, the program is started, otherwise -
3. A sub-shell process is started and the command is fed to the shell for execution by Unix. Following execution of the command, *command mode* is resumed.

If an existing internal command (1), or BASIC program (2) conflicts with the name of a Unix command, begin the command line with **!** to force Unix execution (3). In addition, the **!** command reloads the term file and may be used within BASIC *program mode* without sacrificing open files normally closed for IRIS users when *command mode* is entered.

Since commands are performed in a sub-shell child process, changes to environment variables or current working directory are only effective while in the sub-shell. When it terminates, the parent (UniBasic) resumes unaware of the child's activities.

The Unix command **sh** can be issued to enter the shell for a series of operations. The shell remains active until CTRL+D (EOF) or an **exit** command is issued. At that time, *command mode* is reentered.

Because Unix is a multi-processing system, it is possible to have many processes running concurrently on a port. The Unix command **ps** may be used to display all active processes. If this command is executed from *command mode*, you may see a number of shells:

```
#ps

  PID    TTY  TIME  COMMAND

10308     00  0:01  sh

10334     00  0:02  UniBasic

10336     00  0:00

10337     00  0:00  ps
```

The first **sh** is the active shell launched at login. This process will not be displayed if the .profile contains **exec UniBasic** since **exec** replaces the current **sh** with the *UniBasic* process. Next, the *UniBasic* process is running which launched a sub-shell in *command mode* to execute the **ps** command.

When making changes to your current *term* file using a Unix editor (such as **vi**), execute a **!** command from either *command mode* or BASIC *program mode* to force a reloading of the *term* file with your changes.

! Command

SYNOPSIS

Execute External Unix Command

SYNTAX

#! *command*

! *command*

DESCRIPTI0N

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

Upon completion of the *command*, the default *term* file is reloaded, and any opened Windows are cleared.

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

You may use the **!** command to launch another copy of UniBasic (for debug purposes) should you wish to leave the current process intact, i.e. files open, variables undisturbed, etc. Issue the command:

!PORT=xx UniBasic

Where 'xx' is an unused UniBasic *port number*. You then are controlling 2 different port numbers as if you went to another terminal When a **BYE** command or **SYSTEM 0** is executed from the second, the first resumes as if the command was never issued. Care must be exercised with respect to locked records or files, since the second process obeys locks placed on files by the first.

EXAMPLES

```
!ls -l
```

```
!vi /usr/ub/sys/term.tvi925
```

ERRORS

As reported by Unix for specific command

See also

CD Command, Unix Documentation

/ Command (BITS only)

SYNOPSIS

Load and **RUN** a SAVED BASIC program.

SYNTAX

***/** { *filename* }

DESCRIPTI0N

filename is any optional *filename* or full *pathname* to a BASIC program to which you have read-permission. If omitted, the current program (if any) is executed.

The **/** command is only available when operating within the **BASICMODE=BITS** environment.

If **filename** exists as a BASIC saved or system program file, any current program is erased. *filename* is loaded, and execution begins immediately at the lowest *stn* within the program.

This command is identical to a **RUN** *filename* command, available in either IRIS or BITS mode.

EXAMPLES

```
*/

*/payroll

*//usr/ub/programs/pay200

*/sys1:program
```

ERRORS

Filename does not exist

Read Protected File

Not a loadable program file

See also

RUN Command, **LUST** Environment Variable

AUTO

SYNOPSIS

Automatic entry of program statement numbers.

SYNTAX (IRIS)

{*starting stn*} **AUTO** {*increment*}

SYNTAX (BITS)

AUTO {*starting stn*} {,*increment*}

DESCRIPTI0N

starting stn is an optional first statement number in the current program to begin entering new statements. If omitted, 10 is the default. If an existing statement label is supplied, entry begins at that statement number.

increment is the optional statement number increment for automatic entry. If omitted, 10 is the default. If a label is supplied, its statement number is used as the increment. It is suggested that only a number be supplied as the increment.

AUTO displays the *stn* allowing entry of the new statement. If the *stn* 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 and the same *stn* is requested.

AUTO is terminated by pressing **ESC**.

EXAMPLES

```
AUTO 1

AUTO 100,1
```

ERRORS

Various syntax and encoding ERRORS

See also

Program Statements

BASIC (IRIS only)

SYNOPSIS

Load a new **BASIC** programand/or switch to program mode.

SYNTAX

#BASIC {*filename*}

DESCRIPTI0N

filename is any *filename* or *pathname* to a BASIC program or saved System BASIC program to which you have read-permission.

If *filename* is a saved BASIC program file or System BASIC program file, any current program is cleared and the partition is loaded with the new program. If the partition contains a needed program, it should first be saved or dumped.

The error 'Not a Loadable Program File' may occur if the program has been encrypted using the **PSAVE** command. These programs are not accessible unless your system has an authorized **OSN** (OEM Selection Number) installed by the owner.

If the new program was saved with variables (**VSAVE** or **CHAIN "SAVE ..."**, the message 'with variables' is displayed.

The Supplemental Protection Attribute **F** flags an IRIS program file. If the program was saved with the attribute **E** (Execute only), the program is automatically erased from memory after loading.

BASIC is only available when operating in the environment **BASICMODE=IRIS**. Use the **GET** command when **BASICMODE=BITS**.

EXAMPLES

```
#BASIC 23/FILENAME

#BASIC /usr/ub/sys/libr
```

ERRORS

Filename does not exist

Not a loadable program file - wrong revision, protected or corrupted

Read Protected File

See also

GET, MERGE, LOAD, Filenames and Pathnames, **OEM** Command, Supplemental Protection Attributes, **RSAVE, PSAVE, VSAVE, CHAIN "SAVE ...", LUST**

BAUD

SYNOPSIS

Change terminal's IO parameters.

SYNTAX

#BAUD *rate*

DESCRIPTI0N

Baud is the rate of transmission, in bits-per-second on a serial line. Most I/O controllers use the RS-232 serial standard to interface with terminal devices. In this case, rate generally equals the number of bits per second. Since a frame of data is usually 10 bits (start bit, 8 data bits, stop bit), the actual transmission speed in characters per second is calculated by dividing the *rate* by 10.

To change your port’s baud rate, the following conditions should be observed:

1. Your terminal device must be speed-selectable.
2. Your terminal must be connected to a software speed-selectable controller board. Some boards are speed-selectable via switches on the board itself, older boards may not be selectable at all.

3. The baud rate selected must be legal both for your terminal and the port controller board. Most programmable controllers allow the rates: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 and 38400.

If your terminal's *rate* cannot be changed, an error message is returned, and the command is ignored.

BAUD uses the Unix **stty** command to change the rate.

EXAMPLES

```
#BAUD 2400

*BAUD 19200
```

ERRORS

Illegal speed

See also

Unix **stty** command

BYE

SYNOPSIS

Terminate UniBasic session.

SYNTAX

#BYE

DESCRIPTI0N

The **BYE** command clears any program from the user's partition, closes all channels, deletes any remaining signals, removes the message queue associated with the port and terminates the current session.

The current day and time, Unix [group-user] and current *port number* are displayed:

```
09 JUL 1986 13:10:47 [7-4] Port=15

CPU=10, Connect=15, Disk=13234
```

The second line contains cpu and connect time usage for the session just ended, followed by the number of available disk blocks on the file system.

The parent process resumes when UniBasic terminates. If UniBasic was launched from the shell (or .profile) using the command **exec UniBasic**, the terminal is signed off, and the login prompt is displayed. Otherwise, the shell or calling process is resumed.

If you are in debug mode following a non-trapped error, **[ESC]**, or **[EOBC]** in a child UniBasic process launched using the **[Hot-Key]** or **SWAP** statement, the parent UniBasic is resumed.

EXAMPLES

```
#BYE

*BYE
```

ERRORS

None

See also

SYSTEM 0

CD

SYNOPSIS

Change current working Directory.

SYNTAX

#CD { *pathname* }

DESCRIPTI0N

pathname is any *logical unit*, *pack name*, *directory name* or full Unix *pathname*.

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

If a *logical unit*, *packname* or *directory name* is specified, the Logical Unit Search Table **LUST** is searched for the first full *pathname* where the directory is below. The current working directory is changed to the *new pathname*.

This command is not totally compatible to the Unix **cd** command. The Unix environment variable **CDPATH** is not searched. The command is provided for convenience since direct execution of the Unix **cd** command is performed in a sub-shell, and changes do not affect the current process.

EXAMPLES

```
#CD 23

#CD /usr/ub/text

#CD data:
```

ERRORS

System Error - No such file or directory

See also

PACK, *Filenames* and *Pathnames*, **UNIT**, **LUST**, **CLU**

CHAIN "SAVE. . ." (IRIS only)

SYNOPSIS

Save the current program with variables.

SYNTAX

CHAIN "SAVE *filename*{**!**}**"**

DESCRIPTI0N

The **CHAIN** statement is executed in *immediate mode* as a long chain to the **SAVE** command providing for the saving of variables and program state for later debugging.

The optional **!** provides for replacement of an existing *filename*.

filename is any legal *filename* or *pathname* to contain the saved version of the current program.

All variables, **GOSUB** stack, **FOR/NEXT** stack, User Defined Function stack are saved. A prompt 'with variables' is displayed during the **SAVE** as well as during later loading of the program using **BASIC** or **GET**.

CHAIN "SAVE ..." is used to save a copy of a program for later debugging. Any open file information is not saved. Applications may use a combination of error-branching (**ERRSET**, **ERRSTM**, or **IF ERR**) and **CHAIN "SAVE ..."** to facilitate later debugging of an application failure.

EXAMPLES

```
CHAIN "SAVE ERRORS/"+MSF(4)

CHAIN "SAVE PROGRAMERROR!"
```

ERRORS

Filename already exists; use "!" to replace

No program in partition

See also

VSAVE, CHAIN, LUST

CHANGE (BITS only)

SYNOPSIS

Change filename or attributes.

SYNTAX

***CHANGE** *filename* (*newfilename*{**!**} | *<new attributes>*)

DESCRIPTI0N

filename is any *filename* or *pathname* to a file with write permission. The optional **!** provides for the replacement of an existing *filename*.

newfilename is any new *filename* or *pathname* if a name change is desired.

new attributes is any IRIS, BITS, Unix or Supplemental Attributes if the file permissions are to be changed.

The **CHANGE** command is only available when operating in the environment **BASICMODE=BITS**. To change *filenames* or *attributes* in an **IRIS** environment, see the **CHANGE** utility, or **MODIFY** statement.

CHANGE uses the Unix **mv** command to rename a file, and the **chmod** command to change *attributes*. Supplemental Attributes are stored in files unique to UniBasic within each file's header.

If the file is an Indexed Data File, both the data and ISAM portion are changed.

CHANGE may also be used to move a file from one directory to another. If a *Logical Unit*, *Packname*, or *pathname* is specified for *filename*, you must include the same for the *newfilename* or the file is moved to your current working directory.

EXAMPLES

```
*CHANGE PACK:FILENAME PACK:NEWFILENAME

*CHANGE PACK:FILENAME <EO666>
```

ERRORS

Filename does not exist

See also

CHANGE Utility, **MODIFY** Statement, Filenames and Pathnames, File Attributes, Protection and Permissions, **LUST**

CHECK (IRIS only)

SYNOPSIS

Scan program for proper blocked-IF's.

SYNTAX

CHECK

DESCRIPTI0N

The current program is checked for proper Blocked-**IF** structure. If any ERRORS are detected, an error is printed.

Blocked IF statements are also checked whenever a program is **SAVE**d.

CHECK is only available when operating in the environment **BASICMODE = IRIS**. For **BITS** environments, use the **VERIFY** command.

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

EXAMPLES

```
CHECK
```

ERRORS

IF without ENDIF

ENDIF without IF

ELSE without ENDIF

See also

VERIFY, SAVE, IF

CLU (IRIS only)

SYNOPSIS

Change current working Directory.

SYNTAX

#CLU { *pathname* }

DESCRIPTI0N

pathname is any *logical unit*, *packname*, *directory name* or full Unix *pathname*. If no *pathname* is specified, the current default working *pathname* is displayed.

If a *logical unit*, or *packname* or *directory name* is specified, the Logical Unit Search Table **LUST** is searched for the first full *pathname* where the directory is below. The current working directory is changed to the new *pathname*.

This command is not totally compatible to the Unix **cd** command. The Unix environment variable **CDPATH** is not searched. The command is provided for convenience since direct execution of the Unix **cd** command is performed in a sub-shell, and changes do not affect the current process.

EXAMPLES

```
#CLU 23

#CLU /usr/ub/text
```

ERRORS

System Error - No such file or directory

See also

PACK, Filenames and Pathnames, **UNIT**, **LUST**, **CD** Command

CONTINUE

SYNOPSIS

Resume execution of stopped program.

SYNTAX

CONTIN{UE}

DESCRIPTI0N

CONTINUE resumes execution of a program stopped by Breakpoint, **STOP**, non-trapped error, or **[EOBC]** (usually CTRL+D).

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

When operating in the environment **BASICMODE=IRIS**, *command mode* automatically closes all open channels. To perform shell or other system commands, use the **!** command to invoke a shell or another copy of UniBasic.

EXAMPLES

```
CONTINUE

CONTIN
```

ERRORS

none

See also

STOP, **END**, BASICMODE, Program Breakpoints, Single Step Execution, **TRACE**, **SYSTEM 20/21/22/23**

DEL (BITS only)

SYNOPSIS

Delete a file.

SYNTAX

***DEL** *filename ...*

DESCRIPTI0N

filename is any legal *filename* or *pathname* to a file to which you have write permission.

The **DEL**ete command may be used to delete one or more files from disk.

Multiple *filenames* separated by spaces may be included on the

command line up to the length of the input buffer.

If, for any reason, a file cannot be deleted, a message to this effect is output and deletion of files stops immediately. **DEL** does NOT indicate which file caused the error. It is best to delete multiple files only when you are sure all can be deleted, or use the **KILL** utility.

DEL is only available when operating in the environment **BASICMODE=BITS**. For **IRIS** environments, use the **KILL** utility.

EXAMPLES

```
*DEL PACK:FILENAME

*DEL /usr/ub/sys/term.old
```

ERRORS

File does not exist

File is read-protected

See also

KILL, MFDEL, KILL, LUST

DELETE (IRIS only)

SYNOPSIS

Delete program statements.

SYNTAX

{*starting stn*} **DELETE** {*ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to delete. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is used.

ending stn is an optional last *stn* in the current program to delete. If omitted, the highest statement number is selected. If the *ending stn* does not exist, the first existing lower *stn* is used.

DELETE without a *starting stn* or *ending stn* removes all statements in the current program. It is not the same as a **NEW** command which also clears variable names and values.

When operating in the environment **BASICMODE=BITS**, use the **ERASE** command to remove statements.

EXAMPLES

```
INPUT: DELETE END_INPUT:

9900 DELETE

100 DELETE 200
```

ERRORS

none

See also

ERASE, Program Statements, Starting & Ending Statement Numbers

DUMP

SYNOPSIS

Decode a program to a file, device or pipe.

SYNTAX (IRIS)

{*starting stn* } **DUMP** {*opt*} {*<attr>*} *filename*{**!**} {/*text*/} {*ending stn*}

SYNTAX (BITS)

DUMP {*opt*} {{*<attr>*} *filename*{**!**}} {/*text*/}{*starting stn*}{*,ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to decode. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is used.

opt are optional parameters to control the display. Each parameter is a single letter preceded by / or - :

- **Tn** Set the tab stop to column 'n'. Statements are tabbed to column 'n' for easier readability. This option is most useful when statement labels are used instead of standard statement numbers.
- **L** Substitute labels for statement numbers in **GOTO**, etc. wherever possible.
- **N** Do not list statement numbers.

<attr> are any optional valid file *attributes*, *protections*, or *permissions* to apply to the file on creation. Since the file is created as a standard Unix Text file, Supplemental Protection Attributes are not permitted. Standard IRIS, BITS, or Unix permissions may be supplied. If omitted, file creation is defaulted to permissions 0666 (Read/Write by all users) subject to any **umask** in effect.

filename is any *filename* or *pathname* to a directory 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 Unix Text File compatible with standard editors such as **vi**. *filename* may also be a pipe by beginning the *filename* with a **$**.

/*text*/ is any optional string to search each statement for. 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. Note that all *text* is case dependent. Statements, variables, etc. must be searched for using uppercase as shown during program listings.

ending stn is an optional last *stn* in the current program to decode. If omitted, the highest statement number is selected. If the *ending stn* does not exist, the first existing lower *stn* is used.

When using **BASICMODE=IRIS**, the first format is used. **BITS** requires entry using the second form.

EXAMPLES

```
*DUMP -T5 FILENAME /WRITE #/ 100,200

100 DUMP -L FILENAME! 200

INPUT: DUMP -T5 FILENAME! /INPUT/ END_INPUT:
```

ERRORS

Write Protected File

Illegal Filename

Filename already exists; use ! to replace

See also

LIST, **FIND**, Pipes, Filenames and Pathnames, Starting & Ending Statement Numbers

EDIT

SYNOPSIS

Edit and change an existing statement.

SYNTAX

EDIT *stn*

EDIT .

DESCRIPTI0N

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

EDIT . displays the last command or line entered for editing. This is helpful when an error is made during program or command entry.

After an **EDIT** command is issued, the statement is displayed and the cursor is placed in the first position. Typing **H** displays a help screen.

EDIT commands:	(n = optional repetition count, default = 1)
D	Displays the current line and repositions to the

	beginning.
nSpace	Moves n positions to the right, echoing characters.
nBackspace	Moves n positions to the left, echoing characters.
I<text>	Inserts <text> at the current position until ESCape is pressed.
A<text>	Appends <text> to the end of the line until ESCape is pressed.
nX	Deletes n characters to the right of the cursor.
nS<text>	Substitutes n chars to the right with <text> until ESCape is pressed.
R<char>	Replaces the current character with <char>.
n/<text>	Searches forward for the nth occurrence of <text>.
n?<text>	Searches backward for the nth occurrence of <text>.
N	Search Next. Repeats last / or ? command.
U	Undo all editing and reload edit buffer with original contents.
Q	Exit the edit mode, ignoring any changes.
<Return>	Exit edit mode and encode the resultant line.

While in an insert type command (I, A), data is not re-displayed until ESC is pressed. During delete, the screen is updated as each character is deleted.

EXAMPLES

```
EDIT 10
```

ERRORS

No such statement number

ERASE (BITS only)

SYNOPSIS

Delete program statements in BITS mode.

SYNTAX

***ERASE** {*starting stn*} {,*ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to erase. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is used.

ending stn is an optional last *stn* in the current program to erase. If omitted, the highest statement number is selected. If the *ending stn*

does not exist, the first existing lower *stn* is used.

ERASE with a single *stn* will delete only that statement.

ERASE without a *starting stn* or *ending stn* deletes all statements in the current program loaded in memory. It is not the same as a **NEW** command which also clears variable names and values.

When operating in the environment **BASICMODE=IRIS**, use the **DELETE** command to remove statements.

EXAMPLES

```
*ERASE 10

*ERASE ,100  !From beginning to 100 inclusive

*ERASE 1000, !From 1000 to end of program

*ERASE INPUT:, END_INPUT:
```

ERRORS

none

See also

DELETE, Program Statements, Starting & Ending Statement Numbers

EXEC (IRIS only)

SYNOPSIS

Execute contents of a text file.

SYNTAX

#EXEC *filename*

DESCRIPTI0N

filename is any legal **filename** or **pathname** to a Text File to which you have read permission.

Standard Input is switched to the text file performing all commands within the file until EOF.

EXEC is an internal command. To perform a Unix **exec** command, the **!exec** form must be used.

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.

The **DIR** and **MAKECMND** utilities may be used to construct a series of commands on all (or selective) files in a directory.

When operating in the environment **BASICMODE=BITS**, use the **GET** command to execute a text file.

EXAMPLES

```
#EXEC filename
```

ERRORS

File does not exist

Read Protected file

See also

DIR, GET, MAKECMND, Filenames and Pathnames, **LUST, LOAD**

EXIT (IRIS only)

SYNOPSIS

Exit program mode to command mode.

SYNTAX

EXIT

DESCRIPTI0N

EXIT is used to terminate BASIC *program mode* and enter into *command mode*. **EXIT** is identical to pressing CTRL+C. However, it may be included in a text file executed using the **EXEC** command.

EXAMPLES

```
EXIT
```

ERRORS

none

See also

[INTR], CTRL+C

FILE

SYNOPSIS

Display current program and all open files.

SYNTAX

FILE

DESCRIPTI0N

FILE displays the name of the current BASIC program loaded into the partition. If the program was entered via a **[HOT-Key]** or **SWAP** statement, the calling program and stn are also displayed:

```
Program: ar.custmaint
```

```
SWAP at statement:    110;1 in: op.openorder1

SWAP at statement:     20;1 in: sp.selector
```

For all data files opened, the *channel number* and full *pathname* are displayed:

```
CHANNEL#    FILENAME OPENED

0           /usr/ub/sys/lpt

1           /usr/ub/3/ar.customers

2           /u/5/orderheader
```

Note: When operating in the environment **BASICMODE=IRIS**, all channels are closed whenever **END**, **CHAIN ""**, or *command mode* # is entered.

EXAMPLES

```
FILE
```

ERRORS

none

See also

Filenames and Pathnames, **CHF, CHN**

(Filename)

SYNOPSIS

Load and RUN a SAVED BASIC program.

SYNTAX

#filename

DESCRIPTI0N

filename is any *filename* or full *pathname* to a BASIC program to which you have read-permission.

If *filename* exists as a BASIC saved or system program file, any current program is erased. *filename* is loaded, and execution begins immediately at the lowest stn within the program.

This command is identical to a **RUN** *filename* command.

EXAMPLES

```
#payroll

#/usr/ub/23/payroll
```

ERRORS

Filename does not exist

Read Protected File

Not a loadable program file

See also

/ Command, **RUN** Command, LUST

FIND

SYNOPSIS

Search & list statements.

SYNTAX (IRIS)

{*starting stn*} **FIND** {*opt*} /*text*/ {*ending stn*}

SYNTAX (BITS)

FIND {*opt*} /*text*/ {*starting stn*} {, *ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to search and decode. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is used.

opt are optional parameters to control the display. Each parameter is a single letter preceded by / or - :

- V Visual mode. The first screen full of lines are displayed. If additional lines are included, the user is prompted at the bottom of the screen with **[MORE]**. Pressing **[RETURN]** displays additional program lines one at a time. Pressing **[SPACE]** displays the next screen full of lines starting with the last line of the previous screen. This process is repeated until no more lines are found, or **[ESC]** or **[EOBC]** (usually CTRL+D) is pressed.
- Tn Set the tab stop to column 'n'. Statements are tabbed to column 'n' for easier readability. This option is most useful when statement labels are used instead of standard statement numbers.
- L Substitute labels for statement numbers in **GOTO**, etc. wherever possible.
- N Do not list statement numbers.

/*text*/ is any optional string to search each statement for. 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. Note that all *text* is case dependent. Statements, variables, etc must be searched for using uppercase as shown during program

listings.

ending stn is an optional last *stn* in the current program to search and decode. If omitted, the highest statement number is selected. If the *ending stn* does not exist, the first existing lower *stn* is used.

To perform a **FIND** and re-direct output to a *file*, *device*, or *pipe*, use the **DUMP** command.

When using **BASICMODE=IRIS**, the first format is used. **BITS** requires entry using the second form.

EXAMPLES

```
*FIND -V /OPEN #/

100 FIND -V /V$=/ 500

100 FIND -V /CHAIN WRITE/ INPUT:
```

ERRORS

none

See also

LIST, **DUMP**, Starting & Ending Statement Numbers

GET (BITS only)

SYNOPSIS

Load a text or saved BASIC program.

SYNTAX

*__GET__ *filename*

*__GETI__ *filename*

*__GETB__ *filename*

DESCRIPTI0N

filename is any *filename* or *pathname* to a text file, BASIC program or saved System BASIC program.

If the supplied *filename* exists as a saved BASIC program file, System BASIC program file or Text File, any current program is cleared and the partition is loaded with the new program. If the partition contains a needed program, it should be saved or dumped (using the **SAVE** or **DUMP** commands) first.

All saved BASIC programs retain a current checksum of the entire program file. The error 'Not a Loadable Program File' may occur if the program has been encrypted by the owner using the **PSAVE** command. These programs are not accessible unless your system has an authorized **OSN** (OEM Selection Number) installed by the owner.

If a saved program is loaded successfully, it’s checksum and

compatibility are output in the form:

```
*GET ABC Check = AF23 *** BITS PROGRAM ***

*GET ABC1 Check = D680 *** IRIS PROGRAM ***
```

If the new program was saved with variables (using **VSAVE** or a long **CHAIN** to **SAVE**, the message 'with variables' is printed.

The Supplemental Protection Attribute **F** is used to indicate an IRIS program file. If the program was saved with the attribute **E** (Execute only), the program is automatically erased from memory after loading.

The **GET** command is only available when using **BASICMODE=BITS**. To load a BASIC program with **BASICMODE=IRIS**, use the **BASIC**, or **LOAD** command.

If *filename* is a Text File, it is loaded using the rules invoked by the **GET** command chosen:

GETI is used to force the loading of a text file using the encoding rules of IRIS.

GETB is used to force the loading of a text file using the encoding rules of BITS.

EXAMPLES

```
*GET 23/filename
```

ERRORS

Filename does not exist

Not a loadable program file - wrong revision, protected or corrupted

See also

BASIC, MERGE, LOAD, Filenames and Pathnames, **OEM**, Supplemental Protection Attributes, **RSAVE, PSAVE, VSAVE, LUST**

GO (IRIS only)

SYNOPSIS

Resume execution of stopped program.

SYNTAX

GO

DESCRIPTI0N

GO resumes execution of a program stopped by Breakpoint, **STOP**, non-trapped error, or **[EOBC]** (usually CTRL+D).

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

Note: When operating in the environment **BASICMODE=IRIS**, entry into *command mode* automatically closes all open channels. To perform shell or other system commands, use the **!** command to invoke a shell or another copy of UniBasic.

EXAMPLES

```
GO
```

ERRORS

none

See also:

CONTINUE, STOP, END, BASICMODE, Program Breakpoints, Single Step Execution, **TRACE, SYSTEM 20/21/22/23**

HALT

SYNOPSIS

Terminate BASIC program on another port.

SYNTAX

#**HALT** *port number*

DESCRIPTI0N

port number is any integer in the range 0 to the upper limit defined by the environment variable **MAXPORT**.

The *message queues* are searched for the process running as the selected *port number*. If found, the program running on that *port number* is terminated unconditionally into BASIC *program mode* with channels left open.

If no program is running, the **HALT** command is ignored.

HALT may be used when an application has disabled **[ESC]** using **IF ERR, ESCSET, ESCSTM** or **ESCDIS**, and no input translation character is defined for **[EOBC]** (usually CTRL+D). It is also useful to terminate a running program started by **SPAWN**, **PORT** or **CALL $TRXCO** on a phantom port.

EXAMPLES

```
#HALT 25

*HALT 25
```

ERRORS

none

See also:

Port Numbering and Phantom Ports

HELP

SYNOPSIS

Print text DESCRIPTI0N of an error.

SYNTAX

HELP {*error number*}

DESCRIPTI0N

error number is any optional positive integer representing a BASIC error number, or negative integer representing a Unix system error as returned by the **ERR(0)** or **SPC(8)** functions.

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.

When running an IRIS program, *error number* is assumed to be an IRIS error number as returned by **SPC(8)**.

When running a BITS program, *error number* is assumed to be a UniBasic or BITS error number.

The environment variable **BASICMODE** does not determine the interpretation of *error number*. Instead, the current program type, BITS or IRIS, determines the error text returned. Negative (system) ERRORS are identical for either type of program.

If you are unsure as to the type of program loaded in the partition, you may issue a **VERIFY** command.

EXAMPLES

```
HELP

HELP 23

*HELP 9
```

ERRORS

No such error number

See also:

ERR, SPC, VERIFY, Appendix C, Error Message file /usr/ub/errmessage

LEVEL

SYNOPSIS

Print UniBasic revision data.

SYNTAX

#LEVEL

DESCRIPTI0N

The **LEVEL** command prints the current UniBasic revision number, PASSPORT revision level, and the UniBasic license number.

EXAMPLES

```
#LEVEL: UniBasic 8.1, PASSPORT daemon 4.1

        Your license# is 9C4D3168
```

ERRORS

none

See also:

UniBasic Security

LIST

SYNOPSIS

Decode BASIC statements.

SYNTAX (IRIS)

{*starting stn*} **LIST** {*switches*} {/*text*/} {*ending stn*}

SYNTAX (BITS)

LIST {*switches*} /*text*/ {*starting stn*} {, *ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to decode. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is assumed.

switches are optional parameters to control the display. Each parameter is a single letter preceded by a / or - :

V Visual mode. The first screen full of lines are displayed. If additional lines are included, the user is prompted at the bottom of the screen with **[MORE]**. Pressing **[RETURN]** displays additional program lines one at a time. Pressing **[SPACE]** displays the next screen full of lines starting with the last line of the previous screen. This process is repeated until no more lines are found, or **[ESC]** or **[EOBC]** (usually CTRL+D) is pressed.

Tn Set the tab stop to column 'n'. Statements are tabbed to column 'n' for easier readability. This option is most useful when statement labels are used instead of standard statement

numbers.

L Substitute labels for statement numbers in **GOTO**, etc. wherever possible.

N Do not list statement numbers.

/*text*/ is any optional string to search each statement for. 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. Note that all *text* is case dependent. Statements, variables, etc must be searched for using uppercase as shown during program listings.

ending stn is an optional last *stn* in the current program to decode. If omitted, the highest statement number is selected. If the *ending stn* does not exist, the first existing lower *stn* is assumed.

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

EXAMPLES

```
LIST -V

*LIST -V /WRITE #0/ START:, INPUT:

100 LIST -V 500

INPUT: LIST END_INPUT:
```

ERRORS

none

See also:

FIND, DUMP, Starting & Ending Statement Numbers

LOAD (IRIS only)

SYNOPSIS

Load BASIC statements from a text file.

SYNTAX

LOAD {*filename*} {*-filename*}

DESCRIPTI0N

filename is any Text File to which you have read-permission. The file must contain BASIC program statements generated from a **DUMP** command or editing program.

-filename strips comments from the text of a BASIC program.

As each line of text is loaded, it is added to the current program in your partition. The statements in the text file need not be in any

particular order. If any statement already exists, it is replaced. For example, assume the following program is currently in your partition:

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

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

```
26 IF A=30 THEN END
30 GOTO 100
```

The resultant program would be:

```
10 LET A=A+1
20 LET B=SQR(A)
26 IF A=30 THEN END
30 GOTO 100
```

EXAMPLES

```
LOAD sys/program
```

ERRORS

Filename does not exist

Read Protected File

See also:

GET, BASIC, MERGE, Filenames and Pathnames, **LUST**

MERGE (BITS only)

SYNOPSIS

Merge statements from a text file.

SYNTAX

*__MERGE__ *filename*

DESCRIPTI0N

filename is any Text File to which you have read-permission. The file must contain BASIC program statements generated from a **DUMP** command or editing program.

MERGE is similar to **GET** except that the user's partition is not cleared first. When operating in IRIS mode, the **LOAD** command is used to merge statements.

As each line of program text is merged, it is added to the current program in your partition. The statements in the text file need not be in any particular order. If any statement already exists, it is replaced. For example, assume the following program is currently in your partition:

```
10 LET A=A+1
```

```
20 LET B=SQR(A)
```

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

```
26 IF A=30 THEN END
30 GOTO 100
```

The resultant program would be:

```
10 LET A=A+1
20 LET B=SQR(A)
26 IF A=30 THEN END
30 GOTO 100
```

EXAMPLES

```
*MERGE 23/FILENAME
```

ERRORS

Filename does not exist

See also

GET, LOAD, Filenames and Pathnames, **LUST**

MSG

SYNOPSIS

Transmit a message to another port.

SYNTAX

#**MSG** (*port number* | @) ; *text*

DESCRIPTI0N

port number is any integer from zero to the value defined by the environment variable **MAXPORT** (usually 999). An @ may be used by the root account to transmit a message to all active UniBasic processes.

text defines the string of characters to transmit to the specified **port number** or @ all ports.

If a selected **port number** is running a UniBasic process, the *text* is transmitted immediately, regardless of the receiving port's status. The output is duplicated on the sender's own port. Messages may not be transmitted to processes other than UniBasic. It is preceded by an identification of the sender in the form:

```
[g-u] Port=p/message
```

where [g-u] is the group and user and 'p' is the port number of the sender

CTRL+Z characters may be embedded within the message string and

will be transmitted as a carriage return and line feed.

To transmit a message to any Unix user, issue one of the following Unix commands:

```
mail name message [return] [CTRL+D]

write name message [return] [CTRL+D]
```

EXAMPLES

```
#MSG @; System is going down in 10 minutes

#MSG 0; Please mount Tape #12/23/88

*MSG 7; Please call me for lunch at 12:30
```

ERRORS

Port 'n' is not logged on

See also

Unix mail command, Unix write command

NEW

SYNOPSIS

Clear partition for a new program.

SYNTAX

NEW

NEWI

NEWB

DESCRIPTI0N

The **NEW** command clears your partition of any current program. By default, the operational mode selected by the environment variable **BASICMODE** specifies the type of program to be created.

NEW clears the partition and closes any open channels.

All memory allocated is released, and reallocated.

NEWI selects IRIS program syntax for the creation of a new program.

NEWB selects BITS program syntax for the creation of a new program.

In any case, the programming mode is output, e.g.:

```
NEW   *** IRIS PROGRAM ***

NEWB   *** BITS PROGRAM ***

NEWI   *** IRIS PROGRAM ***
```

EXAMPLES

```
NEW

NEWI

NEWB
```

ERRORS

none

See also

BASICMODE

OEM

SYNOPSIS

Display list of authorized dealer software.

SYNTAX

#OEM

DESCRIPTI0N

The **OEM** command is issued at *command mode*. A list of all active OEM protections enabled is printed. "M" is printed to indicate the presence of a Master OSN. The list is numbered when more than one OEM package is installed. This number corresponds to the **PSAVE** command.

Authorized software:

```
    DESCRIPTION

1
M   Customized Accounting Package
```

This display indicates that an **OSN** (OEM Security Number) is installed allowing the operation of the Customized Accounting Package. In addition, the Master OSN is installed allowing access to program source.

EXAMPLES

```
#OEM
```

ERRORS

none

See also

PSAVE

PACK (BITS only)

SYNOPSIS

Change current working Directory.

SYNTAX

*__PACK__ { *pathname* }

DESCRIPTI0N

pathname is any *logical unit*, *packname*,. *directory name* or full Unix *pathname*. If no *pathname* is specified, the current default working *pathname* is displayed.

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

If a *logical unit*, or *packname* or *directory name* is specified, the Logical Unit Search Table *LUST* is searched for the first full *pathname* where the directory is below. The current working directory is changed to the new **pathname**.

This command is not totally compatible to the Unix **cd** command. The Unix environment variable **CDPATH** is not searched. The command is provided for convenience since direct execution of the Unix **cd** command is performed in a sub-shell, and changes do not affect the current process.

PACK is only available when operating in the environment **BASICMODE=BITS**. For **IRIS** environments, use the **CD** or **CLU** commands.

EXAMPLES

```
*PACK 23
```

```
*PACK /usr/ub/text
```

ERRORS

System Error - No such file or directory

See also

CD, CLU, Filenames and Pathnames, **UNIT, LUST**

PROTECT

SYNOPSIS

Protect individual BASIC statements.

SYNTAX (IRIS)

{*starting stn*} **PROTECT** {*ending stn*}

SYNTAX (BITS)

PROTECT {*starting stn*} {, *ending stn*}

DESCRIPTI0N

starting stn is an optional first statement number in the current

program to protect. If omitted, the first statement number is selected. If the starting *stn* does not exist, the first existing higher *stn* is assumed.

ending stn is an optional last statement number in the current program to protect. If omitted, the highest statement number in the program is selected. If the *ending stn* does not exist, the first existing lower *stn* is assumed.

All program statement numbers inclusive are protected from being decoded. This applies to the commands: **FIND**, **LIST** and **DUMP**.

Protected program lines are output as a stn only by the **LIST** and **DUMP** commands. This is done as a reminder that the lines exist, but are protected.

Once protected, there is no unprotect ability; the lines must be re-entered. It is recommended that an original source copy of a program be kept somewhere for later reference, if necessary.

Any attempt to load an ASCII program with protected lines will produce an error. One must use the original copy without the protected lines in order to **DUMP** and **GET** or **LOAD** a program.

EXAMPLES

```
100 PROTECT 999</file>

*PROTECT 1000,1100</file>

READ_FILE: PROTECT END_READ_FILE:</file>
```

ERRORS

none

See also

PSAVE, Starting & Ending Statement Numbers

PSAVE

SYNOPSIS

Protect & SAVE the current program.

SYNTAX

#**PSAVE** {**-ro**} {**OSN#**}, {{<*attributes*>} {*filename*{**!**}}}

DESCRIPTI0N

OSN# is the number displayed by the **OEM** command. It selects the application group and encryption algorithm to employ. If omitted, the first group is selected. The **OSN#** may be omitted if only one encryption algorithm is installed on the system.

<*attributes*> are the desired protection, permission or attributes to apply to the newly created *filename*. If <*attributes*> are supplied, a *filename* must also be specified.

(Release 9.1) Use the command option "-ro" to prevent modifications without the master OSN.

filename is any optional *filename* or full *pathname* to a directory to which you have write-permission. If omitted, the original *filename* for the program in memory is used. An error is generated if the current program was not previously saved using **PSAVE, SAVE, VSAVE**, or **CHAIN SAVE**.

PSAVE is only available at *command mode* and is used to initially encrypt BASIC programs. The **OEM** command prints the current list of **OSN** numbers installed on your system. You can only **PSAVE** an application with an **OSN** that is a Master **OSN**.

Once a program is encrypted using **PSAVE**, normal **SAVE** or **RSAVE** commands preserve the encryption status. Protection on a system is driven by three distinct security numbers:

SSN System Security Number

OSN OEM Security Number

PDN Product DESCRIPTION Number

Only the **SSN** is necessary to run UniBasic; the **OSN** and **PDN** are optional numbers used in the program protection scheme. The **PDN** is of interest only to the dealer.

PSAVE simultaneously encrypts and saves the current program using a key derived from the dealers own company name and/or DESCRIPTION of his product. A protected program may be copied to a user's system, but is prevented from execution until the system is authorized by the dealer. Should someone attempt to **RUN**, **CHAIN** to, or otherwise load an unauthorized program, the following error is generated:

```
Not a loadable program file; wrong Revision, protected
or corrupted
```

If the application is authorized, the program becomes executable by **RUN**, **CHAIN**, etc. but may not be decoded using **FIND**, **DUMP**, or **LIST**. Program changes may be entered, checksums taken, and the program re-saved, and the **PSAVE** encryption status is always maintained. It is impossible to remove this encryption without first decoding the program to text, and reloading. Decoding operations are prohibited unless a Master **OSN** is installed by the owner of the application.

The **OSN** is the number used to authorize a user's system. Each **OSN** entered on a system is displayed each time UniBasic is started, e.g.:

```
Authorized software:

     DESCRIPTION

1M   ABC Software Inc.
```

1 indicates that this is the first package authorized on this system. As implied, there may be many different packages authorized to run on a

single machine. The **M** indicates that a Master **OSN** is installed, and protected programs may be decoded. If an **M** is not displayed at startup, or by the **OEM** command, then a User **OSN** is installed allowing execution and changes only, without decoding.

To perform in-house development using **PSAVE**, the machine should be authorized with a Master **OSN**'s. The Master has the same properties as the User with the exception that decoding of protected programs is allowed.

The concept of a Master **OSN** has an interesting application in field debugging. Such a number makes it possible for the dealer to enter it at a user site and temporarily be granted decoding capabilities for his programs. To accomplish this, sign on to the system, and issue the command:

```
UniBasic -t
```

You will be asked to enter a temporary **OSN**. The Master **OSN** is entered and becomes effective only for that port and only for that session. All other users on the system remain unaware that the Master **OSN** was entered. This is particularly effective for cases like modem debugging.

If no **OSN**'s are entered on a system, **PSAVE** is ignored and performs a normal **SAVE** command.

EXAMPLES

```
#PSAVE 3, <22> filename!

#PSAVE

#PSAVE -ro 3,<22> filename!
```
(Release 9.1)

ERRORS

No program in partition

Write Protected File

Program Channel not OPEN; cannot RSAVE until SAVE/PSAVE issued

File already exists; use '!' to replace

Illegal Filename

IF without ENDIF

See also

Filenames and Pathnames, File Attributes, Protections and Permissions, **OEM** command, **SAVE, RSAVE, VSAVE, LUST**

RENUMB

SYNOPSIS

Renumber statements in a program.

SYNTAX (IRIS)

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

SYNTAX (BITS)

RENUMB {*begin stn*} {, *step* }

DESCRIPTI0N

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. If *step* is a *label*, its current statement number is used as the step.

RENUMB is used to make room for new statements when all statement numbers have been used.

When statements such as **ON, GOTO, GOSUB, THEN** or **ELSE** point to non-existent statement numbers, an error is generated and you are asked whether to proceed, and all references to non-existent lines are cleared. If the non-existent statement numbers are outside the range of the old or new numbering, they are cleared. If an overlap occurs, the non-existent statement numbers are changed to : (null label). For example:

```
1 GOSUB 1000 ! 1000 non-existent & out of range

2 STOP

3 END

RENUMB

Non-existent lines referenced.

Continue with renumber? (Y-N/N) y

Line  Referenced by

1000  10

LIST

10 GOSUB 1000

20 STOP

30 END
```

The default parameters 10,10 are used during renumbering starting at statement 10 and progressing in steps of 10. In the example above, line 1000 does not exist, and is outside the range of the program before and after renumbering if the default is used. The reference to statement 1000 remains unchanged allowing later entry of that statement.

In the following example, the first line references statement number 30, which is non-existent. The default renumbering parameters of

10,10 result in the creation of a statement 30. Assuming that it is not the intention of the program to resolve the reference to the newly created statement number 30, all references to statement 30 are replaced to reference: (a null label).

```
NEW *** IRIS program ***

1 GOSUB 30 ! 30 non-existent within range

2 STOP

1000 END

RENUMB

Non-existent lines referenced.

Continue with renumber? (Y-N/N) y

Line  Referenced by

30    10

LIST

10 GOSUB :

20 STOP

30 END
```

EXAMPLES

```
1000 RENUMB 10

*RENUMB 1000,10
```

ERRORS

Non-existent lines referenced

See also

Statement Numbers, Starting & Ending Statement Numbers

RSAVE (BITS only)

SYNOPSIS

Re-SAVE the current program.

SYNTAX

***RSAVE**

DESCRIPTI0N

RSAVE is only available at *command mode* and is used to re-save a program using the same *filename*.

If the program was previously encrypted using **PSAVE**, the encryption status is preserved.

A **CHECK** command is performed prior to re-saving the program to

verify the logic of Blocked-**IF** statements. If any discrepancies exist, an error is printed.

A *checksum* is maintained for each saved BASIC program file. When **RSAVE** is performed, the *filename*, checksum and type of BASIC program is displayed:

```
#RSAVE /u/2/file Check=AF3E ***IRIS Program***
```

RSAVE is only available when operating in the environment **BASICMODE = BITS**. When operating in an IRIS environment, the **SAVE** command performs a re-save whenever a *filename* is not specified.

EXAMPLES

```
*RSAVE
```

ERRORS

No program in partition

Write Protected File

Program Channel not OPEN; cannot RSAVE until SAVE/PSAVE issued

File already exists; use '!' to replace

IF without ENDIF

Illegal Filename

See also

Filenames and Pathnames, File Attributes, Protections and Permissions, **OEM** command, **SAVE, RSAVE, VSAVE, LUST**

RUN

SYNOPSIS

Execute a program in memory or on disk.

SYNTAX

{*stn*} **RUN**

#**RUN** {*filename*}

DESCRIPTI0N

stn is any statement number contained within the current program. This form of the command is restricted to BASIC *program mode*.

filename is any legal *filename* or *pathname* to a SAVED BASIC program file to which you have read-permission. This form of the command is only available in *command mode*.

An initial **RUN** without a supplied *stn* unassigns all variables, closes all channels and begins execution at the lowest numbered statement of

the program.

stn **RUN** may be used in debug mode following a **STOP**, **END**, non-trapped error, **[ESC]**, or **[EOBC]** (usually CTRL+D) to resume execution of a program at a specific statement. It is not generally possible to perform a *stn* **RUN** prior to a **RUN** since channels are not open and variables are un-initialized. When loading a BASIC program which was saved with variables (**CHAIN SAVE**, or **VSAVE**, you may perform a *stn* **RUN** if you first manually open the required files using *immediate mode* statement execution.

EXAMPLES

```
#RUN FILENAME

100 RUN

RUN
```

ERRORS

No such statement number

See also

Filenames and Pathnames, Program Debugging Aids, **LUST**

SAVE

SYNOPSIS

SAVE the current program.

SYNTAX

#**SAVE** {{<*attributes*>} {*filename*{**!**}}}

SAVE {{<*attributes*>} {*filename*{**!**}}}

DESCRIPTI0N

<*attributes*> are any optional valid file *attributes*, *protections*, or *permissions* to apply to the file on creation. Standard IRIS, BITS, or Unix permissions may be supplied. If omitted, file creation is defaulted to permissions 0666, subject to any **umask** 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. If omitted, the original *filename* for the program in memory is used. An error is generated if the current program was not previously saved using **PSAVE**, **VSAVE**, or **CHAIN SAVE**.

If the program was previously encrypted using **PSAVE**, the encryption status is preserved in the new *filename*.

A **CHECK** command is performed prior to saving the new *filename* to verify the logic of Blocked-**IF** statements. If any discrepancies exist, an error is printed.

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

A *checksum* is maintained for each saved program file. When any **SAVE** operation is performed, this checksum and type of BASIC program is displayed.

EXAMPLES

```
#SAVE <22> prog! check=AF3E ***IRIS Program***<.font>

SAVE <PWD> dat! check=FFEB ***BITS Program***
```

ERRORS

No program in partition

IF without ENDIF

See also

Filenames and Pathnames, File Attributes, Protections and Permissions, **OEM** command, **SAVE, RSAVE, VSAVE, LUST**

SHOW

SYNOPSIS

Show all statements which contain a specific variable.

SYNTAX (IRIS)

{*starting stn*} **SHOW** {*opt*} *variable* {*ending stn*}

SYNTAX (BITS)

SHOW {*opt*} *variable* {*starting stn*} {, *ending stn*}

DESCRIPTI0N

starting stn is an optional first *stn* in the current program to search for variable. If omitted, the first *stn* is selected. If the *starting stn* does not exist, the first existing higher *stn* is used.

opt are optional parameters to control the display. Each parameter is a single letter preceded by / or - :

V Visual mode. The first screen full of lines are displayed. If additional lines are included, the user is prompted at the bottom of the screen with **[MORE]**. Pressing **[RETURN]** displays additional program lines one at a time. Pressing **[SPACE]** displays the next screen full of lines starting with the last line of the previous screen. This process is repeated until no more lines are found, or **[ESC]** or **[EOBC]** (usually CTRL+D) is pressed.

Tn Set the tab stop to column 'n'. Statements are tabbed to column 'n' for easier readability. This option is most useful when statement labels are used instead of standard statement

numbers.

L Substitute labels for statement numbers in **GOTO**, etc. wherever possible.

N Do not list statement numbers.

variable is any *mat.var, array.var, num.var* or *str.var* to search the specified statements for. For each statement containing usage of *variable*, that statement is decoded, otherwise it is omitted.

ending stn is an optional last *stn* in the current program to search. If omitted, the highest statement number is selected. If the *ending stn* does not exist, the first existing lower *stn* is used.

EXAMPLES

```
SHOW -V A$ 1000,2200

100 SHOW -N -V data 2900

SHOW -V T$
```

ERRORS

No program in partition

Illegal statement number

See also

Filenames and Pathnames, File Attributes, Protections and Permissions, **OEM** command, **SAVE, RSAVE, VSAVE, LUST**

SIZE

SYNOPSIS

Display memory usage for current program/data.

SYNTAX

SIZE

DESCRIPTI0N

The **SIZE** command displays the amount of memory allocated for the storage of a current program and variables. Unused space is also displayed:

```
SIZE: Unused=16370, (Prog)=14, (Vars)=0
```

Unused is the amount of available memory before a reallocation is necessary.

(Prog) is the number of bytes used to store the PCODE encoded program.

(Vars) is the number of bytes used to store data for variables.

The **UNASSIGN** command may be used to clear space occupied by

(Vars). A **NEW** command results in the release of all memory, and a reallocation based upon the default values. This reallocation does not occur during **CHAIN**.

EXAMPLES

```
SIZE
```

ERRORS

none

See also

UNASSIGN, NEW

STATUS (IRIS only)

SYNOPSIS

Prints the name of current program file and execution status.

SYNTAX

STATUS

DESCRIPTI0N

STATUS prints the current execution status and program filename. For example:

```
Now at (line#) in (program name)
```

where:

line# is the line number of the last statement executed.

program name is the filename of the current program.

If the program is halted after the end of the program, *line#* is shown as 0. When the current program does not have a filename, the status is printed as:

```
STATUS at (line#)
```

When the current program is a subprogram, the status of the current program as well as the calling program(s) is displayed.

When the current program is a child from a SWAP, the status of the current program as well as the parent program is displayed.

```
Now at statement:  2310;1 in: HELP.DISP

CALL at statement: 1075;3 in HELP

CALL at statement: 1230;1 in: EXEC

SWAP at statement: 3290;1 in AR.MENU
```

This output is identical to that produced by the **STOP** and **SUSPEND** statements.

EXAMPLES

```
STATUS
```

ERRORS

none

See also

STOP, SWAP, END, CALL

TIME

SYNOPSIS

Display current system time & usage.

SYNTAX

#**TIME**

DESCRIPTI0N

The current system time is displayed in the form:

```
DD Mon Year HH:MM:SS CPU=cpu used Connect= connect used
```

DD is the current day of the month.

Mon is a three-letter month name, such as JAN.

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.

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
```

ERRORS

none

See also

CALL $TIME, MSF, MSF, TIM, SPC

UNASSIGN

SYNOPSIS

Unassign all variables.

SYNTAX

UNASSIGN

DESCRIPTI0N

All variables are unassigned, including common variables. Memory shown by the **SIZE** command for (Vars) is now zero.

UNASSIGN is similar to loading a new program. All dimensioned space and values are cleared without clearing program statements.

Whereas the **NEW** command clears both program and variables, UNASSIGN does not disturb any program statements.

EXAMPLES

UNASSIGN

ERRORS

none

See also

NEW, SYSTEM 4, SYSTEM 5

USERS

SYNOPSIS

Display current number of ports in use.

SYNTAX

#USERS

DESCRIPTI0N

The Message Queues are searched for all in-use *port numbers*. The total number is then displayed on the terminal.

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

EXAMPLES

```
#USERS
```

ERRORS

none

See also

Port Numbering and Phantom Ports, Message Queues

VARIABLE

SYNOPSIS

Control and display variables.

SYNTAX

VARIAB{LE} { (+ | -) }

VARIAB{LE} { = } {{<*attributes*>} *filename*{**!**}}

VARIAB{LE} *old name = new name*

DESCRIPTI0N

+ or - enables or disables the use of long variable names in BASIC *program mode*. This command overrides the setting of the environment variable **LONGVARS**. The new setting remains until changed. The setting is not affected by **NEW, GET, BASIC, LOAD** or other commands.

A program containing long variable names may be **RUN** in either mode. Program changes, loading of BASIC Text Files, **PRINT** in immediate mode, etc. require long variables enabled if they are used within a program. Both long and short variable names may be used when long variables are enabled.

Care should be exercised using long variable names. Spaces are required between keywords; spaces or parentheses are required around functions. For example, **LENA$** must be entered as **LEN A$** or **LEN(A$)** to avoid the creation of a *str.var* named **LENA$**.

The second general form, **VARIABLE =** is used to print the current variables used by a program. The = operator forces the display of the variable and its value. Omission of the = produces a listing of variable names only.

<*attributes*> are any optional valid file *attributes, protections*, or *permissions* to apply to the file on creation. Since the file is created as a standard Unix Text file, Supplemental Protection Attributes are not permitted. Standard IRIS, BITS, or Unix permissions may be supplied. If omitted, file creation is defaulted to permissions 0666 (Read/Write by all users) subject to any **umask** in effect.

filename is any *filename* or *pathname* to a directory to which you have write permission. If the *filename* already exists, it must be terminated by a **!** to replace its contents. The file is built as a standard Unix Text File compatible with standard editors such as **vi**. *filename* may also be a *pipe* by beginning the *filename* with a **$**.

The *filename* may be the name of a *device* or *pipe*, otherwise a text file is created with the report.

In the third general form, **VARIABLE** *old name = new name*, *old name* is the name of an existing variable name used within a program. *new name* is the new variable name to replace all occurrences of *old name* in the current program.

EXAMPLES

```
VARIABLE +

VARIABLE V=RECORD_VAR

VARIABLE = $LPT

*VARIAB +
```

ERRORS

none

See also

Variable Names, Filenames and Pathnames, Accessing Drivers ($LPT) and Pipes

VERIFY

SYNOPSIS

Check program & display checksum & type.

SYNTAX

#**VERIFY**

DESCRIPTI0N

The current program is checked for illegal Blocked-**IF** statements. If any are located, an error is printed. Otherwise, the following is displayed:

```
Check; File= 0, Current= 32E2 ***IRIS program ***
```

Check; File is the original checksum of the program when last loaded from disk. A zero is displayed if the program has never been saved.

Current is the current checksum. It will be different from the File= checksum if any program changes have been made.

The type of program *** IRIS Program *** or *** BITS Program *** is displayed.

EXAMPLES

```
#VERIFY
```

ERRORS

IF Without ENDIF

ELSE Without IF

ENDIF without IF

See also

CHECK, SAVE

VSAVE (BITS only)

SYNOPSIS

Save the current program with variables.

SYNTAX

__VSAVE__ {{<attributes>*} {*filename*{__!__}}

DESCRIPTI0N

<attributes> are any optional valid file *attributes*, *protections*, or *permissions* to apply to the file on creation. Standard IRIS, BITS, or Unix permissions may be supplied. If omitted, file creation is defaulted to permissions 0666 (Read/Write by all users) subject to any **umask** in effect. If *<attributes>* are supplied, a *filename* must follow.

filename is any optional *filename* or full *pathname* to which you have write-permission. If omitted, the original *filename* for the program in memory is used. An error is generated if the current program was not previously saved using **PSAVE, SAVE**, or **CHAIN SAVE**.

All variables, **GOSUB** stack, **FOR/NEXT** stack, User Defined Function stack are saved. A prompt 'with variables' is displayed during the **SAVE** as well as during later loading of the program using **BASIC** or **GET**.

VSAVE is used to save a copy of a program for later debugging. Any open file information is not saved. Applications may use a combination of error-branching (**ERRSET, ERRSTM**, or **IF ERR**) and **CHAIN "\377\VSAVE filename"** to facilitate later debugging of an application failure.

EXAMPLES

```
*VSAVE <666> ERRORS23

*VSAVE PROGRAMERROR!
```

ERRORS

Filename already exists; use "!" to replace

No program in partition

Write Protected File

Illegal Filename

IF without ENDIF

See also

CHAIN "SAVE ...", CHAIN, SAVE, File Attributes, Protection and Permissions, Filenames and Pathnames

UniBasic Statements

This section describes the wide variety of statements that make up the numbered lines of

a BASIC program. The computer runs a program by executing the statements in logical order.

There are three modes of BASIC statement execution:

Programming Mode	Numbered statements are entered while in BASIC *program mode*. A collection of statements (program) is executed by a **RUN**(c) command.
Run Mode	Execution of a program begins with a **RUN** command and continues until normal termination (**END** or **CHAIN ""**), abnormal termination (**STOP** or Breakpoint), non-trapped Error, non-trapped **[ESC]** or Escape Override Branch Character **[EOBC]** (CTRL+D).
Immediate Mode	Each statement entered without a statement number, in *program mode*, is performed immediately.

In this chapter, statements are listed alphabetically with the general forms given in terms of literal elements in **bold** type or metalinguistic variables in Backus-Naur form in *italic* type. Bold type is used for all key words such as utilities, statements, functions, and environment variables. Key words are all cross-referenced in the Index at the back of this guide. Each statement begins on a separate page and conforms to the standard format.

Program Debugging Aids

Extensive program debugging aids are included. Any BASIC statement may be executed immediately by entering the statement without a statement number using *immediate mode*.

Special program termination (**STOP**, Breakpoint), non-trapped Error or forced termination (ESCAPE or CTRL+D) leaves all channels OPEN and available for immediate statement operations. To resume execution, simply type **CONTINUE**.

Single-Step Program Execution

Single statement execution is performed by entering a period and pressing return. The current statement is executed and the next statement to execute is displayed. Subsequent periods are used to step through the program. To force execution at a specific statement, issue a **GOTO** *stn* and press [RETURN]. Single step or **CONTINUE** can then be performed from that *statement number*. 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 period and pressing return; or bypassing the single step operation of a sub-program by entering two periods and pressing return. Bypassing with two periods actually performs the subprogram, but

does not single step through it.

Trace Mode

Trace mode is used when it is desirable to observe the statement number program flow without performing single steps. **SYSTEM 20**, **TRACE ON**, or **TRACE ON *chn num*** enables tracing; **SYSTEM 21**, **TRACE OFF**, or **CLOSE *chn num*** turns trace off. These statements may be used in *immediate mode*, or imbedded within specific code segments of a program. For each statement executed, the statement number *stn* and sub-statement number *sub-stn* (statements on the same BASIC line) is printed.

The following information is displayed on the terminal during *trace mode*:

```
TR - statement number ; sub-statement number        BITS

[statement number]                                  IRIS
```

In BITS mode, "TR -" indicates trace mode is enabled and the next *stn* and *sub-stn* to be executed are displayed. In IRIS mode, a new-line is performed, and the stn only is displayed within [ ]. The execution of the statement then proceeds. Output from a **PRINT** is displayed following the trace information.

Program Breakpoints

Breakpoints are used to terminate normal execution when a specific *stn* is reached. The statement **SYSTEM 22**, **stn** sets a breakpoint at statement *stn*. Typing *stn* **RUN** or **CONTINUE** resumes execution until the statement is reached. **SYSTEM 23** clears any active breakpoint. These statements can be within a program or executed in *immediate mode*. By inserting breakpoint statements within a program, you can control when a breakpoint is set, for example:

```
229 SYSTEM 22,6620 \ GOSUB 6600 \ SYSTEM 23

Brk @ stn:stn
```

To resume execution at the breakpointed statement, issue a **SYSTEM 23** to clear the breakpoint, followed by a **CONTINUE** command.

A breakpoint will not occur if *trace mode* is enabled when the breakpointed statement is reached. When a breakpoint occurs, any other **SYSTEM** or **IOxx** modes return to their default states.

Statement Documentation Format

Each statement Statement Documentation Format occupies one page, documented in the form:

STATEMENT

SYNOPSIS

UniBasic statement to ...

SYNTAX

STATEMENT {*num.var*,} *str.var*

DESCRIPTION

General description of what the statement does and how it uses arguments.

EXAMPLES

statement examples

ERRORS

Text description of errors likely to result from the improper use of the statement. For a complete list of all errors and their descriptions, see Appendix C

See also

COM, ISAMFILES Other index keywords to refer to for additional information.

BUILD

SYNOPSIS

Build and open a new text or data file.

SYNTAX

BUILD #*channel*,{+}*filename.expr* {,{#*channel*,}{+}*filename.expr*}

DESCRIPTION

The *channel* expression is evaluated, truncated to an integer and used to select the channel on which to open the file once it has been created.

The optional + character specifies that the file is to be built as a standard text file without any special header information.

Each *filename.expr* contains the file's *attributes* and *filename* to be created. Multiple strings may be specified to create several files and they will be opened on successive *channel* numbers. Any *new channel number (#channel)* in the filename list will cause assignment of channels to continue from that number.

The *attributes* are optional and may consist of several items, selecting the type, structure, and protection of the file. If attributes are to be selected, they must be specified in the form <*attributes*> and precede the filename.

The *filename* is any legal *filename*, operating system full *pathname* beginning with /, or a blank quote (""). If the *filename* is to replace an existing file on the system, the name must be terminated with an exclamation point (**!**). When the "" is used, BUILD is used to check for "Channel already open" prior to building the file.

If the file is to be created as a Contiguous data file, the initial *Record Count* and *Record Length* must be specified in the form "[count:length]". The *Record Count* is the initial number of records to be allocated to the file. *Record length* is specified in 16-bit words.

If no record count/length is specified, the file is created as a Formatted Item file. The Record Length and format is defined by the program when Record 0 is written.

If the *str.expr* defining the filename is preceded by a + sign (note: the + character is not within the *str.expr*), the file is created as a text file.

EXAMPLES

```
BUILD #0,"2/ABC" , + "/usr/ub/3/textfile!"

BUILD #C,"<644> [1000:256] PAYROLL/CFILE!"
```

ERRORS

Illegal parameter or syntax for command

File already exists; use "!" to replace

Illegal filename

Illegal channel number specified

Channel is already OPEN and in-use

See also

CREATE, File attributes and permissions, channel, filename, files, **PREALLOCATE, IBITSFLAG**

CALL

SYNOPSIS

Call an external BASIC or C-Language subroutine.

SYNTAX

CALL *filename.expr* | *$string* | *num.expr* , *var.list*

DESCRIPTION

CALL is used to invoke the execution of a BASIC subroutine, named C subroutine or numbered C subroutine. Any given **CALL** statement may only invoke one of the three types.

To call a named BASIC subroutine, *filename.expr* is any *str.expr* which contains the name of BASIC sub-program to be executed.

To call a named subroutine written in C, *$string* selects the named routine.

To call a numbered subroutine written in C, *num.expr* selects a numbered subroutine in the range 1 to 128.

Named and numbered C subroutines must be compiled and linked

into your system by DCI or a qualified systems programmer. BASIC subroutines may be created for use on any UniBasic installation by experienced BASIC programmers. No special linking is required to create or utilize BASIC subroutines.

Either a comma or semicolon separates the CALL name/number and the *var.list*.

BASIC programs called as subroutines are referred to as *subprograms*. A subprogram accepts a list of argument variables passed by the calling program by use of the **ENTER** statement. The number and type of arguments in the **CALL** statement must match those in the **ENTER** statement of the called program, and only one **ENTER** statement is allowed in the subprogram. The maximum number of arguments is limited only by the maximum statement length.

A subprogram accepts and returns values through the passed list of arguments which may be any combination of: *str.var, num.vars, array.vars,* or *mat.vars*. *num.vars* must be **DIM**ed in the calling program or an IRIS error 33 will occur. Variables are passed by reference meaning that the actual names of the variables may differ from the calling program to the subprogram. For example, if the calling program passes A$ and T, the subprogram may **ENTER** with DATA$ and VALUE. The variable names specified by **ENTER** are mapped to reference the data space of the variable names passed in the **CALL**. All other variables in a subprogram are considered local to the subprogram.

Expressions, including substrings, are passed to subprograms by value. If a subprogram updates or returns a value in a referenced variable, that operation will be lost if the caller passed an expression.

Subprograms can be nested indefinitely, limited only by the maximum process size of Unix.

Standard named and numbered C subroutines are documented in the User Calls section of this guide. Certain systems may include additional **CALL** statements. For a complete listing of **CALL** statements included with your system, contact your distributor.

A maximum of 63 arguments may be passed to a **CALL** C subroutine.

The **var.list** may be defined as any combination of *str.vars, num.vars, mat.vars, str.exprs, num.exprs, array.vars* or *str.lit*, depending on the requirements of the subroutine being called. A *mat.var* in **CALL** or **ENTER** is given a *num.var* with empty subscripts; e.g. A3[]. The subroutine may use these items for input and output of data. A variable (not an expression) must be specified in positions of the *var.list* which return information to the program.

CALLS accessed by number can be re-mapped using the environment variable **ALTCALL**. For example, if your application utilizes a **CALL 60**, which is provided as **CALL 20**, include in the *.profile* the command: **ALTCALL=20:60**.

Table of CALL Names & Replacement Numeric CALL ID's

$NAME	Replacement No.	$NAME	Replacement No.
$ATOE	77	$RDFHD	97
$AVPORT	--	$RENAME	--
$CKSUM	--	$STRING	82
$ECHO	78	$SWAPF	--
$ETOA	76	$TIME	99
$FINDF	96	$TRXCO	98
$INPBUF	--	$VOLLINK	91
$LOCK	--	$WINDOW	--
$LOGIC	88		

EXAMPLES

```
CALL "pgm",A$,B[],C[2],INPUT$
CALL 98, P, A$, A, P1
CALL $STRING, A$
```

ERRORS

Parameter list overflow

Error detected in/by user CALL routine

Not enough parameters passed to user CALL

User CALL parameters out of order

Subprogram file not found

See also

User **CALLS, ENTER, LIB**

CHAIN

SYNOPSIS

Transfer control to another program.

SYNTAX

CHAIN *filename.expr*{,*num.expr*{,*num.var*}}

DESCRIPTION

The *filename.expr* is any *str.expr* containing the *filename* of a BASIC program (type B) to which you have access. If the program is found and is not protected against you, **CHAIN** terminates execution of the

current program and **RUN**s the selected program. If the program is not accessible, an error is generated and the current program remains intact.

The optional *num.expr* selects a starting *stn* in the new program to begin execution. If not specified, execution begins with the first *stn*.

The *num.var*, if included, is set to the *stn* following the **CHAIN** in the current program. The variable should be common, and may be used to chain to a subroutine, passing the return *stn* in the variable. In this case, the second program must contain the necessary **COM** or **CHAIN READ** statement.

CHAINing to a null string terminates the current program and returns the user to **SCOPE** *command mode*.

There are two types of **CHAIN** operations; *short* and *long*.

A *short* **CHAIN** transfers control from one BASIC program to another. All files remain open and common variables are passed using **COM** or **CHAIN READ / CHAIN WRITE**. A *short* **CHAIN** is performed if the *filename.expr* is the name of an existing BASIC program, or begins with the string 'RUN' or 'run'.

Only type B (SAVE) files may be *short* **CHAIN**ed to. If the selected file is not accessible or is not type B, a *long* **CHAIN** is performed. Certain BASIC SAVE programs reclassified as SYST (System BASIC programs) are not available in *short* **CHAIN** mode. These are supplied system commands, such as **LIBR** and are designed to be treated as commands instead of programs.

A *long* **CHAIN** appends the supplied *filename.var* to the type-ahead buffer, exits the program to *command mode*, and processes type-ahead as though the command was entered from the keyboard.

Several commands may be within a *long* **CHAIN**, and they are executed in sequence. A *long* **CHAIN** is performed for IRIS programs whenever a *short* **CHAIN** fails.

Each command should be terminated with an **[EOL]** terminator, usually \215\ or CTRL+Z. The number of characters that can be passed in this fashion is limited to the size of the user's input buffer (value of **INPUTSIZE** environment variable).

The existing contents of the type-ahead buffer may be cleared by specifying the character \230\, \231\, \210\ or \377\ as the first character of the supplied *str.var*. The IRIS \231\ mode providing for inserting data before the current contents of the type-ahead buffer is not supported at this time.

Any *long* **CHAIN** which enters or passes input to *command mode* first closes all channels.

Any **CHAIN** terminates the current program.

For BITS applications, all **CHAIN** operations are assumed to be short unless an ASCII 377_8 is the first character of the supplied *str.var*. An error is generated if the supplied program name is not found.

Following the program name, the inclusion of a \377\ code provides for appending data to the type-ahead buffer.

Note: If characters are passed through to the input buffer using *long* **CHAIN**, a terminating [RETURN] code is appended to the string unless the \377\ character is explicitly used.

The \377\ character must be explicitly used in a BITS program to send a command through *command mode*. This character may also be used in IRIS programs to force a *long* **CHAIN**. All data following the \377\ is appended to the type-ahead buffer.

EXAMPLES

```
CHAIN "3/FILENAME"

CHAIN "LIBR [OUTPUT]\215\RUN PART2"

CHAIN Q$,4000,B

CHAIN "\377\DIR /L=$LPT\215\RUN MENU\215\"
```

ERRORS

No such line (stn) number

File does not exist

Not a loadable program file; Protected, wrong revision or corrupted

See also

COM, CHAIN READ, CHAIN WRITE, INPUTSIZE, BASICMODE

CHAIN READ

SYNOPSIS

Read variables from a previous program.

SYNTAX

CHAIN READ *var.list*

CHAIN READ =

CHAIN READ *

DESCRIPTION

CHAIN READ specifies common variables passed to this program via **CHAIN WRITE** statements in a preceding program. Multiple **CHAIN READ** statements may be used, and they may be placed anywhere within a program. Variables listed in a C**HAIN READ** may not be dimensioned by a **DIM** statement.

CHAIN READ = causes all variables passed as common to be read into the program. All such variables must appear in the program at least once (even if not used).

CHAIN READ * functions like **CHAIN READ =** except that variables passed to, but not appearing in this program are ignored.

The **CHAIN READ** statement is ignored if executed. When a program passes data to another using **CHAIN WRITE**, the new program's **CHAIN READ** statements are executed during the **CHAIN** operation.

The actual **CHAIN READ** statements may be placed anywhere in a program, however the best method is to group them together at the beginning of a program near your **DIM** statements.

CHAIN READ statements may not be used together with **COM**.

EXAMPLES

```
CHAIN READ A,B,C,X$

CHAIN READ *
```

ERRORS

Variable in CHAIN READ not passed by CHAIN WRITE

Variable from CHAIN WRITE not in this program

Variable in CHAIN READ already contains data

See also

CHAIN WRITE, COM

CHAIN WRITE

SYNOPSIS

Write variables to the next program.

SYNTAX

CHAIN WRITE *var.list*

CHAIN WRITE *

DESCRIPTION

CHAIN WRITE statements specify variables to be passed as common to the next program. All variables specified must be dimensioned or otherwise have a value assigned to them in order to be passed. It is the responsibility of the receiving program to contain the necessary **CHAIN READ** statements to accept the data.

All variables are passed complete to their dimensioned length, such that strings with embedded nulls are passed in their entirety.

A **CHAIN WRITE** must not be directly executed. Multiple **CHAIN**

WRITE statements may be used, and should only be placed as a group after a **CHAIN** or **SWAP** statement (intervening REMs are allowed).

CHAIN WRITE * passes all variables in the program as common. It cannot be used with any other **CHAIN WRITE** statements.

CHAIN WRITE statements may not be used together with **COM**.

EXAMPLES

```
CHAIN WRITE A,B,C,X$

CHAIN WRITE *
```

ERRORS

Illegal function usage

Variable in CHAIN WRITE contains no data

See also

CHAIN READ, COM

CLEAR

SYNOPSIS

Clear {all} open channel.

SYNTAX

CLEAR {*#channel* {*,#channel*}|}

DESCRIPTION

The *channel* expression is evaluated, truncated to an integer and used to select the *channel number* (0 to 99) to clear. Multiple channels, separated by comma may be cleared. If no **#channel** is given, all opened files (Channels 0 to 99) are cleared. Record locks on the file are removed, the file header may be updated and the system file descriptor is released. A cleared channel is available for re-use for another file.

The current BASIC program is said to be open on channel -1; to clear the program channel, use **CLEAR #(-1)**.

If an Indexed Data File is opened, both the data and companion index file are cleared.

Clearing an output pipe causes the reading processes to receive an EOF (end of file) at its next read operation. Printer drivers and other scripts and pipes commonly open will terminate on the EOF.

If the file opened is a newly built Formatted Item file and at least one item has been written, the record format is frozen and the header is updated with the current record length and item count. If no items have been written to a newly built file, the file is unformatted and the

next **OPEN** must define the record format.

IRIS programs generate an error when a specified *#channel* is not currently open.

EXAMPLES

```
CLEAR #5,#8,#X+2

CLEAR
```

ERRORS

Illegal channel number

Channel not open

See also

OPEN, Files, Channel, **CLOSE**

CLOSE

SYNOPSIS

Close {all} open channel.

SYNTAX

CLOSE {*#channel*{,*#channel* }}

DESCRIPTION

The *channel* expression is evaluated, truncated to an integer and used to select the *channel number* (0 to 99) to close. Multiple channels, separated by comma may be closed. If no *#channel* is given, all opened files (Channels 0 to 99) are closed. Record locks on the file are removed, the file header may be updated and the system file descriptor is released. A cleared channel is available for re-use for another file.

The current BASIC program is said to be open on channel -1; to close the program channel, use **CLOSE #(-1)**.

If an Indexed Data File is opened, both the data and companion index file are closed.

Closing a pipe causes the reading processes to get an EOF (end of file) at its next read operation. Printer drivers and other scripts and pipes commonly open will terminate on the EOF.

If the file opened is a newly built Formatted Item file and at least one item has been written, the record format is frozen and the header is updated with the current record length and item count. If no items have been written to a newly built file, the file is closed unformatted and the next **OPEN** must define the record format.

IRIS programs generate an error when a specified *#channel* is not currently open.

EXAMPLES

```
CLOSE #1

CLOSE #5,#8,#X+2

CLOSE
```

ERRORS

Illegal channel number

Channel not open

See also

OPEN, UniBasic Files, Channel, **CLEAR**

COM

SYNOPSIS

Specify Common Variables.

SYNTAX

COM {***%p***,}*var.list* { , {***%p***},*var.list* }

DESCRIPTION

The **COM** statement allocates space and defines precision for variables which can be passed between programs. The form is identical to the **DIM** statement, except that all variables defined by **COM** are flagged as common and eligible to be passed during **CHAIN**.

Precisions can be defined for the variables in the *var.list* by including the optional **%p** or **p%** precision. All further variables in the *var.list* will be at the last specified precision. The default precision is 2% for IRIS, and %4 for BITS applications. The last supplied precision in a **COM** or **DIM** statement is used as the default for all automatically assigned variables.

All **COM** statements in a program must be executed before any statement which allocates or defines a new variable (**LET, DIM, IF**, etc.). Statements such as **REM, ESCSET, GOTO**, etc. which use no variables may precede **COM**. An error is generated if a **COM** statement is executed out of order.

Variables to be passed must be defined in a **COM** statement by each program that is to use them. Generally, two or more programs using a set of common variables will contain identical **COM** statements in order to pass the entire set between them. A program **CHAIN** may exclude certain variables in its common set, and these variables become unassigned. Similarly, the program may add variables to the set, and they will be allocated and initialized as done by a **DIM**. Numeric precision may not be changed between programs, but strings and arrays may be re-dimensioned to smaller sizes using **COM**.

CHAIN READ and **CHAIN WRITE** statements may not be used together with **COM**.

EXAMPLES

```
COM A$[19],B$[1],T4$[132]

COM C$[1762]

COM A[5],T$[120],D[23,14],%3,X[17]

COM %1,A,B,%2,C,D,%3,E,F,%4
```

PROGRAM EXAMPLES

The following examples illustrate common variables being passed between two programs, A and B.

Prog	Statement	Comment
A	10 COM %1,A,B,%2,C,D	All variables common.
B	10 COM %1,A,B,%2,C,D	
A	10 COM Q,D[3,4],S$[10]	Only S$ is common D and Q are lost during
B	10 COM S$[10],T	CHAIN. T is added to the common list.
A	10 COM T,%3,U,V	U is common, T and V are lost, Z is added.
B	10 COM %3,U,Z	

ERRORS

COM statement out of order

Variable precision is not compatible

Variable precision cannot be changed

Array size exceeds initial DIMension

A string may not be re-DIMensioned

See also

CHAIN READ, CHAIN WRITE, DIM, PRECISION

CONV

SYNOPSIS

Convert binary data to decimal.

SYNTAX

CONV *mode*, *str.var*, *num.var*

DESCRIPTION

The *mode* is any *num.expr* which, after evaluation is truncated to an integer to select the operation to be performed. *Mode* 0 converts the binary string to decimal, and *mode* 1 converts the decimal numeric value to a binary string.

The **CONV** statement extracts binary information from a *str.var* and returns the value in decimal into a *num.var*. Additionally, numeric information in a *num.var* can be converted to binary and placed into a *str.var*.

The *str.var* specifies the binary string and must define a string of one to four bytes. The *num.var* is the decimal numeric variable.

The valid numeric ranges, as well as the internal storage format, are determined by the length of the *str.var* given. This variable would usually be subscripted to select the desired length, otherwise the dimensioned length of the string would be assumed. The following table compares the string length with the range of values that can be stored.

str.var	Size	Decimal
B$[x,x]	1 byte	0 to 255
B$[x,x+1]	2 bytes	0 to 65535
B$[x,x+2]	3 bytes	0 to 16777215
B$[x,x+3]	4 bytes	-2,147,483,648 to 2,147,483,647

The conversion process allows positive integers only to be represented in 1, 2, or 3 byte lengths. A negative value must be converted to a 4 byte length to retain its negative sign. Converting a negative value to a shorter length and back would result in a truncated positive integer different from the original value.

The 4 byte length described here is identical to the internal format of a double-precision integer numeric variable written to a file, and such a value could be read as a string and converted to numeric. The 2 byte length, however, is NOT compatible with the %1 format because it is unsigned. Signed values could be converted using 1, 2, or 3 byte lengths provided the program performs an adjustment for 16-bit two's complement notation.

PROGRAM EXAMPLE

```
100 REM Convert binary to decimal D
110 CONV 0,A$[1,n],D
120 IF D>R THEN LET D=D-A
```

```
200 REM Convert decimal D to binary
210 IF D<0 THEN LET D=D+A
220 CONV 1,A$[1,n],D
```

Size (n)	Range (R)	Adjust by (A)
1 byte	-128 to 127	256 (2^8)
2 bytes	-32768 to 32767	65536 (2^{16})
3 bytes	-8388608 to 8388607	16777216 (2^{24})

This method causes the upper bit of each string to be considered a sign bit, just as is done by **CONV** with the 4 byte length. In the case of 2 bytes, for example, the values 0 thru 32767 represent themselves, while 65535 thru 32768 represent -1 thru -32768.

ERRORS

Illegal subscript specified

Subscript exceeds DIMension

See also

PRECISIONS, STRINGS

CREATE

SYNOPSIS

Create a new Data File.

SYNTAX

CREATE *#channel*, *filename.expr*{,{*#channel*,}*filename.expr*. . .}

DESCRIPTION

The *channel* is any *num.expr* which, after evaluation, is truncated to an integer and used to select the *channel* on which to open the file for read and write access once it has been created.

Each *filename.expr* may contain *file attributes* enclosed within < >, and the *filename* to be created. Multiple strings specify creation of several files and they will be opened on successive channel numbers. Any new *channel* seen in the list will cause assignment of channels to continue from that number.

The *file attributes* include the type, structure, and permissions of the file in the form **<count:len type permissions>**. **count** specifies the initial number of records to be allocated followed by a colon; **len** the fixed-record length in bytes; **type** specifies the type of data file to be created (**T** for tree-structured Data File, **I** for Formatted Item File, and **C** for Contiguous Data File); and **permissions** specifies BITS Attribute letters, IRIS Protections, or UNIX Permissions as three-

digits.

To create a standard Text File, use the **BUILD** statement.

The *filename* is any legal *filename* or operating system full *pathname* beginning with /. If the *filename* is to replace an existing file on the system, the name must be terminated with the **!** character.

If the program is an IRIS program, an error is generated if the specified *chn.expr* is already in use.

EXAMPLES

```
CREATE #3, "<100:254C PWD> PAYROLL:FILENAME!"

CREATE #R, "<1:510T644> TEMPFILE"+STR(MSC(0))
```

ERRORS

Illegal parameter or syntax for command

Illegal filename

File already exists; use "!" to replace

Illegal channel number specified

Channel is already OPEN and in-use

See also

BUILD, Attributes, channel, filename, files, **PREALLOCATE, IBITSFLAG**

DATA

SYNOPSIS

Define Internal Program Data.

SYNTAX

DATA {*str.lit* | *num.lit*} ...

DESCRIPTION

Each *str.lit* or *num.lit* is stored within the program as simple ASCII text. Multiple data items on the same line must be separated by commas in the statement, but a comma cannot be the last character of the statement.

To include commas or special characters in the form \xxx\, the data element must be quoted.

No other statement may follow **DATA** on the same program line. All text up to the end of the line is considered part of the DATA statement.

DATA statements may appear anywhere within a program and are ignored if executed, that is, they are treated like **REM** comments.

Each **DATA** statement may contain as many values as can be entered, up to the size of the input buffer as defined by the environment variable **INPUTSIZE**.

Numeric data items must be separated by comma, but can be in decimal and E-notation. A comma cannot be part of a numeric item that will be read into a *num.var*.

For IRIS compatibility, you may include a **%p** or **p%** declaration before numeric items. These items will be ignored when **READ** into a *num.var*. Since the data is stored as string (and can also be read as such), the precision is determined at the time of the **READ**.

EXAMPLES

```
DATA 200,300,400,500,600,700.25,800,23.45

DATA "quoted string, has comma", "\215\\215\"
```

ERRORS

Syntax error

See also

PRECISIONS, READ, MAT READ, RESTORE

DEF FN

SYNOPSIS

Define User Function.

SYNTAX

DEF FN<letter>(*num.var*) = *num.expr*

DESCRIPTION

The **FN <letter>** designator must be a single letter A thru Z, such as **FND**, yielding a maximum of 26 concurrently defined user functions. Each user function must have a **DEF** statement executed before it can be used. User functions may be redefined using successive **DEF** statements with the same **<letter>** designator.

The parenthesized *num.var* is considered a dummy argument. The *num.expr* is the expression to be evaluated whenever the function is called. When this occurs, the actual argument supplied will be substituted for every occurrence of the dummy argument in the given expression. Any variable currently in use with the same name as the dummy argument is not affected by the function call.

A user function may call another user function in its definition, provided the called function has already been defined. User functions may be nested in this manner up to a maximum of 8 levels.

EXAMPLES

```
DEF FNA(X)=(X^3)*(X^2)*X

DEF FNC(V)=(V^4)*FNA(V)  ! Nested FNA

DEF FNR(X)=SGN(X)*ABS(100*INT(X)+.5)/100
```

ERRORS

User defined functions nested too deep

Expression too complex for evaluation

Arithmetic error - (X/0, overflow, LOG(0) or SQR(-X))

Illegal function usage

Syntax error in **DEF**ined function

Variable not defined

User function not defined

See also

Functions

DIM

SYNOPSIS

Allocate space for variables.

SYNTAX

DIM {*%p*,} *var.list* { {*%p*},*var.list* }

DESCRIPTION

The **DIM** statement allocates space and defines precision for variables which are considered local to the current program. The form is identical to the **COM** statement, except that all variables defined by **DIM** are not automatically passed during **CHAIN** statements unless specified using **CHAIN WRITE** and **CHAIN READ**.

Precisions can be defined for the variables in the *var.list* by including the optional **%p** or **p%** precision. All further variables in the *var.list* will be at the last specified precision. The default precision is 2% (2-word floating) for IRIS, and %4 for BITS applications. The last supplied precision in a **COM** or **DIM** statement is used as the default for all automatically assigned variables.

If the *var.list* contains an *str.var*, in the form *str.var$[num.expr]*, the *num.expr* within subscripts is evaluated, truncated to an integer, and used as the maximum size of the string variable in characters. Any attempt to store data beyond this maximum results in data truncation. String variables must appear in a **DIM** or **COM** statement before use by any other statement. They can be re-dimensioned within the program to a smaller size only.

If the *var.list* contains a *num.var* without subscripts, it is allocated at

the current default precision as a simple numeric variable.

If the *var.list* contains a *num.var* in the form *num.var[num.expr]*, or *num.var[num.expr1,num.expr2]*, it is allocated at the current default precision as a one or two dimensional *array.var* or *mat.var* respectively. The expression within subscripts are evaluated, truncated to integers, and used to select the size (number of elements) of the array. Variables specifying one expression result in a one-dimensional array (vector or list). Two expressions separated by a comma result in a two-dimensional array (matrix). Any array used in a program without specifically being mentioned in a **DIM** or **COM** statement is automatically dimensioned to [10] if used as an a*rray.var*, or [10,10] when used as a *mat.var*.

It is considered good programming practice to define all variables (other than temporaries and variables to use the default precision) in a **DIM** or **COM** statement.

The final **%n** executed in your program selects the default for any run-time variable assignments. If not specified, the default precision is **%4** for BITS programs and **2%** for IRIS programs.

EXAMPLES

```
DIM A$[19],B$[1],T4$[132]

DIM C$[1762]

DIM A[5],T$[120],D[23,14],%3,X[17]

DIM 1%,A,B,2%,C,D,3%,E,F,4%
```

ERRORS

Variable precision cannot be changed

Array size exceeds initial **DIM**ension

A string may not be re-**DIM**ensioned

Illegal subscript specified

Attempt to **DIM**ension an existing simple variable

Strings can have only one **DIM**ension

Subscript exceeds **DIM**ension

See also

COM, PRECISION, STRINGS, ARRAYS, MATRICES, LET

DUPLICATE

SYNOPSIS

Make a duplicate copy of a program or file.

SYNTAX

DUPLICATE *filename.expr*

DESCRIPTION

The *filename.expr* is any *str.expr* containing the source *filename* to be duplicated, a space, and the destination *filename* for the duplicate copy.

The Unix **cp** command is used to perform the duplication operation. An exact copy of the file is created, and the process may take some time to complete. Since the command is sent to Unix, **[ESC]** and **[EOBC]** are disabled. To abort the operation, press the **BREAK** or **DEL** character.

If the file is an Indexed Data File, two **cp** commands are performed; one for the data portion (lower-case name), and one for the ISAM portion (upper-case name).

If the file is a Universal Indexed Data File, two **cp** commands are performed; one for the data portion (*filename*), and one for the ISAM portion (*filename* with an *.idx* extension).

DUPLICATE does not check for existence of the destination *filename*. If a check is desired, perform an **OPEN** or **CALL $FINDF** first.

(Release 9.1) Can be used with encrypted files without an encryption key.

EXAMPLES

```
DUPLICATE "PAYROLL PAY1QTRBKUP"

DUPLICATE "/usr/ub/23/file /u/u1/23/file"
```

ERRORS

Illegal pack or filename

See also

Unix **ln, cp, mv** commands, Filenames and Pathnames

EDIT

SYNOPSIS

Format numeric and string expressions.

SYNTAX

EDIT *format str.expr*, *destination str.var* ; *var.list*

DESCRIPTION

The *format str.expr* is any *str.expr* defining the format string to apply to the list of variables in the *var.list*. Output is formatted according to the rules for the String Operator: USING.

destination is any string variable to receive the formatted result.

var.list is any list of numeric or string variables (*num.var, str.var, num.expr, str.expr*) to be formatted into the *destination str.var*. Only numeric data is formatted, string data is copied exactly to the destination.

Each item in the *var.list* must be separated by commas.

The **EDIT** statement is used to format string and numeric output. **EDIT** operates similar to **LET USING**; formatting output and storing the result in a string variable. Unlike **LET USING**, **EDIT** allows a list of arguments for the formatted result.

EXAMPLES

```
EDIT "$#,##&.##",D$;T,E,F,"TAXES",T9

EDIT A$,B$;"TOTAL DUE",Z,"BALANCE",Q,R$,T9
```

ERRORS

Formatted Output overflows output string

See also

String Operator **USING, PRINT USING, LET USING**

END

SYNOPSIS

Normal termination of a running program.

SYNTAX

END

DESCRIPTION

The **END** statement is used to indicate the normal termination of a program. An **END** statement causes program execution to cease, and the user is returned to *BASIC Program Mode* following the prompt:

```
Ready
```

Other statements may follow an **END**, and inclusion of an **END** is optional. If a program reaches its physical end of the program and no **END** statement exists, an implied **END** is performed.

END leaves the current program (with all variables) in the user's partition. If the program is an IRIS program, all channels are closed automatically.

If the running program is a BITS program, all channels normally remain open.

If the running program executing the **END** statement has additional attributes <O> or <E>, special conditions are observed as documented under Supplemental Protection Attributes.

EXAMPLES

```
END
```

ERRORS

none

See also

STOP, SYSTEM, CHAIN, Supplemental Protection Attributes

ENTER

SYNOPSIS

Accept Variables from a **CALL** Statement

SYNTAX

ENTER *var.list*

DESCRIPTION

A subprogram accepts argument variables from a **CALL** statement within a separate BASIC program. **CALL** <subprogram> invokes a BASIC program as a subroutine.

The **ENTER** statement can be located on any line of the subprogram, but the variables cannot be used until the **ENTER** statement has been executed. This means that the **ENTER** statement should be at the beginning of the program in most cases.

Only one **ENTER** statement is allowed, and the number and types of variables in the statement must match the **CALL** statement exactly or an error message is displayed.

The *var.list* may be defined as any combination of *str.vars*, *num.vars*, *mat.vars*, or *array.vars* depending on the requirements of the subprogram. The subprogram can only return data within arguments that are passed as variables, subscripted numeric variables, or matrix variables. A matrix variable in **CALL** or **ENTER** is given as a numeric variable with empty subscripts; e.g. A3[].

If a subprogram is called with arguments, but no **ENTER** statement is executed, no error will occur and the arguments will not be changed. If a subprogram has no parameters, an **ENTER** statement with no parameters can be used to detect unnecessary arguments on the invoking **CALL** statement.

Called subprograms can be nested indefinitely, limited only by the maximum process size in Unix.

EXAMPLE

```
(from master program)     CALL PGM,B$,A,D$[4,7]

(from called subprogram)  ENTER B$,J,F$
```

ERRORS

ENTER statement is illegal if not in a subprogram

The ENTER statement can only be executed once in a subprogram

Number/types of arguments do not match parameter list

Parameter variable in ENTER statement has already been allocated

See also

CALL, LIB

EOFCLR

SYNOPSIS

Clear End-of-File branching.

SYNTAX

EOFCLR *stn*

DESCRIPTION

EOFCLR clears any special end-of-file branching in effect. Normal error processing is resumed. If an error branch is in effect from an **ERRSET, ERRSTM**, or **IF ERR**, it will be in control of further end-of-file errors.

EXAMPLES

```
EOFCLR
```

ERRORS

none

See also

IF ERR, ERRSET, ERRSTM, EOFSET

EOFSET

SYNOPSIS

Enable End-of-File error branching.

SYNTAX

EOFSET *stn*

DESCRIPTION

EOFSET traps any further occurrence of the error, “Illegal record number or End of File". If such an error occurs on any channel, the program will branch to the *stn* given in the **EOFSET** statement. *EOFSET* affects only this single error. Other errors are processed in

the current error handling mode.

IF ERR, ERRSET and **ERRSTM** statements are used to trap all errors, including end-of-file. The **EOFSET** statement is used to override normal error branching for this special error.

EOFSET is typically used by BITS applications when reading a text file. Other applications may utilize this function to handle Formatted Item or Contiguous Files.

EOFSET branching remains in effect until specifically cleared by **EOFCLR**. Other error branching disable functions do not clear this special branch.

EXAMPLES

```
EOFSET 1050

EOFCLR
```

ERRORS

No such statement number

See also

IF ERR, ERRSET, ERRCLR, ERRSTM, EOFCLR

EOPEN

SYNOPSIS

Exclusively **OPEN** a Data File.

SYNTAX

EOPEN *#channel*, *filename.expr* {, { *#channel*} *filename.expr*} ...

DESCRIPTION

The *channel* is any *num.expr* which, after evaluation, is truncated to an integer and used to select the *channel* on which to open the file for read and write access once it has been created.

The **EOPEN** statement exclusively links a selected file to a channel.

EOPEN differs from **OPEN** in that the request will exclusively lock the file to the program. Other **EOPEN, OPEN**, or **ROPEN** requests by you or other users will not be allowed.

Note: At this time, **EOPEN** cannot guarantee that another does not already have the file opened, however an **EOPENED** file cannot be subsequently opened by another user.

An IRIS application cannot **EOPEN** certain types of files such as SAVED BASIC Programs. A special error is generated when a file exists, but is generally unavailable.

EXAMPLES

```
EOPEN #1,"23/MMFILE", C$

EOPEN #2,"FILE1","FILE2",#10,"FILE4"
```

ERRORS

File is Read Protected

No such file

File is already OPENed and Locked

Channel is already OPENed

Not a data file (Can't OPEN or replace)

See also

OPEN, ROPEN, File Attributes and Permissions, Accessing Data Files Through a Channel

ERRCLR

SYNOPSIS

Clear Error Branching.

SYNTAX

ERRCLR

DESCRIPTION

ERRCLR clears any error-branching in effect and returns normal error processing to the application. Normal error processing is to abort the current running program and output the error message text:

Error in statement *stn;sub-stn* / Text description of error

Special end-of-file branching in effect from the **EOFSET** statement is not cleared by **ERRCLR**.

ERRCLR is used to clear automatic branch-on-error conditions previously set using **ERRSET, ERRSTM** and **IF ERR**.

Normal error termination does not close all opened data files.

EXAMPLES

```
ERRCLR
```

ERRORS

none

See also

Error Messages, **EOFSET, ERRCLR, IF ERR, ERRSTM JUMP, ERR, SPC, MSC, MSF**

ERRSET

SYNOPSIS

Enable Branch to statement on error.

SYNTAX

ERRSET *stn*

DESCRIPTION

ERRSET is used to specify a *stn* to receive program control upon the occurrence of any BASIC error.

Error branching remains in effect until an **ERRCLR** is executed.

When the **ERRSET** statement is executed, any existing error branching from an **IF ERR**, or **ERRSTM** is reset to branch to the selected *stn* upon occurrence of any error.

ERRSET does not affect the state of the special **EOFSET** branch on end-of-file error.

EXAMPLES

```
ERRSET 8000
```

ERRORS

No such statement number

See also

Error Messages, **EOFSET, ERRCLR, IF ERR, ERRSTM JUMP, ERR, SPC, MSC, MSF**

ERRSTM

SYNOPSIS

Specify statements to execute on an error.

SYNTAX

ERRSTM *any basic stmts*

DESCRIPTION

The **ERRSTM** statement specifies a line of statements to be executed upon the occurrence of any error.

Error statement processing remains in effect until an **ERRCLR** statement is executed.

When the **ERRSTM** statement is executed, any existing error branching from an **IF ERR**, or **ERRSET** is reset to perform the *stmts* following **ERRSTM** upon the occurrence of any error. Normal

execution resumes at the next BASIC line, reserving all *stmts* following **ERRSTM** for when an error occurs.

ERRSTM must be the last statement of a multi-statement line.

ERRSTM has no effect on any special **EOFSET** end-of-file branch in effect.

EXAMPLES

```
ERRSTM PRINT "ERROR OCCURRED AT LINE:";SPC 10

ERRSTM CLOSE \ STOP

ERRSTM IF SPC 8 = 42 STOP ELSE REM
```

ERRORS

Syntax error

See also

Error Messages, **EOFSET, ERRCLR, IF ERR, ERRSET JUMP, ERR, SPC, MSC, MSF**

ESCCLR

SYNOPSIS

Clear any **ESC**ape branching in effect.

SYNTAX

ESCCLR

DESCRIPTION

ESCCLR removes any special **ESC**ape branching or disabling in effect.

Previous **ESC**ape branching or disable set by **ESCSET, ESCSTM** or **ESCDIS** statements is disabled, and normal **ESC**ape termination of a program is resumed.

The **[EOBC]** character may be used to override and abort any program that has **ESC**ape disabled, or an **ESC**ape branch in effect.

EXAMPLES

```
ESCCLR
```

ERRORS

none

See also

ESCSET, ESCDIS, ESCSTM, IF ERR

ESCSET

SYNOPSIS

Enable branch to statement on **ESC**ape.

SYNTAX

ESCSET *stn*

DESCRIPTION

ESCSET specifies a *stn* to receive program control upon pressing of the **ESC**ape key.

Escape branching remains in effect until an **ESCCLR** is executed.

The **[EOBC]** character may be used to override and abort any program that has **ESC**ape processing.

When the **ESCSET** statement is executed, any existing **ESC**ape branching from the **ESCSTM** or **ESCDIS** is reset to branch to the **ESCSTM** *stn* upon the occurrence of an **ESC**ape.

ESCCLR is used to clear automatic branch-on-**ESC**ape and resume normal **ESC**ape processing. Normal **ESC**ape processing terminates the running BASIC program and produces a **STOP at** prompt on the screen:

```
STOP at statement xx;yy in program name
```

Normal **ESC**ape termination does not close all opened data files.

Note that **ESC**ape's function may be assigned to keys other than **ESC**ape itself, just as the **ESC**ape key may be assigned to perform some other function. The **ESC**ape statements described above will act upon any key currently defined as an **[ESC]**.

EXAMPLES

```
ESCSET 8000
```

ERRORS

No such statement number

See also

JUMP, ERR, SPC, MSC, MSF, STOP, Input Character Processing

ESCDIS

SYNOPSIS

Disable **ESC**ape key.

SYNTAX

ESCDIS

DESCRIPTION

The **ESCDIS** statement prevents unauthorized **ESC**ape termination of

any BASIC program. Any pressing of the **ESC**ape key by the user is ignored.

ESCDIS remains in effect until an **ESCSET, ESCSTM** or **ESCCLR** is executed.

When the **ESCDIS** statement is executed, any existing **ESC**ape branching is reset to ignore further **ESC**ape characters.

The **[EOBC]** character may be used to override and abort any program that has **ESC**ape processing.

EXAMPLES

```
ESCDIS
```

ERRORS

none

See also

JUMP, ERR, SPC, MSC, MSF, Input Character Processing

ESCSTM

SYNOPSIS

Specify statements to execute on **ESC**ape.

SYNTAX

ESCSTM *any basic stmts*

DESCRIPTION

The **ESCSTM** statement specifies a line of statements to be executed upon the pressing of an **ESC**ape key.

ESCape statement processing remains in effect until an **ESCCLR** statement is executed.

The **[EOBC]** character may be used to override and abort any program that has **ESC**ape processing.

When the **ESCSTM** statement is executed, any existing **ESC**ape branching from the **ESCSET** or **ESCDIS** is reset to perform the *stmts* following **ESCSTM** upon the occurrence of any error. Normal execution resumes at the next BASIC line, reserving all *stmts* following **ESCSTM** for an **ESC**ape.

ESCSTM must be the last statement of a multi-statement line.

Note that **ESC**ape's function may be assigned to keys other than **ESC**ape itself, just as the ESCape key may be assigned to perform some other function. The **ESC**ape statements described above will act upon any key currently defined as an **[ESC]**.

EXAMPLES

```
ESCSTM PRINT "ESCAPE PRESSED AT LINE";ERR(2)

ESCSTM CLOSE \ STOP

ESCSTM CLOSE \ CHAIN "MAINMENU"
```

ERRORS

Syntax error

See also

JUMP, ERR, SPC, MSC, MSF, Input Character Processing

EXECUTE

SYNOPSIS

Compile and Execute BASIC Statement.

SYNTAX

EXECUTE *str.expr*

DESCRIPTION

The *str.expr* must contain at least one valid BASIC statement without a preceding *stn*. This statement is executed as if it were in the program in place of the **EXECUTE** statement.

Multi-statement lines are not allowed inside the string. Only statements which are allowed in *immediate mode* are available for **EXECUTE**.

EXECUTE itself cannot be included within the **str.expr**. **EXECUTE** is useful where the syntax of a BASIC statement itself must be made variable. For example, allowing a user to enter a numeric expression with **INPUT** and then evaluating such an expression would require considerable program code. **EXECUTE** could be used by constructing an appropriate **LET** statement within a string, and allowing UniBasic to perform the evaluation.

EXAMPLES

```
EXECUTE "LET A = SQR(B7) * 100"
```

ERRORS

All possible encoding and runtime errors.

See also

Each statement and function to be included.

FOR

SYNOPSIS

Looping; repeating a group of statements.

SYNTAX

FOR *num.var* = *initial* **TO** *final* {**STEP** *step* }

DESCRIPTION

The **FOR** statement is used in conjunction with the **NEXT** statement for repetitive statement execution. Statements between the **FOR/NEXT** may be re-executed a given number of iterations. This repetitive execution is known as a *loop*.

The *num.var* is termed the *index* variable and is used to control the *loop*.

initial is any *num.expr* which, after evaluation defines the first value of *num.var* for the *loop*. This value is stored in **num.var** if the *loop* is to be executed (see below).

final is any *num.expr* which, after evaluation is stored as the final value of *num.var* for the *loop*.

The optional *step* is evaluated and used as the increment for each iteration of the *loop*. *num.var* is incremented by this value for each iteration. If no explicit *step* is defined, the default *step* value is 1.

Looping is initiated by setting the *index* variable equal to the *initial* value. At this point, a preliminary check is made to see if the *loop* should be executed at all. If: *initial* > *final* AND *step* > 0, or *initial* < *final* AND *step* < 0, then the *loop* statements are not executed and the program resumes following the associated **NEXT** statement (**NEXT** with same *index* variable). If not, execution continues with the statement following the **FOR**.

Upon execution of the associated **NEXT** statement, the *step* value is added to the *index*. If the new *index* will exceed the *final* value, normal program execution resumes at the statement following the **NEXT** with the *index* variable unchanged for BITS applications, and set to the terminating value for IRIS applications; e.g. if the *step* value is such that the *index* will eventually equal the *final* value, the loop terminates with *index* = *final* for BITS, and *index* = *final*+*step* for IRIS. In IRIS applications, *index* is set to the first value causing the loop to terminate.

A step value of zero will produce an infinite loop.

A complete **FOR/NEXT** loop may be executed on a single line in *immediate mode*:

```
10 FOR I=1 TO 10 \ PRINT I \ NEXT I
```

An error is generated if, in *immediate mode*, a **NEXT** is not included on the line containing **FOR**.

FOR/NEXT loops may be nested if certain precautions are taken. The following is an example of valid nesting:

```
10 FOR A=1 TO 10
```

```
20   FOR B=1 TO 5
30     FOR C=B+1 TO 4*A
40       REM Statements
50     NEXT C
60   NEXT B
70 NEXT A
```

The range of **FOR/NEXT** loops may not overlap. The following is an example of invalid nesting:

```
10 FOR I=1 TO 10
20   FOR J=I+1 TO 20
30     REM Statements
40   NEXT I
50 NEXT J
```

FOR/NEXT statements may be nested to the number of levels defined in the environment variable **FORNEXTNEST** minus 1.

EXAMPLES

```
FOR I=1 TO 3
  REM Statements
NEXT I
```

Initially, I is set to 1, *final* is set to 3 and *step* defaults to 1. Each execution of the **NEXT** first checks if (I+1)>3. When (I+1)>3, execution resumes following the **NEXT** with I=3, if the program is a BITS program, and I=4 if the program is an IRIS program.

```
10 FOR I=10 TO 1 STEP -2
20   REM Statements
30 NEXT I
```

Initially, I is set to 10, *final* is set to 1, and *step* is set to -2. Each execution of the **NEXT** first checks if (I-2)<1. When (I-2)<1, the loop terminates, in this example with I=2. The loop is performed 5 times for I = 10, 8, 6, 4, and 2. A BITS program terminates the loop with I=2, whereas IRIS programs would terminate with I=0.

ERRORS

Nested FOR with the same Index variable

FOR without matching NEXT

Variable not specified

See also

NEXT, Environment Variable **FORNEXTNEST**

GOSUB

SYNOPSIS

Unconditional branch saving return point.

SYNTAX

GOSUB *stn*

DESCRIPTION

The **GOSUB** statement is used in conjunction with the **RETURN** statement to provide internal program subroutines. Commonly executed groups of statements can be used as subroutines to save user space as well as produce a more structured program.

GOSUB, like **GOTO**, performs an unconditional branch to the specified line number. Unlike **GOTO**, however, the statement number performing the **GOSUB** is saved. Upon the execution of a **RETURN** statement, normal execution would resume at the statement following the **GOSUB**. **GOSUB** and **RETURN** are not paired as are **FOR/NEXT**; i.e. any **RETURN** will return to the last **GOSUB** issued.

Subroutines may be nested to the number of level defined in the environment variable **GOSUBNEST** before a **RETURN** must be executed.

Failure to return from all nested levels can cause an error.

See the **RETURN** statement for variations on returning from subroutines.

EXAMPLES

```
GOSUB 1000

GOSUB START_INPUT:
```

ERRORS

Gosubs nested too deep

No such statement number or label

See also

GOSUBNEST, RETURN

GOTO

SYNOPSIS

Unconditional branch to a statement.

SYNTAX

GOTO *stn*

DESCRIPTION

The **GOTO** statement is used to unconditionally branch to another statement within a program and resume normal execution there.

GOTO always transfers control to the first sub-statement on the specified line, and the line must exist. For transfer to any sub-statement on a line, see the **JUMP** statement.

The verb **GOTO** may also be entered as **GO TO** .

A statement that performs a **GOTO** itself may cause an infinite loop terminated only by **ESC**ape, or **ESC**ape Override **[EOBC]**.

In *immediate mode*, **GOTO** is used to specify the next statement to be executed for *single-step* mode or **CONTINUE**.

EXAMPLES

```
GOTO 1000

GOTO BEGIN:
```

ERRORS

No such statement number or label

See also

JUMP, GOSUB

IF

SYNOPSIS

Conditional statement execution.

SYNTAX

IF *relation* {**AND** *relation*}{**OR** *relation*} {**THEN**} *stmt* {**ELSE stmt**}

DESCRIPTION

The **IF** statement tests an arithmetic or string relation and conditionally performs statements based on the relation being true or false. In general, these *relations* are defined as:

num.expr relation num.expr

- or -

str.expr relation str.expr

- or -

num.expr

- or -

str.expr

The *relation* is one of the *Relational Operators* or *Boolean Operators*. If no relation is specified, the statement is interpreted as true if the *num.expr* is non-zero or *str.expr* is a non-null string.

The **IF** statement will test the given *relation* for validity and execute the *stmt* following **THEN** if and only if the *relation* proves true. If the *relation* is not true, the statement is checked for the **ELSE** operator. If found, the *stmt* following the **ELSE** will be executed; otherwise, the program continues normally.

Entry of the **THEN** operator is generally optional.

The *stmt* following **THEN** and/or **ELSE** may be any BASIC statement or a stn alone implying a **GOTO** *stn*. The verb **GOTO** can also be specifically entered, with the same result. Either **THEN** or **GOTO** must be supplied in order to perform a **GOTO**.

In IRIS mode, a false **IF** condition continues execution with the next statement line, instead of with the next *sub-stn*. When an **IF** is true, all remaining statements on the line are executed. An **ELSE** can be used to override this feature. Both of the following examples perform the same function. In the first example, both statements are executed in IRIS mode if the relation A=100 is true. If false, execution resumes on the next line of statements.

The second example performs a **GOTO** the next statement if the reverse relation is true, otherwise the **ELSE** is executed following with the remaining statements on the line:

```
IF A=100 GOSUB 1000 \ GOTO 1000

IF A<>100 GOTO 120 ELSE GOSUB 1000 \ GOTO 1000
```

In BITS mode, a false **IF** condition continues execution with the next *sub-stmt* (if any), or proceeds to the next statement line. Statements following both the **THEN** and **ELSE** operators must be single statements only; any backslash code \ is considered the end of the **IF** statement. In order to execute a series of statements based on a relation, the relation's logic must be reversed and the true condition made to branch to the next statement after the series.

```
IF A=100 GOSUB 1000 ELSE IF B=100 STOP ELSE 200

IF A<>100 GOTO 110 \ GOSUB 1000 \ GOSUB 2000
```

A blocked-**IF** structure provides a more convenient method of executing several statements for both the true and false conditions for both IRIS and BITS applications. The general form of a blocked-**IF** is:

IF *relation* {**AND** *relation*}{**OR** *relation*} {**REM** Comment}

{**THEN**} {**REM** This is a comment}

stmts to be executed on a single line or multiple lines

{**ELSE** {**REM** This is a comment}

stmts to be executed on a single line or multiple lines}

ENDIF {**REM** This is a comment}

Blocked-**IF** statements are assumed whenever an **IF** statement ends following a *relation*. No *stmts* may follow the *relation* excepting an optional **REM**.

Inclusion of an **ELSE** block is optional. The **THEN** statement is completely ignored and can be omitted, if desired. **THEN**, **ELSE**, and **ENDIF** must be the only statements on their line (except that they may be followed by a trailing **REM** comment).

Statements to be executed on the *relation* being true follow the **IF** (or **THEN**) on subsequent lines. All statements up to the associated **ELSE** or **ENDIF** are part of the true condition.

ELSE defines an optional block of *stmts* to execute when the corresponding Blocked-**IF** was false.

ENDIF defines the end of a blocked **IF**.

Blocked-**IF**s can be nested to any level, and are indented like **FOR-NEXT** loops for readability. There must be an **ENDIF** for every blocked-**IF** in the program. The integrity of the blocked-**IF**s is checked by the **RUN, CHAIN, SAVE, VERIFY** and **CHECK** commands. Once checked, a program is flagged OK eliminating further verification until a statement is changed within a program.

EXAMPLES

```
IF A*5 > B*10 THEN GOSUB 200

IF LEN(A USING A$ TO ".") >132 PRINT #3;

IF A-5 THEN 340 ELSE IF J=100 GOSUB 100 ELSE STOP

IF C$[1,1]<=Z$[10,10] AND C$<>"X" THEN 280

IF (J=10 OR C=20) AND (T=10 OR F=12) STOP

Blocked-IF:

IF (A=100 AND B=200) OR (C=200 AND D=300)
  GOSUB 1000 \ PRINT T
  IF J
    GOSUB 2200 \ WRITE #3,R;A$
  ELSE
    GOSUB 2200 \ READ #3,R;A$
  ENDIF
ENDIF
```

ERRORS

Arithmetic Overflow

IFs without 'ENDIF'

'ELSE' without 'IF'

'ENDIF' without 'IF'

See also

Operators and Expressions, Boolean operators AND OR, Arithmetic Operators, String Operator USING, Concatenation Operators, Unary Operators, String Operator TO, CRT Mnemonics and Expressions, Numeric and String Expressions

IF ERR

SYNOPSIS

Specify statements to execute on an error.

SYNTAX

IF ERR *error mode expr* {*stmt*}

DESCRIPTION

IF ERR 0 is used to specify a line of statements to be executed upon the occurrence of any error. **IF ERR 1** may also be used to specify an error branch, however a separate error number is not reserved for CTRL+C.

When an **IF ERR 0** statement is executed, any existing error branching from a previous **IF ERR 0** , **ERRSET**, or **ERRSTM** is reset to the *stmts* following the **IF ERR 0**. Normal execution resumes at the next BASIC line, reserving all *stmts* following **IF ERR 0** for error processing.

ESCape is also trapped generating a special Error code to the application.

ESCSTM, ESCSET, EOFSET, and **ESCDIS** statements can be used in addition to **IF ERR**.

In *immediate mode*, **IF ERR** can only be used to clear an existing error branch. Attempting to set a new branch results in an error.

Error statement processing remains in effect until an **ERRCLR** or **IF ERR 0** statement is executed without any trailing *stmt*.

IF ERR statements must be the last statement of a multi-statement line.

EXAMPLES

```
IF ERR 0 GOSUB 1000

IF ERR 0
```

ERRORS

Syntax error

See also

Error Messages, **JUMP, ERR, SPC(8),SPC(10), MSC, MSF,**

ERRSTM, ERRCLR

INDEX

SYNOPSIS

Indexed File maintenance statement.

SYNTAX

INDEX *#channel* ; *mode*, *index*, *key*, *record*, *status*

DESCRIPTION

The *channel* is any *num.expr* which, when truncated to an integer, specifies an opened *channel* currently linked to a UniBasic Indexed Data file.

The *mode* is any *num.expr* which, when truncated to an integer, specifies a mode of operation for the **INDEX** statement. For a detailed list of *mode* operations, See also: Indexed Data Files.

The *index* is any *num.expr* which, when truncated to an integer, specifies to which Index or Directory (list of keys) the operation is being directed.

The *key* is any **DIM**ensioned *str.var* which must be **DIM**ensioned to at least the size of the Key for the specified Index.

The *record* is any *num.var* and contains (or returns) a value for the statement *mode*.

The *status* is any *num.var* used to return a status value to the program. Refer to the following pages for a list of *status* values and their meanings.

Parameters may be separated by either a comma or semicolon terminator.

EXAMPLES

```
INDEX #5;4,1,K$,R1,E \ IF E GOTO 1000

E=3 \ INDEX #J,1,0,K$,R1,E \ IF E GOTO 1000
```

ERRORS

Selected channel is not open

Illegal parameter or syntax for command

Selected data record is locked

File is not indexed or mapped

Illegal or nonexistent index number selected

Index selected is not yet initialized

Indexed file structure error or svar dim length < Key length

See also

Indexed Data Files, **SEARCH #**

Summary of INDEX Modes

Mode	Operation
0	Define and Create indices within a Contiguous Data File.
1	Return miscellaneous index information.
2	Search for an exact key.
3	Search for the next highest key.
4	Insert a new key into an index.
5	Delete an existing key from an index.
6	Search for the previous key (Search Backward).
7	Unused, included for compatibility.
8	Maintain the B-Tree insertion algorithm for an index.
9	Temporarily same as Mode 6 - Reserved for future use.

Detailed Table of INDEX Modes

Mode	Index	Status	Operation Performed
0	1<d<63		For a new Indexed File, sets the *key* length of the selected *index* to the number of bytes specified by *record.var*. The maximum key length is 122 bytes. Indices must be defined starting at one and proceed sequentially.
0	0		Freeze the file definition and build the ISAM portion of the file. Total number of initial data records is specified by the *record.var*.
1	>0		Return the key length of the specified *index* in bytes.
1	0	=0	Returns the record number of the First Real Data Record. Normally zero unless the file was built using BUILDXF or copied from an IRIS or BITS system.
1	0	=1	Return the number of Available Records in the file. This value is either the value of the environment variable AVAILREC, or the value based upon the files current size.
1	0	=2	Allocate and return a new record for the application.
1	0	=3	Return a record to the file that is no longer needed. Deleted records will be re-used before the file is extended.

1	0	=4	Return in *record.var* the number of records in the file. IRIS Applications only; Error for BITS applications.
1	0	=5	Return in *record.var* the number of records in the file. For BITS applications, performs the same operation as 4 above.
1	0	=6	Set the First Real Data Record to the value supplied in *record.var*. This option is only available during file structuring.
1	0	=7	Return the current number of records in use (allocated) in the data portion of the file.
2			Search the specified *index* for the exact match of the supplied *key.var*. If found, return the full key in the supplied *key* variable, and the associated record number in *record.var*. The *status.var* is set to 0 if the *key* was found, and 1 if the *key.var* was not in the *index*.
3			Search the specified index for the first *key* whose value logically exceeds the supplied **key.var**. If found, *status.var* is set to 0, the full key is returned in *key.var*, and the associated record number is returned in *record.var*.
4			Insert *key.var* into the specified *index* using the supplied *record.var* as the associated pointer. The record should have been previously allocated using *mode* 1, *status* = 2 above. A *status.var* of 0 indicates a successful operation. If the *key.var* already exists in the *index*, a 1 is returned as *status.var*.
5			Delete the supplied *key.var* from the specified *index*. If successful, *record.var* is returned as the associated pointer, and the *status.var* is set to 0. A *status.var* of 1 indicates an unsuccessful operation; i.e., the *key.var* was not found in the index. The record should be returned to the file using *mode* 1, *status* = 3 above.
6			Search the specified *index* for the first key whose value is logically less than the supplied *key.var*. If found, *status.var* is set to 0, the full key is returned in key.var, and the associated record number is returned in *record.var*.
7			No operation. Reserved for future use.
8			B-Tree algorithm maintenance. If *record.var* is negative, return in *record.var* the current B-Tree algorithm for *index*. If *record.var* is positive, change the insertion algorithm to the value passed in *record.var*. Set to zero (default) for random insertion, 1 for increasing insertion, 2 for decreasing insertions.
9			Temporarily, the same as Mode 6. Reserved for future

use.

Table of INDEX status return values

Value	Description of Status
0	No error, the Index operation was successful.
1	Operation was unsuccessful; i.e. *key* not found.
2	End of *index*. Given on *modes* 3, 6 and 9 when the beginning or end of the *index* is reached.
3	End of data; all records are allocated. This error is only generated when the environment variable PREALLOCATE is defined to limit the number of data records.
4	File has no Indices, cannot perform an Indexed File operation.
5	Indexed file structure error; given when *key* length **DIM** is less than the actual size of the key from an Index on Modes 2, 3, 6 and 9. Indicates a **DIM**ension error or structure problem, possibly a c-tree file structuring error. Printing the value of ERR(8) will provide a more concise description of the error.
6	Index number not in sequence during creation. You must sequentially define all directories.
7	File is not a Contiguous File.
8	File is already Indexed.
9	Value of *record* is negative or too large.
10	Illegal Index Number. Must be between 1 and 62.

INPUT

SYNOPSIS

Retrieve keyboard or channel input.

SYNTAX

INPUT {*#chn.expr*;} {*crt.expr*;} {*control*} {"*prompt*"} *var.list*

DESCRIPTION

The **INPUT** statement assigns values to variables. The values are accepted from either keyboard (operator) input, or through a channel (file or device).

If a *chn.expr* is specified, the standard input for this statement will be satisfied from the selected *channel*, *record* and *byte-displacement*. If the running program is a BITS program and *chn.expr* is not specified (or the selected *channel* is not open), input will be taken from the keyboard. When requesting input from a *chn.expr*, the *crt.expr*,

control, and "*prompt*" options should not be used.

If a *crt.expr* is specified, it is evaluated and output based upon the information from current active *term* file. If no *term* file is active, the *crt.expr* is ignored. If the *crt.expr* contains undefined mnemonics, those undefined will be ignored. Typically, a *crt.expr* is used to position the cursor on the screen and/or clear lines, etc. prior to the request for input. Use of a *crt.expr* will suppress the normal prompt unless a specific "*prompt*" is specified.

If a "*prompt*" is specified, the default prompt-message **?** is replaced by the literal text within quotes. A null prompt "" suppresses the output of the prompt-message as does the inclusion of any *crt.expr*.

If a *control* is specified, the input is restricted by a character count, length of time, or both. A special *control* is provided to read the contents of the terminal's input buffer and is used by programs to read parameters entered on a command line. Two different mechanisms exist to invoke *control* features.

(***mode.expr, num.var***)	***control with a returned response***

The *mode.expr* is evaluated and truncated to an integer. The second parameter must be a *num.var* and will be set following the **INPUT** as the response.

If the *mode.expr* evaluates to zero, the entire contents of the input buffer is selected as the standard input. The *num.var* is not set to any value in this mode. Typically, this *mode* is used within a program that can accept its input from a command line. To read the last command line, the input must be performed prior to any other **INPUT** or **PRINT** statements which corrupt the input buffer. Programs such as: **PORT**(u), **LIBR**(u), **QUERY**(u), etc. use this mode to read the command line into a string variable and then parse off the parameters.

If the *mode.expr* evaluates to a positive value, the program is suspended for that number of tenth-seconds or until the **[EOL]** character is entered terminating the input. The maximum wait time is 65535 tenth-seconds, or approximately 109 minutes. The actual number of tenth-seconds that were spent waiting for **INPUT** is returned as a positive value in *num.var*. If no **[EOL]** character (return) is received within the specified interval, the num.var is set to the negative of the specified tenth-second wait interval and any input characters are passed to the **INPUT** *var.list*.

If the *mode.expr* evaluates to a negative value, the value is converted to a positive number selecting the maximum number of characters to be accepted for input. -5 causes the system to wait for the input of 5 characters. The actual number of input characters is returned in the *num.var*. The **[EOL]** character may be used to terminate a character limited input prior to exhausting the specified character count.

LEN / TIM	***control* with SIGNAL response**
LEN *num.expr*;	Set *num.var* as the character limit

TIM *num.expr*; Wait *num.var* tenth-seconds for input

The *num.expr* is evaluated and truncated to an integer. A semicolon must terminate the *num.expr*, or an error will occur.

If a **LEN** *num.expr*; is specified, the *num.expr* is evaluated, truncated to an integer and set as the maximum number of characters to be accepted for input. The **[EOL]** character may be used to terminate a character limited input prior to exhausting the specified character count.

If a **TIM** *num.expr*; is specified, the *num.expr* is evaluated, truncated to an integer and set as the number of tenth-seconds to wait for input. The maximum wait time is 65535 tenth-seconds, or approximately 109 minutes. If no input is seen within the specified interval, a system **SIGNAL** is sent to the program with the actual number of characters entered. See **SIGNAL 5** to clear the message queue.

Both a **TIM** *num.expr*; and **LEN** *num.expr*; can be specified on the same **INPUT** statement.

GENERAL OPERATION OF DATA INPUT

Following the parsing of the optional parameters, the program is suspended while data is read from the standard input; usually the terminal. Characters previously entered (and buffered) are processed first.

Characters are echoed (for keyboard input) unless echo is disabled by the previous entry of the **[ECHO]** toggle character (normally CTRL+E), a **SYSTEM 9** statement or the **$ECHO CALL**.

If the **INPUT** is not satisfied, the program is suspended until the **[EOL]** character (return) is entered, the specified character limit is reached, or a time-out occurs on timed input. When any of these conditions occurs, the program resumes operation and begins processing input into the variables defined in the *var.list*. The **[ESC]** or **[EOBC]** characters will terminate input and abort the statement.

SYSTEM 26 and **27** alter the operation of character limited input. Normal operation is to automatically resume execution of the program when the limiting number of characters have been processed. Executing a **SYSTEM 27** forces character limited **INPUT** to require entry of the **[EOL]** character (return). When the limit is reached, the terminal's bell is sounded and extra characters (except BACKSPACE or CTRL+X) are ignored. **SYSTEM 26** resets character limited input to operate normally, that is, resume execution when the limiting number of characters have been processed.

No special processing is performed on the characters received. Data is passed to the program exactly as received from the operating system. If you are attempting to connect a system directly to IRIS or BITS, you will need to configure the system to strip the high-order bit using the UNIX **stty** command, or use **CALL 60,3,str.var** to toggle the IRIS/BITS data into standard format.

When binary input **(SYSTEM 14)** or **IOBI** is enabled, all characters

are passed directly to the program excluding CTRL+S (XOFF) and CTRL+Q (XON). All character input processing for **[EOL]**, **[ESC]**, **[BACKSPACE]**, etc. is suspended and the program must process all input data.

WARNING: **When using Binary Input, it is possible to lock the terminal if your program does not provide a way to terminate itself. If you lock a terminal, use another *UniBasic* port to HALT or PORT EVICT(u) the locked program.**

When a *str.var* is specified in the *var.list*, all characters are copied up to, but not including the **[EOL]** character. If the input is larger than the specified *str.var*, the extra input characters are discarded. If the input does not fill up the destination *str.var*, a zero-byte terminator is placed after the last character of data.

If a *num.var* is specified in the *var.list*, the input characters are converted to numeric and stored into the *num.var*. An error is generated if the input is not numeric or contains characters other than digits + - . or **E notation**. If error branching is in effect, the **MSC(1)** function (Last INPUT Element) may be used to determine which input item was in error. For example:

```
10 ERRSET 40

20 INPUT A,B,C,D

30 END

40 PRINT "ERROR IN INPUT VARIABLE";MSC(1)
```

The user would enter the item or items, separating multiple items with a comma "," or **[EOL]**. If too many items are entered, a non-abortive error is generated and the extra items are ignored.

If a non-numeric value is entered for a numeric variable, the message **\Invalid numeric\?** is displayed and the entire input must be re-entered. Numeric values may be entered in scientific notation; however, commas are not allowed within a numeric item; e.g. 1,200 must be entered as 1200. To abort the **INPUT** statement, press **ESC**ape.

BITS applications may use the **INPUT** statement with a *channel*. **INPUT #** is similar to terminal **INPUT**, however, the above mentioned CRT, prompt, and limit features are unavailable. The **INPUT #** statement reads one line of data into the terminal's buffer. Processing proceeds as with normal **INPUT**. IRIS users read sequential data using the standard **READ** statement. For example:

```
100 INPUT #4;D,E,F,G
```

To successfully load these variables, the input source (file or device) would have to contain 4 numeric literals in the form:

number , number , number , number

INPUT # terminates upon the transfer of a new-line, form-feed, zero byte, or end of input buffer. If **INPUT #** terminates on a form-feed, it will be the last character of data. New-line may not be transferred as data. Carriage returns are ignored when transferred.

Normally, the optional *record expr* and *byte expr* specifications are not used, as line-oriented data is generally of variable length. Each successive **INPUT #** starts its transfer immediately after the previous one has been completed.

EXAMPLES

```
INPUT TIM 10; LEN 30; "CUSTOMER NAME >"A$

INPUT @10,23;"Press [RETURN]" T$

INPUT (-1,K) "Enter a single character "A$

INPUT "4 numbers w/ comma ? "A,B,C,D
```

ERRORS

INPUT of wrong type or insufficient

Syntax error

User partition space exhausted

See also

SYSTEM, $TERM, Input Character Processing, **CALL $ECHO**

INTCLR

SYNOPSIS

Clear program interrupt branch.

SYNTAX

INTCLR

DESCRIPTION

INTCLR restores normal operation with respect to user interrupts. CTRL+C, **SIGNAL 1**, and **SEND** no longer automatically interrupt the program and branch to a specific **INTSET** statement number.

EXAMPLES

```
INTCLR
```

ERRORS

None

See also

INTSET, SIGNAL, [INTR], SEND

INTSET

SYNOPSIS

Define a branch for program interrupts.

SYNTAX

INTSET *stn*

DESCRIPTION

INTSET sets the selected *stn* to receive control each time an interrupt character is pressed or a message is waiting to be received. CTRL+C is normally defined as an interrupt character, **[INTR]**, but may be changed within the term file. **INTCLR** removes the branching, and further interrupt requests or messages are ignored.

A program branch is defined to transfer execution to a pre-defined statement when either an 'interrupt' character is pressed or a message is transmitted to your port via the **SEND** or **SIGNAL** statements.

The interrupt handling routine can do any processing desired and return to the main program as if the branch never occurred. Secondary interrupts are inhibited until the program clears the initial interrupt. This is done using the **ERR(3)** function, which also yields the original interrupted statement number. Generally, an interrupt handling routine loops until all interrupts or messages are received. The main body of the program is resumed using the statement:

```
stn JUMP ERR(3)
```

or

```
stn JUMP ERR(3);ERR(7)
```

The latter form is required if multi-statement lines are used within the program.

The interrupt function should not use the **ERR(3)** function other than shown above unless it is re-entrant and stacks multiple return locations.

EXAMPLES

```
INTSET 1000 ! Branch on Signal, CTRL+C
```

ERRORS

No such statement number

See also

SEND, [INTR], SIGNAL, SEND

JUMP

SYNOPSIS

Computed **GOTO** unconditional branch to stn.

SYNTAX

JUMP *stn* {; *sub-stn*} {, *num.var*}

DESCRIPTION

The *stn* is any *num.expr* which, after evaluation is truncated to an integer and used as the statement number to branch to. The optional *sub-stn* is any *num.expr* which, after evaluation is truncated to an integer and used as the sub-statement on that line. **JUMP** performs an unconditional branch to the selected statement (and sub-statement). On multi-statement lines, sub-statements are numbered starting at 1.

If the optional *num.var* is supplied, it will be set to the statement number of line following **JUMP**. This is similar to the **GOSUB** statement, as a subsequent **JUMP** to this variable will essentially perform a **RETURN**. The *num.var* will is set to zero when the **JUMP** is the last statement of a program.

JUMP statements are in no way affected by the **RENUMB** command. Therefore, they are not an acceptable substitute for **GOTO** or **GOSUB** when a literal *stn* can be used.

JUMP is best used in conjunction with system functions that supply statement numbers, retaining the program's ability to be renumbered.

EXAMPLES

```
JUMP K*10

JUMP SPC(10)

JUMP ERR(1);ERR(4),J
```

ERRORS

No such statement number

See also

GOTO, GOSUB, SPC, ERR, INTSET, ESCSET, ERRSET

KILL

SYNOPSIS

Delete a data or program file.

SYNTAX

KILL *filename.expr ...*

DESCRIPTION

The *filename.expr* may contain a single *filename* or list of *filenames* to be deleted. Multiple strings may each contain a single *filename* or

group of *filenames* separated by spaces.

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.

An application may delete a program or data file currently in use or opened by itself or another user. The effect is to remove the entry of the *filename* from the system directory preventing it from being opened again. When the last user closes the file, the system releases the disk space. Prior to closing, all types of access, including extending the file is permitted.

EXAMPLES

```
KILL "23/ABC 23/DEF"

KILL A$,B$,C$
```

ERRORS

File does not exist

Read Protected File

Write Protected File

See also

DELETE, MFDEL(u), KILL(u), `filename`

LET

SYNOPSIS

Assign values to numeric & string variables.

SYNTAX

{**LET**} *var* = *expr* {(, | ;) } ...

{**LET**} *str.var* = *num.expr* **USING** *str.expr* {, *num.expr* ...}

{**LET**} *str.var* = *str.var* **TO** *str.expr* { (; | :) *num.var*}

DESCRIPTION

var is any *num.var, str.var, array.var* or *mat.var* to be assigned a value.

expr is any expression whose result matches the type of the supplied *var*.

A numeric variable *num.var*, *array.var*, or *mat.var* may be assigned any numeric expression or numeric string whereas string variables *str.var* may be assigned any string expression or numeric value. The functions **STR** and **VAL** may be used to convert a numeric result to string or vice versa to match the destination *var*.

The **LET** verb is optional, and is assumed when not entered.

Although entry of the **LET** verb is optional, it is printed whenever the program is listed.

Multiple assignments may appear on a single line separated by commas for BITS and semicolons for IRIS programs.

```
Z=100;Q=1;N=0;A$="TXXX" ! IRIS Applications

Z=100,Q=1,N=0,A$="TXXX" ! BITS Applications
```

Numeric formatting is performed within a **LET** statement with the **USING** operator. This is functionally equivalent to the **EDIT** statement.

```
LET D$=X USING "##,###.##"

LET E$=X USING "##,###.##",Y,Z
```

The **TO** operator allows assignment of string data to terminate upon encountering a given *str.expr*. The *str.expr* may be a single or multiple character string. The optional **num.var** returns the character position at which assignment stopped.

```
LET N$="ABCDEF%GHIJKL"

LET S$=N$ TO "%":K ! IRIS Applications

LET S$=N$ TO "%";K ! BITS Applications

returns: S$="ABCDEF",K=7
```

When using the advanced forms for multiple assignments, choose the following delimiting characters to match the operating system **BASICMODE** and/or program type currently loaded:

Character function	IRIS	BITS
String concatenation	,	+
Multi-assignment	;	,
Terminator for TO	:	;

EXAMPLES

```
LET V=1

LET T$=1/3

LET A=42;T=17;R7=91 !IRIS MULTI-LET

LET B[7]=(A*T)+(R7/4) USING "#####"

LET A$="TOTAL=",D[7]=A/T! BITS MULTI-LET

LET A$="1234565";T=A$;B$=A$ TO "45":T1
```

ERRORS

Syntax Error

String Expression not allowed here

String Expression must be used here

See also

TO Operator, **USING** Operator, Numeric and String Expressions, **STR, VAL**

LIB

SYNOPSIS

Specify a library logical unit for callable subprograms.

SYNTAX

LIB *expr*

DESCRIPTION

expr is any numeric expression to represent a numbered IRIS style logical unit number or string expression representing a unix directory name.

A value of -1 may be used to clear a defined library logical unit.

The library unit is the first unit searched by **CALL** for a subprogram file, unless the subprogram filename itself specifies a logical pathname.

SPC 23 is used to determine the current library logical unit, however its return value is only valid when the library logical unit is numbered.

EXAMPLES

```
LIB 1

LIB "pgms"
```

ERRORS

None

See also

CALL, ENTER, SPC23

MAT =

SYNOPSIS

Copy an entire matrix.

SYNTAX

MAT *dest mat.var = source mat.var*

DESCRIPTION

mat.var is any numeric matrix variable name.

The *dest mat.var* must be at least as large as the *source mat.var*. In the following example, matrix A is dimensioned as [5,5] and matrix B as [6,6]:

```
MAT B=A     is acceptable.

MAT A=B     Is illegal since A is not large enough to contain
            all of the elements in B.
```

The copy is performed element by element. An error or integer truncation can occur if the precisions are not compatible. Row and column zero are not copied. **MAT =** cannot be used to copy single element arrays.

EXAMPLES

```
MAT T=D0

MAT T[4,4] = D9

MAT T[5]=G
```

ERRORS

Syntax error

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT +

SYNOPSIS

Add elements from two matrices.

SYNTAX

MAT *dest mat.var = source mat.var1 + source mat.var2*

DESCRIPTION

mat.var is any numeric matrix variable name.

The two matrices being added must be exactly the same dimensions (rows and columns). The *dest mat.var*, if not already defined, is dimensioned at the current default precision for the same number of rows and columns as the *source mat.var* . An error or integer truncation can occur if the precisions are not compatible. Row and

column zero are not added.

The same *matrix* variable may appear on both sides of the equation

The sum, matrix D, of matrix A and matrix B is:

```
D[X,Y]=A[X,Y]+B[X,Y]
```

for each matrix element.

EXAMPLES

```
MAT T=D0+A9

MAT D0=D0+J
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT *

SYNOPSIS

Multiply elements of two matrices.

SYNTAX

MAT *dest mat.var = source mat.var1 * source mat.var2*

MAT *dest mat.var* = (*constant expr*) * *source mat.var*

DESCRIPTION

mat.var is any numeric matrix variable name.

MAT * performs a multiplication, establishing a new matrix equal to the product of two matrices. Scalar multiplication allows each element of a matrix to be multiplied by a constant.

Following the rules of matrix multiplication, if we multiply matrix A dimensioned [X,Y] by matrix B dimensioned [R,S], then the resulting matrix will be dimensioned [X,S]. An error or integer truncation can occur if the two precisions are not compatible. Row and column zero elements are not multiplied.

The same matrix variable may not appear on both sides of the equation.

Scalar multiplication causes each element of the given matrix to be multiplied by the value of the *constant expr*. The *constant expr.* must be in parentheses, and immediately follow the equal sign (=).

EXAMPLES

```
MAT D=A*B

MAT Q=X*X

MAT C=(5)*A
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT CON

SYNOPSIS

Establish a constant matrix.

SYNTAX

MAT *dest array.var* = **CON** { [*subscript1* {, *subscript2* }] }

DESCRIPTION

array.var is any numeric *array.var* or *mat.var name*.

Each element of the selected *array.var* or *mat.var* is set to the constant value one. Row and column zero are not set.

The optional *subscript1* and *subscript2* are evaluated, truncated to integer and used to select a new working size. The total number of elements in the new size cannot exceed that of the old. A single element *array* can be converted to a *matrix* or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **CON[24]**.

A constant other than one can be accomplished using a combination

of the **CON** function and Scalar multiplication:

```
MAT A=CON \ MAT B=(5)*A \!Fill B with 5's.
```

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT A=CON

MAT D0=CON[7,X/2]
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT IDN

SYNOPSIS

Establish an identity matrix.

SYNTAX

MAT *dest array.var* = **IDN** { [*subscript1* {, *subscript2* }] }

DESCRIPTION

array.var is any numeric *array.var* or *mat.var* name.

The matrix function **IDN** establishes an identity matrix of all zeroes with a diagonal of ones.

Any matrix multiplied by an identity matrix of the same size results in the original matrix. For example: If matrix A is dimensioned [3,3] and matrix B is an identity matrix also dimensioned [3,3], the result of: MAT C=A*B produces matrix C equal to A. Row and column zero are not affected by **IDN**.

The optional *subscript1* and *subscript2* are evaluated, truncated to integer and used to select a new working size for the array. The total number of elements in the new size cannot exceed that of the old. A single element *array* can be converted to a *matrix* or vice versa as

long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **IDN[24]**. An identity *array* is an array of all zeros.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT Q=IDN

MAT T=IDN[4,4]

MAT A8=IDN[X,Y]
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT INV

SYNOPSIS

Invert a matrix.

SYNTAX

MAT *dest mat.var* = **INV**(*source mat.var*)

DESCRIPTION

mat.var is any numeric 'square' matrix variable name.

The matrix function **INV** establishes one square matrix as the inverse of another.

Only square matrices (number of rows = number of columns) may be inverted. Both matrices must also be the same precision and dimension. Row and column zero are not affected by **INV**.

The **DET** function supplies the determinant of the last matrix inverted by your program, e.g. if two matrices are inverted before the **DET** function is used, the determinant returned will be from the second

inversion.

Since numeric precision in UniBasic is accurate only to 20 significant digits, matrix elements will be rounded accordingly.

EXAMPLES

```
MAT C=INV(A)
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT TRN

SYNOPSIS

Transpose a matrix.

SYNTAX

MAT *dest mat.var* = **TRN**(*source mat.var*)

DESCRIPTION

mat.var is any numeric matrix variable name.

The matrix function **TRN** is used to establish one matrix as the transposition of another.

Transposition causes each element [X,Y] of the original matrix to be moved to element [Y,X] of the transposed matrix. Note that this also causes the dimension of the transposed matrix to be the reverse of the original. For example:

Original matrix [3,4]				**Transposed matrix [4,3]**		
1	2	3	4	1	5	9
5	6	7	8	2	6	10
9	10	11	12	3	7	11
				4	8	12

An error or integer truncation can occur if the two matrix precisions are not compatible. Row and column zero are not affected by **TRN**.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT C=TRN(A)
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT ZER

SYNOPSIS

Zero an entire matrix.

SYNTAX

MAT *array.var* = **ZER** { [*subscript1* {, *subscript2* }] }

DESCRIPTION

mat.var is any numeric *array.var* or *mat.var* name.

The matrix function **ZER** allows each element of a matrix to be set to zero. Row and column zero are not set.

The optional *subscript1* and *subscript2* are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old. A single element *array* can be converted to a matrix or vice versa as long as the total number of elements does not exceed the original DIMensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **ZER[24]**.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT C=ZER
```

```
MAT R7=ZER[4,4]
```

ERRORS

Syntax error

Same matrix on both sides of MAT is illegal here

Matrix has zero DIMension

Matrix DIMensions are not compatible for this operation

Variable not specified

Matrices have different DIMensions

Variable name not DIMensionable

See also

Numeric, Array and Matrix Variables

MAT INPUT

SYNOPSIS

Assign keyboard/file input to a Matrix.

SYNTAX

MAT INPUT {*#chn.expr*;} *array.var*{**[***subscript1* {, *subscript2* }**]**} ...

DESCRIPTION

The optional *#chn.expr* is any channel expression used to specify an input device or text file for the operation. Only BITS applications may perform **MAT INPUT #**.

The *array.var* is any *mat.var* or *array.var* with or without subscripts.

MAT INPUT is used to assign values to an entire matrix. The values are accepted from either keyboard (operator) input, or through a channel (file or device for BITS applications).

The optional *subscript1* and *subscript2* are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old. A single element *array* can be converted to a *matrix* or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **array[24]**.

Execution of a **MAT INPUT** statement pauses the program after output of a **?** to your terminal. The program is then suspended and data input is accepted. The user would enter all matrix items, separating each item with either a comma , or **[EOL]** (return). **MAT INPUT** does not complete until all elements have been accepted.

The *array* elements are assigned by rows, starting with [1,1] thru [1,n], then continuing with [2,1] thru [2,n], etc. Row and column zero

are not assigned. For example, a 4 by 4 matrix might be entered as:

```
17,42,87,12 <-

18,14,26,14 <-

15,0,18,29 <-

34,29,86,69 <-
```

Using **MAT INPUT** from a channel is similar to terminal **MAT INPUT**, except the data is read from the channel and must include row and column zero elements. The data must be separated by either commas or **[EOL]** (return), and cannot be in the format generated by a **MAT PRINT #**. Only a BITS application may perform **MAT INPUT #**.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT INPUT T

MAT INPUT A,B[4,10],C

MAT INPUT #3;X

MAT INPUT #2,R,20;E1,E2
```

ERRORS

Syntax error

Matrix has zero DIMension

Variable not specified

See also

MAT PRINT, Numeric, Array and Matrix Variables, Channel Expression

MAT PRINT

SYNOPSIS

Print contents of an array or matrix.

SYNTAX

MAT PRINT { *#chn.expr*; } *array.var* { ; }...

DESCRIPTION

The optional *#chn.expr* is any legal channel expression re-directing the output to a file or device.

Each *array.var* is any *array.var* or *mat.var* to be printed in ASCII form without subscripts. Each variable may be followed by either a

comma (,) or a semicolon (;). A comma will cause the matrix variable preceding it to be spaced using comma fields. These are generally 20 characters long, but can be changed by setting the environment variable **TABFIELD**. A semicolon will cause minimal spacing between elements. Elements are normally preceded by a space or "-", indicating negative or positive, and will be followed by one space. When all items in a matrix row have been output, two blank lines are output to produce double spacing between rows.

Row and column zero elements are only printed for **MAT PRINT** in *immediate mode* and when the data is directed through a *channel*.

If a *channel* is specified to **MAT PRINT**, output is attempted to that channel. If the selected channel is not open, output is sent to the terminal.

EXAMPLES

```
MAT PRINT A

MAT PRINT I,J

MAT PRINT X;Y;Z;

MAT PRINT #3,T;H1,S1
```

ERRORS

Syntax error

Matrix has zero DIMension

Variable not specified

See also

Numeric, Array and Matrix Variables, Channel Expression.

MAT RDLOCK

SYNOPSIS

Read an array, matrix or string with locking.

SYNTAX

MAT RDLOCK *#chn.expr*; *var.list...*

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to read data.

MAT RDLOCK # transfers data into any *var*, *mat.var*, *array.var* or *str.var*. The operation is similar to a **READ #** statement, except that an entire *array* or *matrix* is transferred; including row and column zero elements. If the specified *var* is a string, its entire specified length is transferred including zero-byte terminators.

If the variable in the list is an *array.var*, an optional *subscript1* and

subscript2 may be specified. If given, these are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old size. A single element *array* can be converted to a matrix or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **array[24]**.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by subscripts. The transfer size is rounded up to an even number of bytes. Either an even or odd subscript may be specified. The address for the start of the transfer within the *str.var* is not changed. If an odd number of bytes is specified in the subscript (such as *svar[2,2]*), the size is rounded up to an even number of bytes resulting in the transfer of 2 bytes into *svar[2,3]* in this example. All characters are transferred including zero-bytes.

If the application is an IRIS program, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.

MAT RDLOCK transfers data and unconditionally locks the record.. The data record remains locked until a non-locking operation is performed by that same program to the same channel. While a record is locked, other users will be unable to access the record. **MAT RDLOCK#** is identical to **MAT READ#** omitting the trailing semicolon. See the **MAT READ#** statement for details on the transfer of data to different types of files.

EXAMPLES

```
MAT RDLOCK #3,R1,100;A

MAT RDLOCK #C,R;A$
```

ERRORS

Syntax error

Selected Channel is not OPEN

See also

Numeric, Array and Matrix Variables, Channel Expression, **MAT READ**, Numeric Variable Precision

MAT READ

SYNOPSIS

Read an array, matrix, or string from DATA.

SYNTAX

MAT READ *array.var*{[*subscript1* {, *subscript2* }]}...

MAT READ *str.var* ...

DESCRIPTION

The *array.var* is any numeric *array.var* or *mat.var*. A *str.var* is any string variable name. You may mix *array.vars* and *str.vars* in a single **MAT READ** statement.

MAT READ attempts to transfer data into each *array.var*, *mat.var* or *str.var* listed in the statement. Transfer of each *array.var* or *mat.var* element terminates at a comma (,) or at the end of the **DATA** statement. The format of the data is left to the user. Attempting to read string data into a numeric variable produces the error **DATA** of wrong type (numeric/string).

MAT READ transfers data sequentially from **DATA** statements until the entire matrix has been assigned. Row and column zero are not read.

See the **READ** and **DATA** statements for other rules governing reading from DATA statements.

Any array created by a **MAT** statement with a single dimensions assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

EXAMPLES

```
MAT READ A[2,2], B$

MAT READ B$, J
```

ERRORS

Matrix has zero DIMension

Variable not specified

See also

Numeric, Array and Matrix Variables, **READ, DATA**

MAT READ

SYNOPSIS

Read array, matrix or string from a channel.

SYNTAX

MAT READ #*chn.expr*; *var list*...{;}

DESCRIPTION

The # *chn.expr* is any legal channel expression selecting an open file

from which to read data.

The *var list* is any list of variables: *num.var, array.var, mat.var* or *str.var*.

MAT READ transfers data into any *var, mat.var, array.var or str.var*. The operation is similar to a **READ #** statement, except that an entire *array* or *matrix* is transferred; including row and column zero elements. If the specified *var* is a string, its entire specified length is transferred including zero-byte terminators.

If the variable in the list is an *array.var*, an optional *subscript1* and *subscript2* may be specified. If given, these are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old size. A single element *array* can be converted to a *matrix* or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **array[24]**.

Any array created by a **MAT** statement with a single dimension assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by subscripts. The transfer size is rounded up to an even number of bytes. Either an even or odd subscript may be specified. The address for the start of the transfer within the *str.var* is not changed. If an odd number of bytes is specified in the subscript (such as *svar[2,2]*), the size is rounded up to an even number of bytes resulting in the transfer of 2 bytes into *svar[2,3]* in this example. All characters are transferred including zero-bytes.

The optional semicolon (;) terminator is only available for IRIS applications eliminating the automatic record-lock applied to the supplied *record* in the *chn.expr*. Applications may also utilize **MAT RDLOCK #** for operations with locking, and **MAT READ #** for non-locking transfers.

If the application is an IRIS program, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.

If the transfer is to a Formatted Item file, the item type may be String or Binary for any *str.var* in the list, or Binary or Numeric for any numeric variable. The *byte displacement* specifies the starting item for the transfer. If not specified, item zero is assumed. No conversion takes place during the transfer of a binary item. It is the program's responsibility to maintain the correct precisions of numerics being **MAT READ#** from the file.

If the transfer is to a Contiguous or Tree-structured Data file, the *byte displacement* specifies the starting byte within the supplied *record*. Zero is assumed if no *byte displacement* is given. If the program is an

IRIS program, any given *byte displacement* is rounded up to an even value prior to transfer. Attempting to **MAT READ#** at byte displacement one, automatically rounds up to two, for example.

If the transfer is to a text file, the entire string is read and no conversions of returns to new-lines is performed. Each transfer will read a fixed number of bytes. When printing text data that was **MAT READ#**, append a semicolon (;) to the **PRINT** statement to prevent the insertion of automatic return/line-feeds.

Each item transferred causes the *byte displacement* to be incremented by the adjusted byte size of the item in the *var.list*. Strings are sized by the algorithm (INT((d+1)/2)*2), where *d* is the **DIM**ensioned or subscripted size. *num.vars, arrays* and *matrices* are sized as: (R+1) * (C+1) * (size of P) where R is the number of rows, C is the number of columns, and P is the number of bytes occupied by precision P.

EXAMPLES

```
MAT READ #3,R1,100;A,B$,C[12]

MAT READ #C,R;A$
```

ERRORS

Data does not match item specification and cannot be converted

Selected channel is not OPEN

See also

Numeric, Array and Matrix Variables, Channel Expression, **MAT WRITE#, READ#**, Numeric Data, Numeric Variable Precision, Formatted Item Files, Contiguous Files, Text Files

MAT WRITE

SYNOPSIS

Write array, matrix or string to a channel.

SYNTAX

MAT WRITE *#chn.expr*; *var list...*{;}

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to write data.

The *var list* is any list of variables: *num.var, array.var, mat.var* or *str.var*.

MAT WRITE # transfers data from any *var, mat.var, array.var* or *str.var* to the file opened on the supplied *chn.expr*. The operation is similar to a **WRITE #** statement, except that an entire *array* or *matrix* is transferred; including row and column zero elements. If the specified *var* is a string, its entire specified length is transferred

including zero-byte terminators.

If the variable in the list is an *array.var*, an optional *subscript1* and *subscript2* may be specified. If given, these are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old size. A single element *array* can be converted to a *matrix* or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **array[24]**.

Any array created by a **MAT** statement with a single dimension assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision

If the variable in the list is a *str.var*, its size may be controlled by subscripts. The transfer size is rounded up to an even number of bytes. Either an even or odd subscript may be specified. The address for the start of the transfer within the *str.var* is not changed. If an odd number of bytes is specified in the subscript (such as *svar[2,2]*), the size is rounded up to an even number of bytes resulting in the transfer of 2 bytes into *svar[2,3]* in this example. All characters are transferred including zero-bytes.

The optional semicolon (;) terminator is only available for IRIS applications eliminating the automatic record-lock applied to the supplied *record* in the *chn.expr*. Applications may also utilize **MAT WRLOCK #** for operations with locking, and **MAT WRITE #** for non-locking transfers.

If the application is an IRIS program, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.

If the transfer is to a Formatted Item file, the item type may be String or Binary for any *str.var* in the list, and Binary or Numeric for any numeric variable. The *byte displacement* specifies the starting item for the transfer. If not specified, item zero is assumed. No conversion takes place during the transfer of a binary item. It is the programs responsibility to maintain the correct precisions of numerics being **MAT READ** from the file.

If the transfer is to a Contiguous or Tree-structured Data file, the *byte displacement* specifies the starting byte within the supplied *record*. Zero is assumed if no *byte displacement* is given. If the program is an IRIS program, any given byte displacement is rounded up to an even value prior to transfer. Attempting to **MAT WRITE** at *byte displacement* one, automatically rounds up to two, for example.

If the transfer is to a text file, the entire string is written and no conversions of returns to new-lines is performed. Caution must be exercised to prevent the writing of zero-bytes which terminate a text file. Use **MAT WRITE#** with subscripts such as [1,LEN(*str.var*)].

Each item transferred causes the *byte displacement* to be incremented by the adjusted byte size of the item in the *var.list*. Strings are sized by the algorithm (INT((d+1)/2)*2), where *d* is the **DIM**ensioned or subscripted size. *num.vars, arrays* and *matrices* are sized as: (R+1) * (C+1) * (size of P) where R is the number of rows, C is the number of columns, and P is the number of bytes occupied by precision P.

EXAMPLES

```
MAT WRITE #3,R1,100;A,B$,C[12]

MAT WRITE #C,R;A$
```

ERRORS

Data does not match item specification and cannot be converted

Selected channel is not open

Write Protected File

See also

Numeric, Array and Matrix Variables, Channel Expression, **MAT READ#, WRITE#**, Numeric Data, Numeric Variable Precision, Formatted Item Files, Contiguous Files, Text Files

MAT WRLOCK

SYNOPSIS

Write an array, matrix or string with locking.

SYNTAX

MAT WRLOCK *#chn.expr*; *var.list...*

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to write data.

MAT WRLOCK # transfers data from any *var, mat.var, array.var* or *str.var* to the file opened on the supplied *chn.expr*. The operation is similar to a **WRITE #** statement, except that an entire *array* or *matrix* is transferred; including row and column zero elements. If the specified *var* is a string, its entire specified length is transferred including zero-byte terminators.

If the variable in the list is an *array.var*, an optional *subscript1* and *subscript2* may be specified. If given, these are evaluated, truncated to integer and used to select a new working size for the *array*. The total number of elements in the new size cannot exceed that of the old size. A single element array can be converted to a *matrix* or vice versa as long as the total number of elements does not exceed the original **DIM**ensioned size. For example, a [4,4] matrix has 25 actual elements and could be re-declared as **array[24]**.

Any array created by a **MAT** statement with a single dimension assumes a second dimension of one. For example, MAT C= ZER[15] and MAT C = ZER[15,1] are equivalent.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by subscripts. The transfer size is rounded up to an even number of bytes. Either an even or odd subscript may be specified. The address for the start of the transfer within the *str.var* is not changed. If an odd number of bytes is specified in the subscript (such as *svar[2,2]*), the size is rounded up to an even number of bytes resulting in the transfer of 2 bytes into *svar[2,3]* in this example. All characters are transferred including zero-bytes.

If the application is an IRIS program, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.

MAT WRLOCK # transfers data and unconditionally locks the record. The data record remains locked until a non-locking operation is performed by that same program to the same channel. While a record is locked, other users will be unable to access the record.

MAT WRLOCK# is used by BITS applications and is identical to an IRIS **MAT WRITE#** omitting the trailing semicolon.

See the **MAT WRITE#** statement for details on the transfer of data.

EXAMPLES

```
MAT WRLOCK #3,R1,100;A

MAT WRLOCK #C,R;A$
```

ERRORS

Data does not match item specification and cannot be converted

Selected channel is not open

Write Protected File

See also

Numeric, Array and Matrix Variables, Channel Expression, **MAT READ#, WRITE#**, Numeric Data, Numeric Variable Precision, Formatted Item Files, Contiguous Files, Text Files

MODIFY

SYNOPSIS

Change filename or attributes/permissions.

SYNTAX

MODIFY *filename.expr*

DESCRIPTION

The **filename.expr** is any string expression containing a *source filename* to be operated upon, and either new *attributes* or *destination filename*.

The *source filename* specifies the file to be changed. The *destination filename*, if included, selects a new name or location for the *source filename*. **MODIFY** utilizes the UNIX **mv** command to rename or relocate the *source filename*. Commands in the following form are passed to the system:

```
mv source destination

mv SOURCE DESTINATION (If file Indexed)
```

For an Indexed Data File, two commands are issued; lower-case for the data portion, and uppercase for the ISAM portion.

If the file is a Universal Indexed Data File, two **cp** commands are performed; one for the data portion (*filename*), and one for the ISAM portion (*filename* with an *.idx* extension).

If the *source filename* contains a *lu* or *directory* specifier, these must also precede the *destination filename* or the *source filename* is relocated to the current working directory.

attribute string may be expressed as a 2-digit IRIS protection code, 3-digit Unix permission, or as a set of attribute letters. Either the IRIS or Unix types may also include Supplemental Protection Attribute letters preceding the numeric protection digits.

(Release 9.1) Can be used with encrypted files without an encryption key.

EXAMPLES

```
MODIFY "2/FILE 23/OLDFILE"! Move the file

MODIFY "PAYROLL <77>"

A$= "JUNK" \ MODIFY A$+"<E666>"
```

ERRORS

File does not exist

File is Read Protected

File already exists; use '!' to replace

See also

Filenames and Pathnames, File Attributes, Protection and Permissions, Using IRIS Protections, Using Unix Permissions, BITS Attributes, Supplemental Protection Attributes

NEXT

SYNOPSIS

Continuation of **FOR** Loop Statement.

SYNTAX

NEXT *num.var*

DESCRIPTION

num.var is any numeric variable previously used as the *index* variable for a **FOR** statement.

The **NEXT** statement is used to indicate the logical end of a program loop using **FOR/NEXT**.

The **NEXT** statement must have been preceded by execution of a **FOR** statement defining the parameters of the loop. Nested **FOR/NEXT** loops are paired based on the *num.var* used as the *index* variable.

Upon execution of the **NEXT**, the loop's *step* value is added to the *index*. If the new *index* exceeds the loop's final value, normal program execution resumes at the statement following the **NEXT**; otherwise, the *index* value is updated by the *step* and execution reverts back to the statement following the associated **FOR**. If a *step* was not specified on the associated **FOR** statement, one is assumed.

When a loop terminates in IRIS applications, the *index* variable contains the first value not used within the loop. BITS applications terminate a loop with the last value actually used as a loop value.

In *immediate mode*, a **NEXT** statement is only executable on a multi-statement beginning with **FOR**, i.e.:

```
FOR I=1 TO 10 \ PRINT I \ NEXT I
```

EXAMPLES

```
NEXT J7
```

ERRORS

NEXT without a matching FOR

See also

FOR, FORNEXTNEST

ON

SYNOPSIS

Conditional branch on value of expression.

SYNTAX

ON *num.expr* **GOTO** *stn list*

ON *num.expr* **GOSUB** *stn list*

DESCRIPTION

The *num.expr* is any numeric expression which, after evaluation is truncated to an integer *n*. The program will then branch to the *nth stn* in the given *stn list*. If *no stn* corresponds to *n*, then execution continues with the statement following the **ON**.

GOTO and **GOSUB** work precisely as their singular counterparts. Branching will be to the first sub-statement of the statement number given, and the statement must exist.

EXAMPLES

```
ON Q GOTO 200,300,400,500,600

ON (SGN(A)+2) GOTO 300,450,1000 ! Neg, Zero, Pos

ON (A/100) GOSUB 600,750,840,950
```

ERRORS

No such Statement Number

GOSUBS nested too deep

See also

GOTO, GOSUB, GOSUBNEST

OPEN

SYNOPSIS

Open a File for Read and Write Access.

SYNTAX

OPEN *#channel*, *filename.expr* { ,*#channel* } ...

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select a channel number.

filename.expr is any *str.expr* containing a *filename* (including a path) to be opened for read and write access to the program.

The **OPEN** statement links a selected file or device to the specified *channel*. The *file* must already exist on the system or an error is generated.

Multiple *str.expr's* may be specified to open several files on successive channel numbers. Any new *channel* number (*#channel*) in the filename list will cause assignment of channels to continue from that number.

In IRIS applications, if the specified *channel* is already in use, a **CLOSE** statement must be performed prior to an **OPEN**.

Most files to which a user has access may be opened. The same file

may be simultaneously opened by other users, and may be opened on more than one channel. If a file is already opened for exclusive access via **EOPEN** by another process, an error is generated. IRIS applications may not **OPEN** a saved BASIC program for access.

OPEN will link the selected file for read/write access and update each file's last access date.

A file may not be **OPEN** if it, or its directory does not have read permission for the user requesting access. If the file is read-only to the user, an implied **ROPEN** is performed and only read operations are allowed.

A BITS application may **OPEN** a file on a channel that is already opened for a different file. An implied **CLOSE** is performed prior to opening the selected file.

EXAMPLES

```
OPEN #1,"12/DATAFILE","FILE2",#4,"/usr/path/AR.CHECK"

OPEN #3,"$LPT",L$+A$ !EXPRESSION IS LU+FILENAME

OPEN #D,""
```

ERRORS

File does not exist

Not a Data File, cannot OPEN or Replace

File is in use and Locked

Channel is already OPENed and in use

See also

CLOSE, ROPEN, EOPEN, Filenames and Pathnames, Directories and Paths, Channel, **LUST**

PAUSE

SYNOPSIS

Suspend Program Operation.

SYNTAX

PAUSE *delay*

DESCRIPTION

The *delay* is any numeric expression which, after evaluation is truncated to an integer and used to specify a delay in program operation. The delay is limited to an integer between 0 and (2^{32})-1 representing the number of tenth-seconds to delay.

This is the most accurate method of pausing the execution of a program. Other methods, such as finite program loops, will be affected by the current usage of the system and most likely yield

varying results.

The program is unconditionally suspended for the number of tenth-seconds specified in *delay*. An **[ESC]** without **ESC**ape branching or **[EOBC]** terminates a pause. If the application has an **INTSET** defined, the **[INTR]** (CTRL+C) or **[SIGNAL]** (CTRL+B) will terminate the pause and perform the branch.

EXAMPLES

```
PAUSE 30

PAUSE FNA(Q7)

PAUSE A*10
```

ERRORS

Function argument of Statement mode out of range

See also

SIGNAL 3, HZ Environment Variable

PORT

SYNOPSIS

Attach and control other ports.

SYNTAX

PORT *port, mode, status var* { , (*command str* | *acnt* | *return value*)}

DESCRIPTION

port is any *num.expr* which, after evaluation is truncated to an integer and used to select a port number to attach and effect a command.

mode is any *num.expr* which, after evaluation is truncated to an integer and used to select an operation for *port*. There are 4 modes:

Mode	**Operation Performed.**
0	Attach selected port
1	Place an attached port in command mode
2	Transmit a command string to an attached port
3	Return an attached port's operational status

status is any *num.var* used to return the exception status of the operation. The meaning of the *status* depends upon the mode selected.

command str is any *str.var* used for *mode* 2 to send the command to the specified *port*.

acnt is any *num.expr* which, after evaluation is truncated to an integer and used to select a different account for the attached port when using *mode* 0. The account should be expressed as G*256+U, where G and

U are the desired group and user numbers respectively. It's use is restricted: a group manager may attach only accounts within his own group, root may attach any account. All other users may only attach their own account. The Group and User numbers must be in the range 0 to 255. If not specified, the group and user id of the program executing the attach is set.

return value is any *num.var* used to return the operation status of the specified *port* for *mode* 3.

The **PORT** statement allows a *port* to be attached to a program. Once attached, commands may be transmitted to the *port* for normal processing, and the current status or state of the attached *port* can be controlled and monitored. If the attached *port* has a keyboard, it may be used as any other normal terminal. However, commands transmitted will override any current keyboard operation.

Mode 0—Attach Selected Port

A **PORT Mode 0** statement must be issued once for each port being attached. Once attached, the port remains so until signed-off (sending a **BYE** command or executing **SYSTEM 0** to the *port*).

PORT Mode 0 begins by attempting to attach the *port*. If the *port* is already running under UniBasic, the attach operation is complete and successful.

If the *port* is not currently signed onto UniBasic, a background process is created as the supplied *port* number. It assumes the caller's environment and current working directory. It then becomes a unique process linked to the supplied *port number*. This *port* is then available for **CALL $TRXCO** commands, **PORT, SEND, RECV**, and **SIGNAL** statements from any other UniBasic user as well as the program performing the initial **PORT** *mode* 0.

When sending commands to a *port* which is connected to a terminal and keyboard, you must ensure that *port* is already running UniBasic before sending commands. Otherwise, a *phantom port* is created for the supplied *port number*. If a user later attempts entry into UniBasic on a terminal designated as being the same *port number*, entry into UniBasic will be rejected if **PORT** or **PORTS** is defined for that *port number*.

Note: It is impossible to create a phantom port from a child program.

Upon completion, the status variable is set to indicate:

0 Successful, port is now attached.

1 The selected port is already logged-on to the system and in-use.

2 All available ports are already in use. In some configurations, the allowed number of concurrent users is set less than the actual

number of ports configured. This indicates that either another *port* or *phantom port* must be signed-off, or the number of concurrent users increased on your license.

3 Illegal account number selected. The selected group or user number is out of the range 0-255.

Mode 1—Place an Attached Port in Command Mode

PORT Mode 1 sends an **ESC**ape Override Character **[EOBC]** to the selected *port*, terminating any running program and placing the *port* into *command mode*.

Upon completion, the status variable is set to indicate:

0 Successful, the selected port is now in command mode.

1 The select port is not attached.

Mode 2—Transmit Command String to Attached Port

PORT Mode 2 requires that a *command str* be supplied following the *status* variable. The string data in *command str* is then transmitted to the selected *port*. This *command str* may contain any legal command input for a terminal. It's entire length may not exceed the *port's* input buffer size as defined by the environment variable **INPUTSIZE**. This is generally 256 characters. Any command, such as **NEW, LIST, BYE, RUN**, etc., may be transmitted, as well as program statements. If a terminal is connected to the attached *port*, the *command str* is echoed as it is processed on the attached *port*. An attached *port* connected to a terminal may also receive commands from its keyboard.

A *command.str* cannot be transmitted unless the attached *port* is in an 'input ready' state. A **PORT Mode 3** status check is suggested prior to sending a command.

Upon completion, the status variable is set to indicate:

0 Successful, command transmitted and accepted.

1 The selected port is not attached.

2 The selected port is not in an 'input ready' state.

Mode 3—Return Attached Port's Operational Status

PORT Mode 3 requires that a *return value* be supplied following the *status* variable. This variable will receive a value indicating the port's operational status. A **PORT Mode 3** should always precede any *mode* 2 command transmission to check for 'input ready'. It may also be used to monitor the current state of the attached *port*.

0 Successful, operational status returned.

1 The selected port is not attached.

The value returned as the operational status consists of a mode, an 'Input Ready' flag, and an 'Output in Progress' flag.

This value may be divided into its respective parts as follows:

Assume X = value returned by PORT mode 3.

```
IF X>32767 THEN 'Input Ready' on attached port.
```

The 'Input Ready' flag must be removed from the value prior to testing the 'Output in Progress' flag, since both input and output may be in progress.

```
IF X>32767 THEN X=X-32768 \! Remove flag.

IF X>16383 AND X<32768 THEN 'Output in Progress'
```

The attached port's current mode can be determined by:

```
LET M=X % 16 \! Retrieve mode.
```

Mode	Current State
0	Idle. At command mode or logged-off.
1,2	Command input or execution.
3	Run. Program execution in progress.
4,5	List. Program listing in progress.
6	Statement execution in immediate mode.
7	Get. Program being loaded from text file.
8	Initial Run. Becomes mode 3.
9,10	Enter. Program statement entry using **ENT**er.

EXAMPLES

```
PORT 8,0,S \ IF S STOP ! ATTACH & CHECK STATUS

PORT P,1,S \ IF S STOP ! ABORT & GET READY

PORT P*2,2,E,C$[50] \ IF E STOP ! SEND COMMAND

PORT X,3,Y,Z \ IF Y STOP ! GET CURRENT MODE&STAT
```

ERRORS

Function Argument or Statement Mode out of range

See also

CALL $TRXCO, SWAP, SPAWN, Environment Variables: **PORT, PORTS** and **INPUTSIZE**, Port Numbering & Phantom Ports, Launching UniBasic ports at startup

PRINT

SYNOPSIS

Output ASCII to screen, file or device.

SYNTAX

PRINT {*#chn.expr*;} { **USING** format } *var.list*

DESCRIPTION

PRINT may be abbreviated by the single character **;**.

The optional *#chn.expr* is any legal channel expression selecting an open channel to re-direct output.

The optional **USING** *format* is *str.expr* including a valid **USING** Operator allowed for numeric formatting.

The *var.list* consists of variables, literals, or expressions; numeric or string. Each item in the *var.list* must be separated by either a comma (,) or a semicolon (;). A comma performs a **TAB** to the next comma field before output of the next item. This is generally 20 characters long, but may be changed by the setting of the environment variable **TABSIZE**. A semicolon prevents additional spacing in the output.

Numerics are output preceded by a '-' or space indicating negative or positive, and followed by one space (The **STR** function may be used to omit leading and trailing spaces). Strings are output exactly as stored, from the supplied starting position terminating at the first zero-byte terminator. No preceding or trailing spaces are output.

When all items in the *var.list* are output, a new-line is output to advance the terminal to the next line (or mark end of line in a text file). This can be suppressed by using a comma or semicolon as the last character in the **PRINT** statement. In the case of a comma, a **TAB** is still performed.

The **USING** operator formats numeric data for columnar output. It may also be used to imbed commas, asterisk check fill, floating dollar signs and other special output formats. It must be after any *chn.expr* and before the *var.list*, and only one is allowed per statement. For additional information, see the string operator **USING**.

An output column counter (base zero) is maintained for each terminal holding the current character position on the output line. This counter is reset anytime the **[EOL]** is output (usually a return) or a @0,y cursor positioning operation is performed.

The **TAB** function is used to skip the terminal to a specific column. Its form is:

TAB (*num.expr***)**

The **num.expr** must be a positive value in the range 0 to 255 or an error will be generated. The value is truncated to an integer and set to zero if it is greater than 255 and less than 32768. A **TAB** to a position less than the current position is ignored. A **TAB** is performed by generating the required number of spaces to skip to the desired position.

After all items in the *var.list* are placed into the terminal buffer, it is flushed immediately. No **SIGNAL 3,0** is required to start output, and is ignored if executed.

If a *chn.expr* is specified for **PRINT**, the output is redirected to the selected *channel*. If the *channel* is not open, output is transmitted to the terminal. This allows a program to selectively output to the terminal or a printer by including an **OPEN** of the printer *pipe* on the selected *channel*. A separate output column counter is maintained for each *channel* opened, so that the **TAB** and comma operator will operate on applications doing both screen and file output operations.

PRINT # is generally used to output to a text file, or *pipe* such as a line printer. The most common form used for output to a line printer is:

PRINT *#chn.expr*; *var.list*

The optional *record*, *byte displacement* and *time-out* specifications of a *chn.expr* are normally unused, as line-oriented data is generally of variable length. Each successive **PRINT #** continues its transfer immediately following the previous, unless a *new record* or *byte displacement* is specified.

EXAMPLES

```
PRINT "AVAILABLE";TAB(40);A*100;"$";Z

;@0,23;'CL';"Error in Program";

PRINT #K; USING T$;X,Y,Z,Z/10
```

ERRORS

Function Argument is out of range

Write Protected file

See also

TAB Function, Numeric and String Expressions, **TABSIZE**, Channel Expression, **STR** Function, String operator **USING**

RANDOM

SYNOPSIS

Seed random generator for **RND** function.

SYNTAX

RANDOM *num.expr*

DESCRIPTION

The *num.expr* is evaluated, truncated to a positive integer and used to seed the system's pseudo-random number generator. Seeding implies that a sequence is selected and initiated based on the value supplied.

Each value from 1 to 65535 will select a unique pseudo-random sequence for the **RND** function. A seed value of zero selects a further random sequence based upon the current system time, yielding 36000 different sequences.

Typically, a non-zero seed value is used during program debugging, causing the **RND** function to yield the same sequence of numbers with each successive run. Once the program is completed, a **RANDOM 0** is issued to produce better random selection.

EXAMPLES

```
RANDOM 5

RANDOM 0

RANDOM ((N*100)/E^2)
```

ERRORS

Arithmetic Overflow

Function argument or Statement Mode out of range

See also

RND function

RDLOCK

SYNOPSIS

Read and unconditionally lock a record.

SYNTAX

RDLOCK *#chn.expr*; *var.list...*

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to read data.

RDLOCK # transfers data into any *var, mat.var, array.var* or *str.var*.

If the variable in the list is an *array.var*, an optional *subscript1* and *subscript2* may be specified. If given, these are evaluated, truncated to integer and used to select a single element. If no *subscripts* are supplied, only the first element is transferred.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by subscripts. All characters are transferred including zero-bytes.

RDLOCK transfers data and unconditionally locks the record. The data record remains locked until a non-locking operation is performed by that same program to the same channel. While a record is locked, other users will be unable to access the record.

RDLOCK# is identical to **READ#** omitting the trailing semicolon.

See the **READ#** statement for details on the transfer of data to different types of files.

EXAMPLES

```
RDLOCK #3,R1,100;A

RDLOCK #C,R;A$
```

ERRORS

Syntax error

Channel is not Opened

See also

READ#

RDREL

SYNOPSIS

Read a relative 512-byte block from a file.

SYNTAX

RDREL # *chn.expr*; *str.var*

DESCRIPTION

The # *chn.expr* is any legal channel expression selecting an open file from which to read data. The *chn.expr* must include a *record* which is used to define the relative block within the file to read. The *byte displacement* and *time-out* expressions are ignored and unnecessary.

The *str.var* is any string variable **DIM**ensioned at least 512 bytes. A starting *subscript* may be supplied as long as the **DIM**ensioned size is at least 512 bytes larger than the supplied *subscript*.

RDREL uses the supplied *record* as a relative 512 byte block pointer into the file. For example, *record 0* specifies the first 512 bytes in the file, *record 1*, the second 512 bytes, etc.

Record -1 may be used to load the first 512 bytes of the file. This includes the header and possibly part of record 0. Some headers (of formatted item files) may be larger than 512 bytes and may not be read in entirety. To retrieve header information in a truly machine independent fashion, it is recommended that **CALL 127** be used to unpack the information. **RDREL #** of record -1 is used to change header information by conversion and other utilities.

RDREL is generally used to copy files or otherwise read portions of files not accessible with a normal **READ#** statement. Processing of the data is left completely up to the user.

EXAMPLES

```
RDREL #7,K;A$ ! READ A BLOCK

RDREL #7,K+1;A$[513] ! APPEND A SECOND BLOCK
```

ERRORS

Channel Not Opened

llegal Record or End of File

See also

WRREL#

READ

SYNOPSIS

READ variables from **DATA** statements.

SYNTAX

READ *var.list*

DESCRIPTION

The *var.list* contains any *num.var, array.var, mat.var* or *str.var* names. An *array.var* or *mat.var* with *subscripts* specifies only that single element. Omission of a *subscript* selects only the first element.

READ begins transferring data sequentially from the lowest numbered **DATA** statement found in the program. Subsequent **READ** statements resume transfer at the next element of the **DATA** statement. After all of the items in a given **DATA** statement have been read, reading continues with the next highest numbered **DATA** statement. When all **DATA** statements have been read, any subsequent **READ** will produce the error Out of Data. The **RESTOR** statement can be used at any time to start reading from a specific **DATA** statement.

READ attempts to transfer data into each variable listed in the *var.list*. Transfer of a variable terminates at a comma (,) or at the end of the **DATA** statement. The format of the data is left to the user. You may not transfer string data into any numeric variable. Generally, string items need not be enclosed in quotes (" "), but can be if desired. Quotes will be necessary if it is desired to include a comma inside a string item.

EXAMPLES

```
READ A,B,D[10],A[4,4]

READ A$
```

ERRORS

Out of Data

String Variable is not Dimensioned

Illegal Subscript Supplied

Data of wrong type (numeric/string)

See also

DATA, RESTOR, MAT READ, String Data and Literals, Numeric, Array and Matrix Variables, Variable Names

READ

SYNOPSIS

Read array, matrix or string from a channel.

SYNTAX

READ *#chn.expr*; *var list...*{;}

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to read data.

READ # transfers data into any *var, mat.var, array.var* or *str.var*.

If the variable in the list is an *array.var* or *mat.var*, only the first element ([0] or [0,0]) is read. *Subscripts* may be used to select any individual element to be transferred. The number of bytes transferred is based upon the variable's dimensioned size. The transfer is performed according the rules for a *num.var*.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by *subscripts*. The entire size is then transferred including zero bytes. When no *subscript*, or a single *subscript*, is specified, IRIS programs increment the total number of bytes transferred. When reading from a contiguous formatted file only, **READ** will stop at the first null byte in a record, preserving the existing data within the string variable.

The optional semicolon (**;**) terminator is used by IRIS applications to eliminate the automatic record-lock applied to the supplied *record* in the *chn.expr*. BITS applications utilize **RDLOCK #** for operations with locking, and **READ #** for non-locking transfers.

If the running program is an IRIS program, the following steps are performed prior to transfer:

1. If the variable to be transferred is a *num.var, array.var*, or *mat.var*, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.
2. If a full *str.var* is supplied (single or no *subscript*), its size is incremented by one to account for an extra null byte (A null will be forced into the last position following the transfer). Finally, if the transfer is from a text file, an error is generated

if any *num.var* is supplied.

If the transfer is to a Formatted Item file, the item type may be String or Binary for any *str.var* in the list, and Binary or Numeric for any *num.var, array.var*, or *mat.var*. The *byte displacement* specifies the starting item for the transfer. If not specified, item zero is assumed. No conversion takes place during the transfer of a binary item. It is the program's responsibility to maintain the correct precisions of numerics being **READ** from the file.

If the transfer is to a Contiguous or Tree-structured Data file, the *byte displacement* specifies the starting byte within the supplied *record*. Zero is assumed if no *byte displacement* is given, and IRIS programs round up the *byte displacement* if odd on a numeric variable transfer.

If the transfer is from a text file (IRIS Programs only), data is read up to and including the next new-line character in the file. The new-line is converted and stored in the string as a \215\ for compatibility. A null string is returned when the end of a text file is reached.

Each item transferred causes the *byte displacement* to be incremented by the adjusted byte size of the item in the *var.list*. Strings are sized by the algorithm (INT((d+1)/2)*2), where *d* is the **DIM**ensioned or subscripted size. *num.vars, arrays* and *matrices* are sized as: (R+1) * (C+1) * (size of P) where R is the number of rows, C is the number of columns, and P is the number of bytes occupied by precision P.

EXAMPLES

```
READ #3,R1,100;A,B$,C[12]

READ #C,R;A$
```

ERRORS

Data does not match item specification and cannot be converted

Selected channel is not open

See also

Numeric, Array and Matrix Variables, Channel Expression, **WRITE#, MAT READ#, RDLOCK#**, Numeric Data, Numeric Variable Precision, Formatted Item Files, Contiguous Files, Text Files

RECV

SYNOPSIS

Receive communication message.

SYNTAX

RECV *port*, (*string* | *value1, value2*) {, *delay* }

DESCRIPTION

port is any *num.var* to receive the sender's *port number*, or -1 if no

messages are waiting for your *port*.

string is any *str.var* up to 512 bytes in length to receive a string message.

value1 and *value2* are any two *num.vars* to receive 2 numeric messages. If the second parameter is a *num.var*, two numeric variables must be specified. Their two values are then received. The two variables need not be the same precision.

The optional *delay* is any numeric expression which, after evaluation is truncated to an integer to specify a delay period (in tenth-seconds) during which the program awaits a message. If zero, or not included, no pause is invoked, but any currently waiting message will be received. Any message appearing during a specified delay allows **RECV** to accept the transmitted data and resume program execution immediately. If no message appears during the entire delay, *port* is set to -1.

If the program has an **INTSET** branch enabled, any message sent to your *port* will cause a branch to the selected statement. The interrupt handling routine can then perform a **RECV** to receive the message.

RECV is identical in operation to **SIGNAL 2**.

EXAMPLES

```
RECV P,A,B,600 ! Wait 60 seconds

RECV P,A$
```

ERRORS

Arithmetic Overflow

See also

SIGNAL, SEND, INTSET

REM

SYNOPSIS

Non-executed Program Comments.

SYNTAX

REM any ASCII characters

DESCRIPTION

The **REM** statement allows the placement of comments within a program. A **REM** statement is ignored during execution, but may be referenced within the program.

When **REM** statements are entered, all characters following the **REM** up to the **[EOL]** (usually return) are considered the comment. This includes leading and trailing spaces and control characters.

A **!** may be used to abbreviate the verb **REM** during entry. During listing, **REM** is listed if it is the first statement of the line, otherwise **!** is displayed. When a **REM** statement is processed during program execution, the statement is ignored. Branching (**GOTO**, **GOSUB**, etc.) to **REM** statements is acceptable with little program overhead.

Note that, since all characters following a **REM** are considered part of the **REM**, the **REM** is always the last statement on it's line.

```
400 PRINT A \ REM OUTPUT TOTAL \ GOTO 200
```

Line 400 outputs the value of A and continues with the next program line. The "GOTO 200" is considered to be part of the comment.

EXAMPLES

```
REM Request input of customer name

GOSUB 1000 ! Go receive response
```

ERRORS

none

RESTOR

SYNOPSIS

Reset **DATA** pointer for **READ** Statement.

SYNTAX

RESTOR {*stn*}

DESCRIPTION

RESTOR may also be entered as **RESTORE**.

RESTORE resets the **DATA** statement pointer to the first data item of the first **DATA** statement in the program, just as when the program started.

Including an optional *stn* sets the pointer the first data item of the first **DATA** statement encountered at or past that *stn*.

If no further **DATA** statements are found, the pointer will be set to return an "Out of DATA" error during the next **READ**.

EXAMPLES

```
RESTOR

RESTORE 2200
```

ERRORS

No such statement number

Out of Data

See also

DATA, READ

RETURN

SYNOPSIS

Return from prior **GOSUB** subroutine call.

SYNTAX

RETURN {(+ | -) *increment* }

DESCRIPTION

The optional *increment* is any *num.expr* which, after evaluation is truncated to an integer to specify an offset forward or backward (+ or -) from the normal **RETURN**.

The **RETURN** statement is used with **GOSUB** and indicates the end of a program subroutine.

A normal **RETURN** (or **RETURN +0**) resumes execution at the statement following the matched **GOSUB**. A value of **+1** would branch to the second statement following the **GOSUB** (the first statement past a normal **RETURN**). A value of *-1* would branch to the statement of the *GOSUB* itself.

BITS programs treat **GOSUB** and **RETURN** as line rather than statement oriented functions. A normal **RETURN** resumes execution at the next line of a program; that is subsequent statements on the same line as the **GOSUB** are ignored. *-1* is used to re-execute the line containing the **GOSUB** and **+1** skips the line following. A special **RETURN + 0** returns within a multi-statement line to the statement following the **GOSUB**, if any. Typical BITS applications performing **GOSUB** on multi-statement lines use **RETURN +0** for an error condition, and normal **RETURN** {+ - } as normal exits.

EXAMPLES

```
RETURN

RETURN +1
```

ERRORS

RETURN without prior GOSUB

See also

GOSUB, GOSUBNEST

REWIND

SYNOPSIS

Reset a file to the first data byte.

SYNTAX

REWIND # *channel* {, ...}

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select a channel number.

Multiple *#channel* designations are permitted separated by comma.

The **REWIND** statement resets the selected *channel's* current file position to the beginning of the file. The position is reset to *record* 0, *byte displacement* 0. If the next file transfer does not specify a *record* or *byte displacement*, the transfer will start at the first data byte of the file.

The effect of **REWIND** is to reset the current file position as when the channel was initially opened. **REWIND** is typically used with Text Files accessed sequentially.

A **REWIND** operation is ignored when issued to a *channel* linked to a *pipe*.

REWIND is identical in operation to **SETFP** *#channel, 0, 0* ;

EXAMPLES

```
REWIND #T, #7, #(J*2)
```

ERRORS

Channel Not Opened

See also

Text Files, **SETFP#**

ROPEN

SYNOPSIS

Open a file for Read-Only access.

SYNTAX

ROPEN *#channel*, *filename.expr* { ,*#channel* } ...

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select a channel number.

filename.expr is any string expression containing a *pathname*, or *filename* to be opened for read-only access to the program.

The **ROPEN** statement links a selected file or device to the specified *channel*. The *file* must already exist on the system or an error is generated.

Multiple *filename.expr's* may be specified to open several files on successive channel numbers. Any new *channel* number (*#channel*) in the *filename* list will cause assignment of channels to continue from that number.

In IRIS applications, if the specified *channel* is already in use, a **CLOSE** statement must be performed prior to an **ROPEN** .

Most files to which a user has access may be opened. The same file may be simultaneously opened by other users, and may be opened on more than one channel.

A file may be opened for read-only using **ROPEN** even if it is already opened for exclusive access via **EOPEN**. IRIS applications may not **ROPEN** a saved BASIC program for access.

ROPEN will link the selected files for read-only access without updating its last access date. In addition, a file opened for read-only may read records locked by other applications making this statement especially valuable for reports and general inquiries.

A file may not be **ROPEN** if it, or its directory does not have read permission for the user requesting access.

A BITS application may **OPEN** a file on a channel that is already opened for a different file. An implied **CLOSE** is performed prior to opening the selected file.

EXAMPLES

```
ROPEN #1,"DATAFILE","FILE2",#4,"AR.CHECK"

ROPEN #3,"$LPT",L$+A$ !EXPRESSION IS LU+FILENAME
```

ERRORS

File does not exist

Not a Data File, cannot OPEN or Replace

Channel is already OPENed and in use

See also

CLOSE, EOPEN, Filenames and Pathnames, Directories and Paths, Channel

SEARCH

SYNOPSIS

Search string for sub-string.

SYNTAX

SEARCH *source*, *target*, *location*

DESCRIPTION

source and *target* are any *str.exprs*.

location is any *num.var* to contain the byte position of the *target* within the *source*, or zero if not found.

source is searched for the first occurrence of *target*. If found, *location* is set to the character position of the located substring. If not found, a zero is returned. If the *source* being searched is a single *str.var*, it may include a starting *subscript* if desired, and searching begins at the selected position. Note however that any position returned will be relative to this starting position.

When performing multiple **SEARCH** operations on a single string, it is best to initialize a *num.var* to 1; adjusting for each located identical sub-string.

```
290 LET J=1

300 SEARCH T$[J],"H-",R

310 IF R THEN LET J=(J+R)-1
```

Here, *location* is adjusted for the offset caused by a starting *subscript*. If the substring is not found, *location* is returned as zero. The adjustment needed for any given starting subscript 'A' can be defined as:

actual position in string = starting *subscript* + *location* - 1

SEARCH performs a 7-bit comparison on both strings. This means, for example, that a **\15** code is considered equal to a **\215** code. Searching terminates when a null byte is encountered in the *source* *str.expr*. Entry of the verb **SEARCH** followed by a # character is interpreted as an ISAM file **SEARCH** statement and treated as such.

EXAMPLES

```
SEARCH P$+A$,".",K

SEARCH A$[J],"TIME",K \ J=J+K-1
```

ERRORS

String expression must be used here

See also

CALL $STRING, CALL 44, CALL 56

SEARCH

SYNOPSIS

Indexed File maintenance statement.

SYNTAX

SEARCH *#channel* , *mode* , *index* ; *key*, *record* , *status*

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an

integer, and used to select an opened *channel* currently linked to an Indexed Data file.

The *mode* is any *num.expr* which, when truncated to an integer, specifies a mode of operation for the **SEARCH** statement. For a detailed list of *mode* operations, see Indexed Data Files.

The *index* is any *num.expr* which, when truncated to an integer, specifies which Index or Directory (list of keys) the operation is being directed.

The *key* is any **DIM**ensioned *str.var* which must be **DIM**ensioned to at least the size of the Key for the specified Index.

The *record* is any *num.var* and contains (or returns) a value for the statement *mode*.

The *status* is any *num.var* used to return a status value to the program. Refer to the following pages for a list of *status* values and their meanings.

Parameters may be separated by either a comma or semicolon terminator.

EXAMPLES

```
SEARCH #5,4,1;K$,R1,E \ IF E GOTO 1000

E=3 \ SEARCH #J,1,0;K$,R1,E \ IF E GOTO 1000
```

ERRORS

Selected channel is not open

Illegal parameter or syntax for command

Selected data record is locked

File is not indexed or mapped

Illegal or nonexistent index number selected

File is write protected

Index selected is not yet initialized

Indexed file structure error or svar dim length < Key length

See also

Indexed Data Files, **INDEX #, AVAILREC, BUILDXF, PREALLOCATE, ERR( )**

Summary of SEARCH # Modes

Mode	Operation
0	Define and Create indices within a Contiguous Data File.
1	Return miscellaneous index information.
2	Search for an exact key.

3 Search for the next highest key.

4 Insert a new key into an index.

5 Delete an existing key from an index.

6 Search for the previous key (Search Backward).

7 Unused, included for compatibility.

8 Maintain the B-Tree insertion algorithm for an index.

9 Temporarily same as Mode 6 - Reserved for future use.

Detailed Table of SEARCH # Modes

Mode	Index	Status	Operation Performed
0	1≤d≤63		For a new Indexed File, sets the *key* length of the selected *index* to the number of bytes specified by *record.var*. The maximum *key* length is 122 bytes. Indices must be defined starting at one and proceed sequentially.
0	0		Freeze the file definition and build the ISAM portion of the file. Total number of initial data records is specified by the *record.var*.
1	>0		Return the *key* length of the specified *index* in bytes.
1	0	=0	Returns the record number of the First Real Data Record. Normally zero unless the file was built using **BUILDXF** or copied from an IRIS or BITS system.
1	0	=1	Return the number of Available Records in the file. This value is either the value of the environment variable **AVAILREC**, or the value based upon the files current size.
1	0	=2	Allocate and return a new record for the application.
1	0	=3	Return a record to the file that is no longer needed. Deleted records will be re-used before the file is extended.
1	0	=4	Return in *record.var* the number of records in the file. IRIS Applications only; Error for BITS applications.
1	0	=5	Return in *record.var* the number of records in the file. For BITS applications, performs the same operation as 4 above.
1	0	=6	Set the First Real Data Record to the value supplied in *record.var*. This option is only

			available during file structuring.
1	0	=7	Return the current number of records in use (allocated) in the data portion of the file.
2			Search the specified *index* for the exact match of the supplied *key.var*. If found, return the full key in the supplied key variable, and the associated record number in *record.var*. The *status.var* is set to 0 if the *key* was found, and 1 if the *key.var* was not in the index.
3			Search the specified *index* for the first key whose value logically exceeds the supplied *key.var*. If found, *status.var* is set to 0, the full key is returned in *key.var*, and the associated record number is returned in *record.var*.
4			Insert *key.var* into the specified *index* using the supplied *record.var* as the associated pointer. The record should have been previously allocated using *mode* 1, *status* = 2 above. A *status.var* of 0 indicates a successful operation. If the *key.var* already exists in the *index*, a 1 is returned as *status.var*.
5			Delete the supplied *key.var* from the specified *index*. If successful, *record.var* is returned as the associated pointer, and the *status.var* is set to 0. A *status.var* of 1 indicates an unsuccessful operation; i.e., the *key.var* was not found in the index. The record should be returned to the file using *mode* 1, *status* = 3 above.
6			Search the specified *index* for the first key whose value is logically less than the supplied *key.var*. If found, *status.var* is set to 0, the full key is returned in *key.var*, and the associated record number is returned in *record.var*.
7			No operation. Reserved for future use.
8			B-Tree algorithm maintenance. If *record.var* is negative, return in *record.var* the current B-Tree algorithm for *index*. If *record.var* is positive, change the insertion algorithm to the value passed in *record.var*. Set to zero (default) for random insertion, 1 for increasing insertion, 2 for decreasing insertions.
9			Temporarily, the same as Mode 6. Reserved for future use.

Table of SEARCH # status return values

Value	Description of Status
0	No error, the Index operation was successful.
1	Operation was unsuccessful; i.e. *key* not found.
2	End of *index*. Given on *modes* 3, 6 and 9 when the beginning or end of the *index* is reached.
3	End of data; all records are allocated. This error is only generated when the environment variable **PREALLOCATE** is defined to limit the number of data records.
4	File has no Indices, cannot perform an Indexed File operation.
5	Indexed file structure error; given when *key* length **DIM** is less than the actual size of the key from an *Index* on *Modes* 2, 3, 6 and 9. Indicates a **DIM**ension error or structure problem, possibly a c-tree file structuring error. Printing the value of **ERR(8)** will provide a more concise description of the error.
6	Index number not in sequence during creation. You must sequentially define all directories.
7	File is not a Contiguous File.
8	File is already Indexed.
9	Value of *record* is negative or too large.
10	Illegal Index Number. Must be between 1 and 62.

SEND

SYNOPSIS

Transmit a message to another port.

SYNTAX

SEND *port*, (*string* | *value1*, *value2*)

DESCRIPTION

port is any *num.expr* which, after evaluation is truncated to an integer selecting a UniBasic *port number* to receive a message transmission.

string is any *str.expr* up to 512 bytes in length to transmit to the selected *port*.

- or -

value1 and *value2* are any two *num.exprs* to transmit to the selected *port*.

If the second parameter is numeric, two numeric expressions must be specified. Their two values are then transmitted. The two variables need not be the same precision.

It is up to the program on the receiving port to execute the appropriate

RECV or **SIGNAL 2** statement to receive the type (string/numeric) of data transmitted. If that program has an **INTSET** branch enabled, **SEND** will cause an interrupt to occur in it.

SEND is identical in operation to **SIGNAL 1**.

EXAMPLES

```
SEND 12,22,33

SEND P,A$
```

ERRORS

Illegal Port Number selected

Communication Buffer is full

See also

RECV, SIGNAL, Communications File, **INTSET**

SETFP

SYNOPSIS

Set file position for sequential access.

SYNTAX

SETFP # *chn.expr* ;

DESCRIPTION

The # *chn.expr* is any legal channel expression selecting an open file to reposition. A semicolon must terminate the *chn.expr*.

SETFP specifies a new file position on a *channel* for the next sequential access **READ**, **WRITE**, etc. not specifying a *record* or *byte displacement*. If the next transfer specifies its own *record* and *byte displacement* position, the former position is overridden. The *byte displacement* specification is optional and, if not included, will default to byte zero of the selected record.

For item files, only the *record* specification is relevant, as byte position will be affected by the file's record format when the transfer begins. **SETFP** is normally used by BITS applications, since IRIS sequential transfers are record and not byte oriented.

SETFP to *record 0, byte displacement 0* is identical in operation to a **REWIND**.

EXAMPLES

```
SETFP #6,R,I;

SETFP #5,0,0; ! Same as REWIND #5;
```

ERRORS

Channel Not Opened

Illegal Record or End of File

See also

REWIND#, READ#, WRITE#, Formatted Item Files, Contiguous Data Files, Indexed Data Files, Tree-Structured Data Files, Channel Expression

SIGNAL

SYNOPSIS

Transmit/Receive messages and pause.

SYNTAX

SIGNAL *mode* {*optional parameters*}

SIGNAL 1, *port*, (*string* | *value1, value2*)

SIGNAL 2, *port*, (*string* | *value1, value2*) {, *delay* }

SIGNAL 3, *delay*

SIGNAL 5, *port*, *value1, value2* {, *delay* }

SIGNAL 6, *type*, *value1*, *value2*

DESCRIPTION

The *mode* is evaluated, truncated to integer, and used to specify the desired operation for **SIGNAL**.

Mode	Operation Performed
1	Transmit a message to another port number.
2	Receive a pending message sent to this port number.
3	Pause the program a specific amount of time.
4	unused.
5	Receive System SIGNAL for Input time-out.
6	Clear all messages waiting for this port number.

port is any *num.expr* which, after evaluation is truncated to an integer selecting a UniBasic *port number* for transmission. *port* must be any *num.var* if the *mode* specifies reception of a message.

string is any *str.expr* when transmitting data, or *str.var* dimensioned up to 512 characters when receiving string data from another *port*.

value1 and *value2* are any *num.exprs* when transmitting data, or *num.vars* when receiving numeric data from another *port*. If the second parameter is numeric, two numeric expressions must be specified. Their two values are then transmitted. The two variables need not be the same precision.

The optional *delay* for **SIGNAL 2** or **5** is any *num.expr* which, after

evaluation is truncated to an integer to specify a delay period (in tenth-seconds) during which the program awaits a message. If zero, or not included, no pause is invoked, but any currently waiting message is received. Any message appearing during a specified delay allows **SIGNAL** to accept the transmitted data and resume program execution immediately. If no message appears during the entire delay, *port* is set to -1.

type is any *num.expr* which, after evaluation is truncated to an integer selecting the type of signals to be cleared for *mode 6*.

The **[SIGNAL]** input character (usually CTRL+B) transmits a message of 2 numeric zeros or a null string to your current *port number*. This message is retrieved using **RECV** or **SIGNAL 2**.

Mode 1 - Transmit a message to another port

The *string* expression up to 512 bytes in length, or 2 *num.expr* values are placed into the communication buffer for transmission to the selected *port*. Messages may be transmitted to your current *port number*, or any *port number* that is logged on. An IRIS error 62 is returned if the destination port is invalid.

Messages are FIFO (First in, First out). Messages include those transmitted using **SEND, SIGNAL 1**, and **CALL $TRXCO**.

If numeric data is transmitted, full floating point precision (6-word Base 10000) is transmitted. When numeric values are received with **SIGNAL 2**, they are converted to the precision of the supplied *value1* and *value2 num.vars*.

An error is generated if the communication buffer is full, or an illegal *port number* is specified. Messages transmitted to a *port* not signed into a UniBasic process are discarded, and no error is generated.

Messages awaiting a *port* are deleted when that *port* ends its session (**BYE, SYSTEM 0**, terminated **SPAWN** or UniBasic -F commands).

Mode 2 - Receive messages sent to your port

A scan is performed for the oldest **SIGNAL 1** or **SEND** message transmitted to your *port number*. If found, *port* is set to the *port number* of the sender. If no messages are waiting, *port* is set to -1.

The received message is copied into *string* or *value1* and *value2* as specified. It is the programs' responsibility to select the same format (*str.var* or 2 *num.vars*) used by the sender. The sender's *port number* is returned in the supplied *port* variable. Typically, an application designer chooses one format for all message transmission and reception.

If *delay* is specified and no message is waiting, the program is paused for

the specified number of tenth-seconds. If any message is transmitted during the *delay*, the pause is terminated allowing immediate reception. A -1 is returned in *port* if no message is received within the *delay* period.

The **[SIGNAL]** input character (usually CTRL+B) transmits a message of 2 numeric zeros or a null string to your current port which may be retrieved using **SIGNAL 2**.

All messages may be cleared by performing repeated **SIGNAL 2** statements until *port* is returned with -1, or by issuing a **SIGNAL 6**.

If the program has an **INTSET** in effect, transmission of a message by another port or **[SIGNAL]** character performs an interrupt branch.

Messages awaiting a *port number* are deleted when that *port number* ends its session (**BYE**, **SYSTEM 0**, terminated **SPAWN** or UniBasic -F commands).

Mode 3 - Pause Program Operation

The program is unconditionally suspended for the number of tenth-seconds specified in *delay*. An **[ESC]** without **ESC**ape branching or **[EOBC]** terminates a pause. If the application has an **INTSET** defined, the **[INTR]** (CTRL+C) or **[SIGNAL]** (CTRL+B) will terminate the pause and perform the branch.

If *delay* is zero, the statement is ignored and no pause is performed. The maximum pause time is approximately (2^{32})-1 tenth-seconds.

Mode 5 - Receive System Signal

A scan is made for the oldest system message directed to your *port number*. If no system message is waiting, *port* is set to -1.

If a system message is waiting, *port* is set to -2, *value1* is set to the type of system message, and *value2* returns specific information.

The only system message currently implemented is for **INPUT** timed-out. This occurs when an application performs an **INPUT TIM**, and the input times-out without response from the keyboard. *port* is set to -2, *value1* is set to 0, and *value2* is set to the number of characters entered prior to time-out.

Programs performing an **INPUT TIM** should immediately follow with a **SIGNAL 5** to check the sense of the timed input and prevent overflowing the communication buffer. If *port* returns -1, a response was entered within the prescribed time limit.

Mode 6 - Clear all outstanding signals

All user messages, system messages or both may be cleared using

SIGNAL 6. The *type* selects the messages to be cleared from the system:

Type	Function Performed
-1	Remove all user messages; **SIGNAL 1, SEND**.
-2	Remove all system messages.
-3	Remove both user and system messages.

SIGNAL 6 may be used to clear the message queue for this *port number* . Messages are automatically deleted when a *port* ends its session (**BYE, SYSTEM 0**, terminated **SPAWN** or UniBasic -F commands).

EXAMPLES

```
SIGNAL 1,P,A,B*100

SIGNAL 2,P,A,B,300 !Wait 30 seconds

SIGNAL 3,30 !Pause 3 seconds

SIGNAL 5,P,A,B

SIGNAL 6,-3,A,A
```

ERRORS

Illegal Port Number selected

Function Argument or Statement Mode out of range

Communication buffer is full

See also

SEND, RECV, PAUSE, INTSET, INPUT TIM, SPC(6), MSC(0), Port Numbering and Phantom Ports, Message Queues

SPAWN

SYNOPSIS

Launch a background BASIC program.

SYNTAX

SPAWN *filename.expr* {, *port num.var* }

DESCRIPTION

The *filename.expr* is any legal *str.expr* containing the *filename* or *pathname* of a BASIC program.

SPAWN performs a Unix **fork()** creating another process to run the BASIC program. This child process inherits the current environment and current working directory. All channels are closed, and no **COM** or **CHAIN WRITE** variables may be passed.

SPAWN is simpler than the **PORT** or **CALL $TRXCO** functions to launch a *phantom port* into a BASIC program. It is especially suited for launching background reports, spoolers and other programs

communicated with using **SEND, RECV** or **SIGNAL**.

When the program terminates to *command mode* or BASIC *program mode* from **STOP**, non-trapped error, **END, CHAIN ""**, or **SYSTEM 0/1**, the process terminates releasing the *port*.

SPAWN locates an unused *port number* scanning backward from the value of the environment variable **MAXPORT** (default 999).

The optional *port num.var* is returned with the *port number* assigned to the background program. **SEND** and **SIGNAL**, as well as **CALL $TRXCO** and **PORT** statements may be used to communicate with a *port* initiated by **SPAWN** as long as the running program is not terminated to *command mode* or BASIC *program mode*.

EXAMPLES

```
SPAWN "1/SPOOLER"

SPAWN A$,K ! Start program, get port number
```

ERRORS

System is out of Channels - Notify Manager (No available port located)

See also

CALL $TRXCO, SEND, RECV, PORT, SIGNAL, Port Numbering and Phantom Ports, **MAXPORT**

STOP

SYNOPSIS

Terminate program into DEBUG mode.

SYNTAX

STOP {*str.expr*}

DESCRIPTION

The **STOP** statement terminates a running program and is functionally identical to the **SUSPEND** statement.

str.expr is an optional string expression to be displayed.

The **STOP** statement terminates a program and returns the user to BASIC *program mode*.

The **STOP** statement is usually used to indicate an error condition or some other abnormal mode of program termination. A **STOP** statement, non-trapped **[ESC]** or **[EOBC]** (usually CTRL+D) causes program execution to cease. The program is left in the partition (unless Supplemental Attributes <E> or <O> are enabled), channels remain open, and variables retain their values. The user is returned to BASIC *program mode* with the message:

```
STOP at statement stn ; sub-stn in: program
```

stn is the statement number containing the **STOP**, *sub-stn* is the statement within the line, and *program* is the *filename* of the current BASIC program.

If the running program was started by **SWAP**, the various levels are displayed:

```
STOP at statement 1400; 1 in: program2

SWAP at statement 2400; 2 in: program1
```

This example indicates that a **STOP** occurred in program2, which was *swapped* to from program1.

Other statements may follow a **STOP** in the program.

EXAMPLES

```
100 STOP

220 STOP "Irrecoverable error, contact support"
```

ERRORS

String Expression must be used here

See also

Supplemental Program Attributes, **END, CHAIN, SYSTEM, SUSPEND**

SUSPEND

SYNOPSYS

Terminate program into DEBUG mode.

SYNTAX

SUSPEND {*str.expr*}

DESCRIPTION

The **SUSPEND** statement is functionally identical to the **STOP** statement.

str.expr is an optional string expression to be displayed.

The **SUSPEND** statement is usually used to indicate an error condition or some other abnormal mode of program termination. A **SUSPEND** statement, non-trapped **[ESC]** or **[EOBC]** (usually CTRL+D) causes program execution to cease. The program is left in the partition (unless Supplemental Attributes <E> or <O> are enabled), channels remain open, and variables retain their values. The user is returned to BASIC *program mode* with the prompt:

```
SUSPEND at statement stn ; sub-stn in: program
```

stn is the statement number containing the **SUSPEND**, *sub-stn* is the

statement within the line, and *program* is the *filename* of the current BASIC program.

If the running program was started by **SWAP**, the various levels are displayed:

```
SUSPEND at statement 1400; 1 in: program2

SWAP at statement 2400; 2 in: program1
```

This example indicates that a **SUSPEND** occurred in program2, which was *swapped* to from program1.

Other statements may follow a **SUSPEND** in the program.

EXAMPLES

```
100 SUSPEND

220 SUSPEND "irrecoverable error, contact support"
```

ERRORS

String Expression must be used here

See also

Supplemental Program Attributes, **END, CHAIN, SYSTEM, STOP**

SWAP

SYNOPSIS

Pause & execute another BASIC program.

SYNTAX

SWAP {*mode*,} *filename.expr*

DESCRIPTION

mode is any *num.expr* which, after evaluation is truncated to an integer to select channel and common variable pass-along into the **SWAP** program. If *mode* is omitted, mode 2 is assumed.

SWAP suspends execution of the current program, saves all open channels and variables, using the Unix **fork()** function to create another identical UniBasic process. This *child* (swapped) process inherits the current environment, variables, open channels, and current working directory from the *parent* (calling process).

The selected *filename.expr* is loaded following the same rules as **CHAIN**. Common variables declared using **COM** or **CHAIN WRITE** statements following the **SWAP** statement, and open channels passed to the child process are processed according to the *mode* as follows:

Mode	**Function Performed**
0	Close all open files in the child. Do not pass any common

variables, i.e. ignore **COM** and **CHAIN WRITE**.

1 Pass all open channels to the child, and process the common variables according to the rules for **COM** or **CHAIN WRITE**.

2 (default) Close all open files for the child, but process any common variables according to the rules for **COM** or **CHAIN WRITE**.

The *parent* is the initial process (UniBasic) launched from the shell or automatically during login to the system. It is also the name given to a copy that is currently waiting for a *child* to complete.

The *child* is each identical process created by the SWAP statement using the Unix **fork()**. The *child* inherits a complete copy of the current process including program, variables, open files, current working directory and windows. The *parent* is suspended while the *child* runs. When a *child* terminates, the *parent* continues automatically, unaware of the events of the *child*. To prevent the loss of type-ahead, the *parent* replaces its type-ahead buffer with the actual type-ahead left by the *child* .

A *child* can itself be considered a *parent* if it performs a **SWAP** statement. **SWAP** statements may nest until memory is exhausted, or the Unix Process Table overflows. A unique relationship exists between the *parent* and *child* processes. Variables, File Positions and Window Tracking all flow forward from *parent* to *child*, however no information is passed back to the *parent* upon termination of a child.

Any screen operations performed by a *child* are unknown to the *parent*. If a *child* process is performing screen I/O, the application should make use of Windows. Each *child* process should create and delete a window for its screen I/O. Failure to properly manage process levels performing screen I/O results in an incorrect Window Tracking Map when the *parent* resumes execution.

When a *child* inherits open files, Unix uses the same entries in the system open file table. A *child* can change its copy of the current pointers as well as add or remove locks on records. These operations may confuse the *parent*. Also, since the *child* is a different process, it will be blocked from reading a record locked by the *parent* .

For example, if the *parent* reads 5 records sequentially, the *child* may read 5 additional records in proper order. Upon termination of the *child*, the *parent* may read another 5 records in sequential order. The *parent's* UniBasic channel table is not updated for the operations of the *child*. The **CHF/CHN** functions will not match the Unix pointers for sequential access. If all file access by the *parent* uses a Record/Byte position, there is no need for concern.

When the **SWAP** program terminates using **END, SYSTEM**, or **CHAIN ""**, its process is killed, and the calling program resumes execution at the statement immediately following the **SWAP**. To the caller, it appears as if the **SWAP** statement never occurred.

If a non-trapped **[ESC], [EOBC]** (usually CTRL+D) or **STOP** statement occurs, the swapped program is terminated to BASIC program mode to allow debugging. Execution of a termination statement while in debug mode (**END, SYSTEM,** or **CHAIN ""**), terminates the swap level and resumes execution in the calling program. In debug mode, the **FILES** command displays open channels and **SWAP** levels.

Data may be passed from a swapped program back to the calling program using temporary files, or by placing it into the type-ahead buffer using **CALL $INPBUF**. Data may not be transferred to the calling program using common variables.

Important: a *child* program can communicate with other ports using **CALL 98**, etc., and assumes the same port # as the *parent*. However a *child* cannot create (log-on) a new phantom port because the phantom, being a mirror image of the *child*, exits when its running program exits.

EXAMPLES

```
SWAP "23/PROGRAM3"

SWAP 0,A$
```

ERRORS

File does not exist

Function Argument or Statement Mode out of range

See also

CALL $SWAPF, CHAIN READ, CHAIN WRITE, STOP, END, CHAIN, SPAWN, FILES, HOT-KEY, Using Dynamic Windows, **WINDOW**

SYSTEM

SYNOPSIS

System functions & commands.

SYNTAX

SYSTEM (*mode* {, *parameters*}) {; ...}

DESCRIPTION

mode is any *str.expr* which is to be passed directly to Unix for execution in a sub-shell, or *num.expr* which, after evaluation is truncated to an integer to select an internal special operation.

mode may also be any *num.expr* which, after evaluation is truncated to an integer and used to specify the operation to be performed. Some *modes* require a second *parameter* which is any *num.expr* which, after evaluation is truncated to an integer. The *parameters* are separated by the *mode* using a comma.

Multiple **SYSTEM** *modes* may be invoked separating each with a semicolon.

When *mode* is any *str.expr*, it is passed directly to Unix. This Unix command can be used to launch another application, or perform a system commands such as **mv**, **cp**, etc. If an optional status var follows, the stat_loc that is returned from wait(2) (see Unix Programming Reference Manual) is stored. Any changes to a terminal's stty settings are restored to its original setting upon return to UniBasic.

Following execution of the system command by the operating system, the program resumes operation.

If the system command performs any output, your screen will be compromised unless a new Window was opened prior to, and closed after, the **SYSTEM** command.

Mode	**Operation Performed**
0	Terminate a UniBasic session (**BYE** command). You may also terminate other users by including a *port number* as an additional *parameter*. The general form: **SYSTEM 0,N** terminates port **N**.
1	Clear the port's program partition (issue a **NEW** command), and stop the program.
4	Un-assign all non-common variables. All dimensioned *str.vars, array.vars*, and *mat.vars*, as well as simple *num.vars* are unassigned. This allows re-dimensioning of partition space as long as all variables to be used are re-assigned.
5	Un-assign all variables. Same effect as **SYSTEM 4**, except common variables (**COM** and **CHAIN WRITE**) are also affected.
6	Select baud rate. This mode requires the special form: **SYSTEM 6**, **N** where **N** is a new baud rate.
8	Enable terminal echo. Each character input will be echoed by the system to the terminal.
9	Disable terminal echo. Each character input is received by the system, but not echoed to the terminal. This feature allows for password or other secretive input.
12	Enable Tab mode. Not supported at this time.
13	Disable Tab mode. Not supported at this time.
14	Enable Binary Input mode. All characters input are directly accepted as data. This includes **[EOL]** (usually return), requiring the use of character limited **INPUT**. While in Binary Input mode, **ASC** returns data in true internal format without high-bit toggling.

15 Disable Binary Input mode. Normal character processing is resumed.

16 Enable Binary Output mode. Each character output will be full 8-bit data with no parity generation by the system. Every possible character from 0 thru 377_8 may be output. While in Binary Output mode, **CHR** will not toggle the supplied argument to provide true Binary output capability.

17 Disable Binary Output mode.

18 Enable limited IRIS compatibility mode. Certain statements are affected by **SYSTEM 18** mode, causing them to function in an IRIS-compatible fashion. This mode affects system operation less, and is therefore less IRIS-compatible, than setting **BASICMODE=IRIS**.

19 Disable limited IRIS compatibility mode.

20 Enable Trace mode. See Trace Mode.

21 Disable Trace mode.

22 Set a program breakpoint. See Program Breakpoints.

23 Clear a program breakpoint.

26 Automatic limited input. Causes character limited input to terminate when the specified number of characters have been entered. Affects **INPUT** statement.

27 Disable Automatic limited input. Causes character limited input to require an **[EOL]** (usually return) to be entered, even after the specified limit has been reached. Entry of each extra character sounds the terminal bell until **[EOL]** is entered.

28 Get value of Environment Variable. This function requires the special form: **SYSTEM 28,** ***str.var*** where *str.var* initially contains the name of an environment variable. If found, its value is overwritten in the string, otherwise the *str.var* is unchanged. Note: Most environment variables are in upper-case characters.

34 (Release 8.1.8) Enables a mode that converts all lowercase input characters to uppercase characters. This mode is carried across **CHAIN** and **SWAP** statements.

35 (Release 8.1.8) Disables the mode that converts lowercase to uppercase characters.

100 (Release 9.1) Add new keys to the current key list, modify existing keys or delete the key list. Key names are case insensitive and can contain any printable ASCII character except for single or double quotation marks. Key names beginning with "SYS_" are reserved for special purposes and must not be used for application key names.

Format of the statement is
```
SYSTEM 100,"keyname","passphrase","cipher"
```
where
"keyname" is a string expression that specifies the key name.
"passphrase" is a string expression that specifies the passphrase.
"cipher" is the name of the encryption algorithm. The recommended ciphers are "AES-128" and "AES-256".
Keys can be deleted from the current key list by using:
```
SYSTEM 100,"keyname","",""
```
The entire current key list can be deleted by using "" as the key name.
```
SYSTEM 100,"","",""
```

101 (Release 9.1) Is used to generate a key file string image that contains all of the keys from the current key list (except those that begin with "SYS_"). The format is:
```
SYSTEM 101,S$
```

"S$" is any string variable.

102 (Release 9.1) Restores encryption keys from the key file (see **UBKEYFILE**).

Each port is returned to its normal operational modes (8, 13, 15, 17, 19, 21, 23, and 26) when a program is completed or aborted. **CHAIN, SWAP**, or **SPAWN** statements set modes 19, 21, and 23, clearing limited IRIS mode, trace and breakpoint.

EXAMPLES

```
SYSTEM "ls -l >filename"

SYSTEM 14;16;
```

ERRORS

Function Argument or Statement Mode out of range

See also

TRACE, Trace Mode, Program Breakpoints, **BYE, NEW**, IO Mnemonics, Using Dynamic Windows

TRACE

SYNOPSIS

Enable statement trace debugging.

SYNTAX

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

DESCRIPTION

Trace mode is used when it is desirable to observe the statement number program flow without performing single steps. **SYSTEM 20** or **TRACE ON** enables tracing; **SYSTEM 21** or **TRACE OFF** turns trace off. These statements may be used in *immediate mode*, or imbedded within specific code segments of a program. For each statement executed, the statement number *stn* and sub-statement number *sub-stn* (statements on the same BASIC line) is printed.

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. The following information is output during *trace mode*:

```
TR - statement number ; sub-statement number          (BITS)

[statement number]                                    (IRIS)
```

In BITS mode, “TR -" indicates trace mode is enabled and the next stn and sub-stn to be executed are displayed. In IRIS mode, a new-line is performed, and the stn only is displayed within [ ]. The execution of the statement then proceeds. Output from a **PRINT** is displayed following the trace information.

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

EXAMPLES

```
TRACE ON

TRACE OFF

TRACE ON #5
```

ERRORS

Syntax error

Channel not opened

File is Write-protected

See also

SYSTEM 20, SYSTEM 21

UNIT

SYNOPSIS

Access & control current working directory.

SYNTAX

UNIT *directory, mode* {, *return*}

DESCRIPTION

directory is any *str.expr* containing or returning a Unix *pathname*.

mode is any *num.expr* which, after evaluation is truncated to an integer to select the desired operation.

The optional *return* is any *num.var* used to return information about a directory for certain *modes*.

Mode	Operation Performed
3	Returns the number of the available blocks in the specified file system.
5	Change the current working directory to the selected *pathname*.

An error occurs if the selected *pathname* is illegal or protected.

EXAMPLES

```
A$="/usr/ub/2" \ UNIT A$,5

UNIT "sys",3,B ! B returns number of available blocks
```

ERRORS

Illegal Pack or Filename

See also

Directories and Pathnames

UNLOCK

SYNOPSIS

Unlock any locked records on a channel.

SYNTAX

UNLOCK # *channel* {, # *channel* ...}

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select a *channel* number.

Any record locked by your program on the specified *channel* becomes unlocked. No error is generated if no record has been locked. A record locked by another user cannot be unlocked.

Generally, **UNLOCK** is only used in special circumstances, such as having one file open on two *channels*. In this case, **UNLOCK** can be used to prevent the program from locking itself out of a record.

In IRIS applications, the statement **WRITE #** *channel* ;; is identical to *UNLOCK #*.

EXAMPLES

```
UNLOCK #5, #K, #K+1
```

ERRORS

Channel Not Opened

See also

WRITE #, Record Locking

WINDOW

SYNOPSIS

Maintain Dynamic Windows.

SYNTAX

WINDOW (**ON | OFF | CLOSE | CLEAR**)

WINDOW (**OPEN | MODIFY**) *parameters following on next line:*

@*ulc,ulr*; (**SIZE** *ncol,nrow*; | **TO** @*lrc, lrr*;) {**USING** *str.expr*}

DESCRIPTION

@ specifies a *crt.expr* in the form of a Cursor Address. *ulc* is any *num.expr* which, after evaluation is truncated to an integer to select the Upper Left Column for the Window. *ulr* is any *num.expr* which, after evaluation is truncated to an integer to select the Upper Left Row. Following the *crt.expr* must be a semicolon.

SIZE selects the size of a Window in columns and rows. **TO** specifies the size using a *crt.expr* in the form of a Cursor Address of the last character position in the Window. Either form may be used. If **SIZE** is used, *ncol* is any *num.expr* which, after evaluation is truncated to an integer to select the number of columns. *nrow* is any *num.expr* which, after evaluation is truncated to an integer to select the number of rows. If **TO** is specified, *lrc* is any *num.expr* which, after evaluation is truncated to an integer to select the Lower Right Column for the Window. *lrr* is any *num.expr* which, after evaluation is truncated to an integer to select the Lower Right Row. Following the *crt.expr* must be a semicolon.

The optional **USING** *str.expr* is any string expression to be centered and printed as the title of a Window. The size must be less than the number of columns in the Window, or it is truncated. The inclusion of **USING** specifies that a graphical border is to be placed around the Window. The *str.expr* may be a null-string for a box without heading. The specification of a graphical border reduces the usable space in the Window by one row, and column on the top, bottom and each side.

Note: Before using Windows, the default term file must be defined correctly for number of rows, columns, and mnemonics. The environment

variable **WINDOWS** must be defined for the total number of open windows to be used by the application. See Using Dynamic Windows and Terminal Translation $TERM files.

Window Tracking is normally off so that console commands and Unix functions operate on a full screen. Whenever a program terminates, Window Tracking is turned off. If a program is terminated by **[ESC], [EOBC], STOP**, or Breakpoint, debugging is permitted and Windows remain open, otherwise all Windows are cleared. In either case, Tracking is disabled and screen data may be corrupted.

WINDOW ON enables Window Tracking and should precede any other **WINDOW** function. The Window Tracking Map is initialized by clearing the screen. Subsequent **WINDOW ON** statements are ignored. By default, a **WINDOW ON** is performed automatically whenever a clear-screen is sent in *run mode* on PC/ANSI monitors (*crt_type* :23) to simulate protected fields.

WINDOW OFF temporarily disables Window Tracking. Further screen operations are not updated in the Window Tracking Map, and access outside the current Window is allowed. If Window Tracking was on and protected fields are used, they won't be protected once Window Tracking is turned off. **WINDOW OFF** may be used to improve screen performance in programs not using Windows. It is a good idea to combine a clear-screen operation with a change in Window Tracking status.

WINDOW OFF and **ON** may also be used when secondary Windows (other than the first full-screen) are opened, and access to the full screen is desired. When Tracking is turned off, cursor access is to the full screen. When Tracking is again turned on, the cursor is re-positioned to the last tracked position. Turning Tracking off to modify data outside the screen should be limited to the display of errors or messages in a common area. Any other screen modification is not tracked and, if **SWAP** is used, the *parent* is unaware of these changes.

WINDOW OPEN creates a new Window with the supplied *parameters*. If Tracking is not on, an implied **WINDOW ON** is performed. All *crt.expr* are relative to upper left corner of a Window and all data is forced within its boundaries. **MSC(33)** and **MSC(34)** will reflect the inside limits of the Window, and **MSC(42)** will be incremented to reflect the number of open Windows. Scrolling occurs only on the bottom line of the window.

WINDOW MODIFY is used to change the size of the current Window based upon the supplied *parameters*. Functions **MSC(33)** and **MSC(34)** are updated to reflect the current size. The size of a Window may be changed as many times as desired but it cannot extend beyond the original *parameters* specified to **WINDOW OPEN**. If the Window must be enlarged, perform a **WINDOW CLOSE**, followed by another **WINDOW OPEN**. **WINDOW MODIFY** may be used to create your own borders, to modify the

border created by **WINDOW OPEN**, or implement a series of panes inside a Window that can be accessed randomly.

WINDOW MODIFY merely redefines the writable region inside a window. The window itself is not actually closed and re-opened. No underlying data is revealed or hidden by this statement.

WINDOW CLOSE deletes the current Window repainting the original underlying data. **MSC(33)** and **MSC(34)** now reflect the size of the previous Window and **MSC(42)** is decremented. A Window must always be deleted at the same *parent* / *child* level it was created. For example, you perform a **WINDOW OPEN** in program A, then **CHAIN** to program B, which in turn performs a **SWAP** or **[Hot-Key]** swap to program C (a *child* of B). If program C opens any windows, then **WINDOW CLOSE** should be performed before returning control to program B.

WINDOW CLEAR clears all Windows back to Window Zero and clears the screen. Underlying data from each opened Window is not displayed.

Note: Because of the nature of the *parent* and *child* mentioned above, always delete a Window from the same program it was created to correctly maintain the Window Tracking Map.

EXAMPLES

```
WINDOW ON

WINDOW OPEN @5,5; TO @60,20; USING "Help"

WINDOW OPEN @0,0; SIZE 80,24;

WINDOW MODIFY @7,7 TO @62,18;

WINDOW OFF

WINDOW CLOSE

WINDOW CLEAR
```

ERRORS

No term file loaded

WINDOWS Environment Variable not defined or zero

CRT X,Y coordinate out of range

No more Windows to Close; check MSC(4)

Window Tracking is not on

See also

Using Dynamic Windows, Hot-Key, **CALL $SWAPF, SWAP**

WRITE

SYNOPSIS

Write array, matrix or string from a channel.

SYNTAX

WRITE *#chn.expr*; *var list...*{;}

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file from which to read data.

WRITE # transfers data from *any var, mat.var, array.var* or *str.var* to the file opened on the selected *chn.expr*.

If the variable in the list is an *array.var* or *mat.var*, only the first element ([0] or [0,0]) is written. *Subscripts* may be used to select any individual element to be transferred. The number of bytes transferred is based upon the variable **DIM**ensioned size. The transfer is performed according the rules for a *num.var*.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by *subscripts*. When no *subscript*, or single *subscript*, is specified, IRIS programs increment the total number of bytes transferred to account for an extra null byte. In BITS applications, no increment is performed and the entire supplied size is transferred including zero-bytes.

The optional semicolon (**;**) terminator is used by IRIS applications to eliminate the automatic record-lock applied to the supplied *record* in the *chn.expr*. BITS applications utilize **RDLOCK #** for operations with locking, and **READ #** for non-locking transfers.

In IRIS applications, the statement **WRITE #** channel ;; is identical to **UNLOCK #**.

If the running program is an IRIS program, the following steps are performed prior to transfer:

1. If the variable to be transferred is a *num.var, array.var*, or *mat.var*, the supplied (or current) *byte displacement* is rounded up to an even byte position within the file.
2. If a full *str.var* is supplied (single or no *subscript*), its size is incremented by one to account for an extra null byte. If two *subscripts* are supplied, no increment is performed. From this, the maximum size is determined. The *str.var* is scanned for the first zero-byte. Data is written from the string, stopping at the maximum size or following the first zero-byte. If the write terminated prior to the maximum size, the file pointers are adjusted as if the maximum size was written. Finally, if the transfer is from a text file, an error is generated if any *num.var* is supplied.

If the transfer is to a Formatted Item file, the item type may be String or Binary for any *str.var* in the list, and Binary or Numeric for any *num.var, array.var*, or *mat.var*. The *byte displacement* specifies the starting item for the transfer. If not specified, item zero is assumed. No conversion takes place during the transfer of a binary item. It is the program's responsibility to maintain the correct precisions of numerics being written to the file.

If the transfer is to a Contiguous or Tree-structured Data file, the *byte displacement* specifies the starting byte within the supplied record. Zero is assumed if no *byte displacement* is given, and IRIS programs round up the *byte displacement* if odd on a numeric variable transfer.

If the transfer is to a text file (IRIS Programs only), data is written up to and including the first zero-byte in the string. The file position is then decremented pointing at the zero-byte for subsequent write operations.

Each item transferred causes the *byte displacement* to be incremented by the adjusted byte size of the item in the *var.list*. Strings are sized by the algorithm (INT((d+1)/2)*2), where *d* is the **DIM**ensioned or subscripted size. *num.vars, arrays* and *matrices* are sized as: (R+1) * (C+1) * (size of P) where R is the number of rows, C is the number of columns, and P is the number of bytes occupied by precision P.

EXAMPLES

```
WRITE #3,R1,100;A,B$,C[12]

WRITE #C,R;A$
```

ERRORS

Data does not match item specification and cannot be converted

Selected channel is not open

Selected record is locked

File is Write Protected

See also

Numeric, Array and Matrix Variables, Channel Expression, **READ#, MAT WRITE#, WRLOCK#**, Numeric Data, Numeric Variable Precision, Formatted Item Files, Contiguous Files, Text Files

WRLOCK

SYNOPSIS

Write and unconditionally lock a record.

SYNTAX

WRLOCK *#chn.expr*; *var.list...*

DESCRIPTION

The *# chn.expr* is any legal channel expression selecting an open file to read data from.

WRLOCK # transfers data from any *var, mat.var, array.var* or *str.var* into the file opened on *chn.expr*.

If the variable in the list is an *array.var*, an optional *subscript1* and *subscript2* may be specified. If given, these are evaluated, truncated to integer and used to select a single element. If no *subscripts* are supplied, only the first element is transferred.

If the variable in the list is a simple *num.var*, the transfer size is controlled by the **DIM**ensioned size and precision.

If the variable in the list is a *str.var*, its size may be controlled by *subscripts*. All characters are transferred including zero-bytes.

WRLOCK transfers data and unconditionally locks the record. The data record remains locked until a non-locking operation is performed by that same program to the same channel. While a record is locked, other users will be unable to access the record.

WRLOCK# is identical to **WRITE#** omitting the trailing semicolon.

See the **WRITE#** statement for details on the transfer to different files.

EXAMPLES

```
WRLOCK #3,R1,100;A

WRLOCK #C,R;A$
```

ERRORS

File is Write Protected

Selected Record is Locked

Channel is not Opened

See also

WRITE#, RDLOCK#

WRREL

SYNOPSIS

Write a relative 512-byte block to a file.

SYNTAX

WRREL *# chn.expr*; *str.var*

DESCRIPTION

 # chn.expr is any legal channel expression that selects an open file to which to write data. The *chn.expr* must include a *record* which defines the relative block to write within the file. The *byte displacement* and

time-out expressions are ignored and unnecessary.

The *str.var* is any string variable **DIM**ensioned at least 512 bytes. A starting *subscript* may be supplied as long as the **DIM**ensioned size is at least 512 bytes larger than the supplied *subscript*.

WRREL uses the supplied *record* as a relative 512 byte block pointer into the file. For example, *record 0* specifies the first 512 bytes in the file, *record 1*, the second 512 bytes, etc.

Record -1 may be used to write the first 512 bytes of the file. This includes the header and possibly part of *record 0*. Some headers (of formatted item files) may be larger than 512 bytes and may not be written in entirety. To retrieve header information in a truly machine independent fashion, it is recommended that **CALL 127** be used to unpack the information. **WRREL #** of *record -1* is used to change header information by conversion and other utilities.

WRREL is generally used to copy files or otherwise write portions of files not accessible with a normal **WRITE#** statement. Processing of the data is left completely up to the user.

EXAMPLES

```
WRREL #7,K;A$ ! WRITE A BLOCK

WRREL #7,K+1;A$[513] ! APPEND A SECOND BLOCK
```

ERRORS

Channel Not Opened

Illegal Record or End of File

Write Protected File

See also

RDREL#

User CALLS

This section documents the standard "C" language **CALL** statements included with all versions of UniBasic. Some systems may include additional **CALL** statements added by your supplier for specific applications.

To add or change User calls requires the UniBasic Development Package. The development system includes a README file explaining existing **CALL**s and how to add and remove **CALL**s.

CALL $ATOE

SYNOPSIS

Convert ASCII to EBCDIC.

SYNTAX

CALL (**77** | **$ATOE**), *str.var*

DESCRIPTION

CALL $ATOE converts a supplied ASCII string to EBCDIC.

EXAMPLES

```
CALL $ATOE, A$

CALL 77, A$
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

CALL, ASCII Characters, **CALL $ETOA, CALL 53**

CALL $AVPORT

SYNOPSIS

Find available port number.

SYNTAX

CALL $AVPORT, *port* {, *starting* {, *ending* } }

DESCRIPTION

port is any *num.var* to return the first available *port number*. -1 is returned when no available *port number* is found.

starting is any optional *num.expr* which, after evaluation, is truncated to an integer, and used to specify the first *port number* to search. If omitted, the search begins at Port 0.

ending is any *num.expr* which, after evaluation, is truncated to an integer, and used to specify the last *port number* to search. If omitted, the maximum *port number* as defined by the environment variable **MAXPORT** (default 999) is assumed. An *ending* expression can only be specified if a *starting* value was given.

EXAMPLES

```
CALL $AVPORT, G !Get first Port

CALL $AVPORT, G, 32 !Get available phantom
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

CALL, PORT, PORTS, MAXPORT

CALL $CALLSTAT

SYNOPSIS

Get name of CALLing program.

SYNTAX

CALL $CALLSTAT, *str.var*

DESCRIPTION

Str.var is a string variable the receives the name of the program that **CALL**ed the current program or "" if the current program was not started by a **CALL** statement.

EXAMPLE

```
CALL $CALLSTAT,F$
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

none

CALL $CKSUM

SYNOPSIS

Calculate checksum on a file.

SYNTAX

CALL $CKSUM, *filename, start, end, result* {, *status*}

DESCRIPTION

filename is any *str.var* containing a *filename* or *pathname* to a file to which you have read-permission. The file is opened and read to compute a checksum on its data. The checksum computed is machine independent.

start is any *num.expr* which, after evaluation, is truncated to an integer and used to specify the starting word address in the file for the computation. Zero specifies the start of the file.

end is any *num.expr* which, after evaluation, is truncated to an integer and used to specify the last word address in the file for the

computation. (-1) specifies the current physical end of the file.

result is returned with the computed checksum.

status is optionally returned with a completion status as determined by the CALL. The following exception status is reported:

Status	Description
0	Operation was successful
1	*filename* is not a string
3	*start* is negative
5	*end* is negative (cannot checksum memory)
6	*start* is larger than end.
8	*filename* not found

EXAMPLES

```
CALL $CKSUM, "program", 0, -1, A, B
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

none

CALL $CLU

SYNOPSIS

Change the current logical unit.

SYNTAX

CALL $CLU, *lu.num* {,*status*}

DESCRIPTION

lu.num is a value specified for logical unit number, pack name, or Unix directory name. If *lu.num* is passed as -1, change the current logical unit to the default working directory.

status is optionally returned with a completion status as determined by the **CALL**. The following exception status is reported:

Status	Description
0	Operation was successful
1	Invalid logical unit number
2	Logical unit number not found

EXAMPLE

```
CALL $CLU,1,S

CALL $CLU,N
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

CLU Command

CALL $DATE

SYNOPSIS

Verify and reformat a date.

SYNTAX

CALL $DATE, *source, destination, length, status*

DESCRIPTION

source is any *str.expr* containing a date in the form of MMYY, MMDDYY, or MMDDYYYY.

destination is any *str.expr* containing a returned date in the form of YYMM, YYMMDD, or YYYYMMDD, depending on the length. If the environment variable **EUROPEAN** is set, the form of YYDDMM or YYYYDDMM is returned, depending on the length.

length is any *num.expr* for the length of the destination date. Valid lengths are 4, 6, and 8.

status is an exception value returned to caller providing completion *status* of the desired operation. A *status* value of zero (0) indicates valid date. A *status* value of one (1) indicates an invalid date.

EXAMPLE

```
CALL $DATE,S$,D$,L,E
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

CALL 24

CALL $ECHO

SYNOPSIS

Set or clear terminal echo.

SYNTAX

CALL (**78 | $ECHO**), *mode*

DESCRIPTION

mode is any *num.expr* which, after evaluation, is truncated to an integer and used to select the operation for **$ECHO**.

Mode	Function Performed
0	Disable terminal echo.
1	Enable terminal echo.
2	Toggle terminal echo.

Terminal echo is the process whereby each character entered to the terminal is displayed on the screen. When echo is disabled, input is still processed by the system, but is not visible on the screen.

EXAMPLES

```
CALL $ECHO,0 !Turn off terminal to get password

CALL $ECHO,1 !Re-enable echo

CALL 78, 1
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

IRIS IO Mnemonics, **SYSTEM 8/9, CALL 44**

CALL $ENV

SYNOPSIS

Change the value of an environment variable.

SYNTAX

CALL $ENV , *varname$*, *value$*

DESCRIPTION

varname$ is the variable name to be changed.

value$* is the new value to be given *varname$.

CALL $ENV places the string *varname$ = value$* into the environment of your process. Any environment variables can be added or changed, with the exception of the following UniBasic

parameters which are not changeable:

ALTCALL	BCDVARS	FORNEXTNEST	GOSUBNEST
INPUTSIZE	ISAMBUFS	ISAMFILES	ISAMSECT
LUST	NUMLINES	PORT	PORTS
PROGSIZE	TERM	VARSIZE	WINDOWS

EXAMPLES

```
CALL $ENV,"SCOPEPROMPT","@"
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

Installing UniBasic, UniBasic Environment Variables

CALL $ETOA

SYNOPSIS

Convert EBCDIC to ASCII.

SYNTAX

CALL (**76 | $ETOA**), *str.var*

DESCRIPTION

The *str.var* specifies a string of EBCDIC characters to be converted to ASCII.

EXAMPLES

```
CALL $ETOA,A$ !Convert the string

CALL 76,A$
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

CALL $ATOE, CALL 53

CALL $FINDF

SYNOPSIS

Lookup a file on the system.

SYNTAX

CALL (**96 | $FINDF**), *filename*, *status*

DESCRIPTION

filename is any *str.expr* containing a *filename* or full *pathname* to lookup.

status is any *num.var* used to return a flag. If the supplied *filename* is found, a non-zero *status* is returned. A zero indicates that the supplied *filename* does not exist.

EXAMPLES

```
CALL $FINDF, "23/filename", K

CALL $FINDF, "/usr/bin/UniBasic", j

CALL 96, "14/FILENAME", K
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

LUST, Filenames and Pathnames, **CALL 127, CALL $RDFHD, CALL $RENAME**

CALL $INPBUF

SYNOPSIS

Place data into type-ahead buffer.

SYNTAX

CALL $INPBUF, *str.expr*

DESCRIPTION

The supplied *str.expr* is copied (appended) to the contents of the current type-ahead buffer.

$INPBUF may be used to pass data from a child process back to the parent when using **SWAP** statements or **[Hot-Key]** swapping.

CALL $STRING may be used to drain the contents of the type-ahead buffer.

EXAMPLES

```
CALL $INPBUF, A$ !Copy data to type-ahead
```

```
CALL $INPBUF, A$ + "\215\"
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

WINDOW, Windows and Output Considerations, **SWAP, CALL $STRING**

CALL $LOCK

SYNOPSIS

Lock an opened file.

SYNTAX

CALL $LOCK, *channel*, *mode*, *status*

DESCRIPTION

channel is any *num.expr* which, after evaluation, is truncated to an integer, and used to select an opened data file channel.

mode is any *num.expr* which, after evaluation, is truncated to an integer, and used to specify the operation. A zero value unlocks the file, and non-zero is used to lock the file.

status is any *num.var* used to return a successful or exception status as follows:

Status	Description
0	Operation successful
1	Illegal Channel Number
2	Channel not open
6	File is already Locked
7	File is not locked

$LOCK is similar to **EOPEN**. The selected file is locked to prevent other users from opening the file. **$LOCK** will not provide locks against other users who already have the file opened.

$LOCK is rejected if the file is already locked by another user using **$LOCK** or **EOPEN**.

EXAMPLES

```
CALL $LOCK, 1, 1, E ! lock

CALL $LOCK, 1, 0, E ! unlock
```

ERRORS

Illegal or unimplemented user CALL number or name

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

EOPEN

CALL $LOGIC

SYNOPSIS

Perform Logical Operations.

SYNTAX

CALL (88 | $LOGIC), *operator, variable1, variable2, result*

DESCRIPTION

operator is any *num.expr* which, after evaluation, is truncated to an integer and used to specify the operation for **$LOGIC**:

1 AND

2 OR

3 XOR

4 NOT

variable1 and *variable2* select two identical types of variables to perform an operation upon.

result must be the same type as the supplied *variable1* and *variable2*, and will hold the resulting data from the operation.

If the supplied variables are numeric, they are truncated to unsigned integers (shorts) to perform the operation. String variables are processed a byte at a time until the **DIM**ensioned length of the shortest argument passed is reached.

An *AND* operation results in a 1 bit when the corresponding bit of both variables is 1.

An **OR** operation results in a 1 bit when either of the corresponding bits is 1, or when both are 1.

An **XOR** (exclusive **OR**) results in a 1 bit when only one of the corresponding bits of both variables is 1.

A **NOT** operation only requires *variable1*. *variable2* must be specified for syntactical reasons (use the same variable), but is not used. **NOT** results in a 1 bit if the bit of *variable1* is zero, and results in 0 if the bit is 1.

Entire strings (including zero bytes) can be operated upon using

$LOGIC. To copy a string in its entirety, AND the string to itself. To fully zero fill (zero byte) a string, XOR it with itself.

X	Y	X AND Y	X OR Y	X XOR Y	NOT Y
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

EXAMPLES

```
CALL $LOGIC, 1, A$, A$, B$ ! AND 2 strings

CALL $LOGIC, 1,A[0],32768,J! Is value negative

CALL 88, 1, A$, A$, B$ ! AND 2 strings
```

ERRORS

Illegal or unimplemented user CALL number or name

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

CALL 59

CALL $NCRC32

SYNOPSIS

Calculate 32-bit CRC Checksums.

SYNTAX

CALL $NCRC32, *result*, *string* {,*initialcrc*}

DESCRIPTION

A 32-bit CRC checksum value is calculated for the characters in the *string* variable string and returned in the numeric variable *result*. The result variable should be a 3% or 4% numeric variable to avoid overflow. The checksum includes the entire **DIM**ed size of *string* unless subscripts are used. A checksum can be calculated across multiple strings by specifying the previous accumulated checksum as the optional third parameter *initialcrc*.

EXAMPLES

```
CALL $NCRC32, C, A$[1,10] ! CRC of first 10 characters
```

```
CALL $NCRC32, C, B$, C ! Add CRC of B$ to previous CRC
```

ERRORS

Illegal or unimplemented user CALL number or name

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

CALL $LOGIC

CALL $RDFHD

SYNOPSIS

Read file header information.

SYNTAX

CALL (**97 | $RDFHD**), *lu, rec, file, act, typ, siz, stat, cst, inc, fcd , fla, hdr*

DESCRIPTION

lu may be either a *str.expr* or *num.expr* used to select the logical unit to be searched. If *lu* is a *str.expr*, any Unix *pathname* may be specified.

rec is any *num.var*. It is evaluated, truncated to an integer, and used to select the record number in the directory to be read. Entry zero is always the filename '.' (directory). If the specified entry *rec* is empty, then the next entry is read automatically until a valid entry is located, or the end of the directory is reached. When the end of the directory is reached, *rec* is returned with the value (-1).

Following the **$RDFHD** call, *rec* is incremented by one so that successive calls may be performed without program adjustment. Therefore, the returned *rec* is always one greater than the entry in the directory corresponding to the returned information.

file must be any *str.var* (**DIM**ensioned at least 15 bytes) to receive the name of the file from the directory entry.

act is any *num.var* used to return the Unix user number. This information is not in IRIS format. Only the user's number is returned, not the group number or the privilege.

typ must be any *num.var* used to return the files IRIS type:

Type	Description	
0	Directory	P
1	System program	S

2	BASIC program	B
24	Text file	T
25	Formatted file	F
26	Contiguous file	C
30	Peripheral driver	$

siz must be any *num.var* used to return the files current size in blocks.

stat must be any *num.var* used to return the current files status word. Zero is always returned.

cst must be any *num.var* used to return the current cost in dimes. Zero is always returned.

inc must be any *num.var* used to return the current income in dimes. Zero is always returned.

fcd must be any *num.var* used to return the files creation age expressed in hours since the base system year as returned by the function **SPC(20)**.

fla must be any *num.var* used to return the files last access date expressed in hours since the base system year as returned by the function **SPC(20)**.

hdr must be any *num.var* used to return the files *inode* (Header block) block number.

EXAMPLES

```
CALL $RDFHD,0,R,N$,A,T,S,S1,C,I,D,L,H

IF (R < 0) END ! END OF DIRECTORY

CALL 97,0,R,N$,A,T,S,S1,C,I,D,L,H
```

ERRORS

Illegal or unimplemented user CALL number or name

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

Read Protected File

See also

CALL 127, CALL $RENAME, CALL $FINDF

CALL $RENAME

SYNOPSIS

Rename a File or Program.

SYNTAX

CALL $RENAME, *lu , oldfilename, newfilename, channel, status*

DESCRIPTION

lu is any *num.expr* which, after evaluation is truncated to an integer and used to specify the *logical unit* containing the file to be renamed. To select the current working directory, specify (-1) for *lu*. Any value set in *lu* is used as a default when the supplied *oldfilename* does not contain a directory specifier.

oldfilename is any *str.expr* used to select the filename to be modified. If the filename is in the form *lu/filename*, the supplied name overrides the value passed in *lu*.

newfilename is any *str.expr* used to specify the new filename for the supplied *oldfilename*.

channel is any *num.expr* which is ignored.

status is any *num.var* used to return an exception status from **$RENAME** as follows:

Status	Description
0	Operation was successful, file renamed.
1	Operation was not successful.

$RENAME uses the Unix *mv* command.

EXAMPLES

```
CALL $RENAME,1,"3/FILENAME","3/FILE1",99,A
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

MODIFY, CALL $FINDF, CALL $RDFHD

CALL $STRING

SYNOPSIS

Miscellaneous string functions.

SYNTAX

(a) CALL (**82 | $STRING**), *mode, string*

(b) CALL (**82 | $STRING**), *mode, string, number*

(c) CALL (**82 | $STRING**), *mode, number, string*

DESCRIPTION

mode is any *num.expr* which, after evaluation is truncated to an integer to specify the operation of **$STRING**.

Mode	Format	Operation Performed
1	a	Convert all characters to upper-case.
2	a	Convert all characters to lower-case.
3	b	Convert single character to ASCII.
4	c	Convert ASCII value to single character.
5	a	Read the Input/Output buffer.
6	b	Convert 2-characters to binary.
7	c	Convert binary to 2 ASCII characters.

string is any *str.var* if the resulting operation returns string data, or any *str.expr* if the resulting operation returns numeric information.

number is any *num.var* if the resulting operation returns numeric data, or any *num.expr* if the resulting operation returns string information.

During case conversion, *modes* 1 and 2, only alphabetic letters are modified. All other characters remain unchanged.

mode 3 converts the single character pointed to by the supplied subscripted string to an ASCII value between 0 and 255, and returns the value in *number*. The value returned is toggled from internal to IRIS 8-bit format. The application should utilize IRIS style ASCII (above 128) for printable characters.

mode 4 converts the supplied *number* into an ASCII character into the supplied position in the *string*. If the value is greater than 255, the modulus 256 value is converted (x % 256). The value is toggled from IRIS to internal format. The character at the position immediately following the specified position in string is zeroed.

mode 5 is used to read the contents of the terminals Input/Output buffer (Type-ahead data placed into the buffer by **CALL $INPBUF** is not accessible). All characters up to the first **[EOL]** (usually return) are placed into string. This option permits a program to read parameters on the same line as the program name, such as:

```
#RUN REPORT 132 DISPLAY
```

CALL $STRING must precede any **PRINT** or **INPUT** statements within the program, or the contents of the buffer will have been modified.

mode 6 is used to convert 2 adjacent characters at the starting position in string to a 16-bit integer returned in *number*. The formula used is:

First character * 256 + second character

mode 7 converts *number* into 2 ASCII characters using the formula: First char = *number* / 256, and second char = *number* % 256. No additional null character is stored.

EXAMPLES

```
CALL $STRING,5,A$ ! Read Buffer

CALL 82,5,A$ ! Read Buffer
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

CALL 29, CALL 30, CALL 43, CALL 44, CALL 56, CALL 57, CALL 60

CALL $SWAPF

SYNOPSIS

Control Hot-Key swapping.

SYNTAX

CALL $SWAPF, *mode* {, *executive program name*}

DESCRIPTION

mode is any *num.expr* which, after evaluation, is truncated to an integer to select the function performed whenever the **[Hot-Key]** is pressed during **INPUT**. Pressing a **[Hot-Key]** has no effect until an **INPUT** statement is reached.

Mode	Description
0	Disable the Dynamic [Hot-Key] operation.
1	SWAP on Dynamic [Hot-Key] with channels OPEN with normal common variables as contained in COM statements.
2	SWAP on Dynamic [Hot-Key] with normal common variables as contained in COM statements.
3	SWAP on Dynamic [Hot-Key] with channels OPEN and no common variables.

The optional *executive program name* is any *str.expr* defining a program to **SWAP** to whenever the **[Hot-Key]** is pressed, and the *mode* is non-zero. This can be any BASIC program *pathname* up to 62 characters in length.

The default values assigned by the system, if a **CALL $SWAPF** is not issued, is *mode* 1, executive program name is *sys/exec*.

An error is generated if a **[Hot-Key]** is pressed and the specified *executive program name* does not exist.

EXAMPLES

```
CALL $SWAPF,0 !Disable Hot-key this program
```

```
CALL $SWAPF,2,"AR.CUST" ! To Cust maint, no files
```

ERRORS

Error detected in/by user CALL routine

See also

WINDOW, [HOT-KEY], Windows and Output Considerations

CALL $TIME

SYNOPSIS

Get date and time.

SYNTAX

CALL (**99 / $TIME**), *string*

DESCRIPTION

string is any *str.var*, **DIM**ensioned at least 22 bytes, to return the current date and time using the Unix function **localtime()**. If you are receiving an incorrect time, check your environment to be sure that you have included the **TZ** (Time-zone) environment variable if required. Please refer to your system documentation for information on the function **localtime**.

CALL $TIME may not be used to reset the system time. That function must be performed using the Unix **date** command from the root password.

Date format returned is:

```
Mon dd, year HH:MM:SS  IRIS applications

dd Mon Year HH:MM:SS   BITS applications.
```

EXAMPLES

```
CALL $TIME, T$ \ PRINT T$

CALL 99, A$
```

ERRORS

Error detected in/by user CALL routine

See also

MSF(0), MSF(3)

CALL $TRXCO

SYNOPSIS

Phantom port control.

SYNTAX

CALL (98 | $TRXCO), *port, command*, {, *status* {, *priority* }}

DESCRIPTION

port is any *num.expr* which, after evaluation is truncated to an integer and used to select the *port number* for this operation.

command is any *str.expr* which selects a command to be sent to the specified *port*. The supplied *command* is copied into the specified *port's* type-ahead buffer to be processed the next time *port* is awaiting input. The *command* may be any system command or prompt response for a running program. Multiple commands, separated by \215\ may be included in the *command* string.

The optional *status* is an exception value returned to the caller providing completion status of the desired operation:

Status	Description
0	Successful operation; command transmitted.
1	*port* is not a numeric expression.
2	Specified *port* is out of range.
3	Specified *port* is not running UniBasic.
4	Specified *port* is the user's own *port*.
5	*command* is not a valid *str.expr*.
6	unix fork() operation failed, or *port* is not ready for input.
7	Specified *port* has input already in progress.

The optional *priority* is any *num.expr* which, after evaluation is truncated to an integer and used as the system priority for the *command* transmitted. The valid range is from 1 to 7. The supplied value is converted to a Unix value for the **nice()** function changing the process priority.

$TRXCO begins by attempting to attach the *port*. If the *port* is already running UniBasic, the *command* is copied into the *port's* type-ahead buffer. A carriage return is appended to the *string* supplied.

If the *port* is not currently running a UniBasic process, a background process is created as the supplied *port number*. It assumes the callers login, *.profile* configurations, environment and current working directory. It then becomes a unique process linked to the supplied *port* number. This port is then available for **CALL $TRXCO** commands, P**ORT, SEND, RECV**, and **SIGNAL** statements from any other UniBasic user as well as the program performing the initial **CALL $TRXCO**.

When sending commands to a *port* which is connected to a terminal and keyboard, you must ensure that *port* is within UniBasic before sending commands. Otherwise, a *phantom port* is created for the

supplied *port* number. If a user later attempts entry into UniBasic on a terminal designated as the same *port*, entry will be rejected.

If the program is an IRIS program and the *command* consists of a single \334\ character, the current job on the *port* is logged off. If any other character, such as \215\ follows the \334\, the current job on the *port* is aborted and the port is placed in *command mode*. A \334\ always aborts any running command on the *port* even if the *port* has Error or Escape branching in effect.

Always pause at least 2 seconds between subsequent **$TRXCO** calls to the same or different ports. This permits the receiving *port* time to respond and avoids an overflow of the inter-process communication buffer.

It is impossible to create a phantom port from a child program. See the **SWAP** statement for details.

EXAMPLES

```
A$="LIBR [$LPT] 1/ @ ^ \215\BYE\"

CALL $TRXCO,10,A$,E,2 !LIBR Low priority

IF E STOP ! Error trying to start LIBR

CALL 98,10,A$,E,2 !LIBR Low priority
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

PORT, Port Numbering and Phantom Ports, Environment Variables: **PORTS & PORT** Environment Variables, Launching UniBasic Ports at Startup, **CALL $AVPORT**

CALL $VOLLINK

SYNOPSIS

Create a Polyfile volume.

SYNTAX

CALL (91 | $VOLLINK), *channel, master chan, vol, stat, param*

DESCRIPTION

channel is any *num.expr* which, after evaluation is truncated to an integer and used to select an open channel containing a built contiguous file to be converted into a polyfile volume.

If *channel* is negative, only parameters are returned in the *param* array.

master chan may be any *num.expr* which, after evaluation is truncated to an integer and used to select an open channel containing the master volume of a polyfile.

vol may be any *num.expr* which, after evaluation is truncated to an integer and used to select an operation:

Volume	Operation
= 0	Create the *vol* opened on *channel* as the master volume. The polyfile flag is set in the files header, and record zero is available to the application.

status is any *num.var* used to return an exception status from **$VOLLINK**.

param is any *num.array* **DIM**ensioned as *array[n]* where *n* is at least 10.

If *status* is returned as a non-zero value, then an error occurred.

Status	Description
1	Illegal channel number
16	Volume vol is not defined.

As implemented in UniBasic, **$VOLLINK** can only be used following initial creation of the contiguous file to define the master volume. Attempts to define additional volumes or receive full status information is not available at the time of this writing.

EXAMPLES

```
CALL $VOLLINK,5,5,0,S,E ! Structure Volume 0

CALL 91,5,5,0,S,E ! Structure Volume 0
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

Indexed Data Files

CALL 15

SYNOPSIS

Pack and Unpack Numeric Strings.

SYNTAX

CALL 15, *source* , *dest*

DESCRIPTION

source is any *str.var* used as the source string. If dimensioned greater

than *dest*, a pack is performed. If dimensioned less, an unpack is performed.

dest is any *str.var* used to contain the destination string. Receives data following pack/unpack.

CALL 15 permits strings containing only numeric digits (0 thru 9) to be packed or unpacked into four-bit nibbles (two digits per byte). The packed string is then half the length of the original string. Generally, this call is used to reduce file size when numeric-only keys are used.

In addition to digits, the characters "+,-." and space are valid for packing. The following table depicts each digit's packed representation:

CHAR	PACKED	CHAR	PACKED
+	0001	,	0010
-	0011	.	0100
Space	0101	0	0110
1	0111	2	1000
3	1001	4	1010
5	1011	6	1100
7	1101	8	1110
9	1111		

EXAMPLES

```
CALL 15,A$,B$

CALL 15,B$,A$
```

ERRORS

String Expression must be used here

Error detected in/by user CALL routine

See also

CALL 20/21, CALL 45/46

CALL 18/19

SYNOPSIS

Pack and Unpack Radix 50.

SYNTAX

CALL 18, *ascii* , *packed* ! Pack a string

CALL 19, *packed* , *ascii* , {*flag*} ! Unpack a string

DESCRIPTION

ascii is any *str.var* containing ASCII characters to be packed into Radix 50. Packing permits 3 characters to be stored into 2-byte positions reducing a strings length by 33%.

packed is any *str.var* **DIM**ensioned at least 66% of the size of the *ascii* and contains packed Radix 50 characters.

Radix 50 packing allows 3 bytes of string information to be packed into 2 physical bytes of storage using the formula:

$(C_1*40+C_2)*40+C_3$ where C is a character's pack value..

A 40-character (50_8) subset is utilized for this type of packing. Lower case letters will be converted to upper case automatically. The resulting *packed* string is 66% the length of the *ascii* string.

flag is any optional *num.var* used to specify whether to remove trailing spaces following an unpack. If the flag is omitted or zero, then *ascii* will be space-filled past the end of data and up to its dimensioned length.

The following table depicts each valid radix 50 character and its packed representation:

CHAR/PACK	CHAR/PACK	CHAR/PACK	CHAR/PACK
0 01	A 11	K 21	U 31
1 02	B 12	L 22	V 32
2 03	C 13	M 23	W 33
3 04	D 14	N 24	X 34
4 05	E 15	O 25	Y 35
5 06	F 16	P 26	Z 36
6 07	G 17	Q 27	, 37
7 08	H 18	R 28	- 38
8 09	I 19	S 29	. 39
9 10	J 20	T 30	SP 00

EXAMPLES

```
DIM A$[100],B$[66]

CALL 18,A$,B$ !PACK

CALL 19,B$,A$,1 !UNPACK TRIM TRAILING SPACES
```

ERRORS

String Expression must be used here

Not enough parameters passed to CALL

See also

CALL 48/49

CALL 20/21

SYNOPSIS

Pack/Unpack Numeric Strings.

SYNTAX

CALL 20, *source, dest* ! Pack

CALL 21, *source, dest* ! Unpack

DESCRIPTION

When using **CALL 20**, *source* is any *str.var* containing only numeric digits to be packed into BCD data, 2 digits per byte. *dest* is any *str.var* **DIM**ensioned at least 50% of the **DIM** of source to receive the packed data.

When using **CALL 21**, *source* is any *str.var* containing a previously **CALL 20** packed string variable to be converted back to ASCII digits. The resulting un-packed data is placed into the string variable *dest*. The **DIM**ension of *dest* must be at least twice the **DIM** of *source*.

The following table depicts each digits packed representation:

DIGIT	PACKED	DIGIT	PACKED
0	0001	5	0110
1	0010	6	0111
2	0011	7	1000
3	0100	8	1001
4	0101	9	1010

EXAMPLES

```
A$="0123456787778877878"
CALL 20,A$,B$  ! PACK
CALL 21,B$,A$  ! UNPACK
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

String Expression must be used here

See also

CALL 15, CALL 45/46

CALL 22/23

SYNOPSIS

Check for Numeric /Arithmetic Field.

SYNTAX

CALL 22, *string*

CALL 23, *string*

DESCRIPTION

string is any *str.expr* which is to be checked for numeric or arithmetic data.

CALL 22 scans the selected string to ensure that it contains no non-numeric bytes. Acceptable characters are the digits 0 thru 9 only, however, a null string is accepted. An error is generated if any non-numeric character is encountered, or if the parameter passed is not a string variable.

CALL 23 scans the selected string to ensure that it contains a valid arithmetic field. Characters accepted are the digits 0 thru 9, a maximum of one period (decimal point), and a prefixed sign (+ or -). At least one digit must be present, except in the case of a null string, which will be accepted. An error will be generated if any invalid character is detected or the parameter passed is not a string variable.

EXAMPLES

```
CALL 22,"12345"

CALL 23,"+1234.55"
```

ERRORS

Error detected in/by user CALL routine

String Expression must be used here

See also

CALL $STRING, CALL 29, CALL 30, CALL 43, CALL 44, CALL 56, CALL 57, CALL 60

CALL 24

SYNOPSIS

Verify Date Inputs.

SYNTAX

CALL 24, *date* {,*result* {, *result flag* {,*mode*}}}

DESCRIPTION

date is any *str.var* containing a date to be verified. Many different date formats are supported. Valid *date* formats are:

1) xxx dd, {19}yy	2) dd xxx {19}yy	3) mm/dd/yy
4) mm.dd.yy	5) mm.dd.yyyy	6) mm/dd/yyyy

xxx is a month name of three characters or more, such as "JAN", "APR", "AUGUST", etc. *dd* is the day of the month, and *yy* or *yyyy* is the year. Any separator may be used between the fields with form 3 and 4 above as long as the same character is used, i.e. 12.22.88, or 12-22-88.

If the environment variable **EUROPEAN** is set, dates in forms 3 and 4 are assumed to be day month year.

If the *date* is valid, *date* is rearranged to the form 'yymmdd' or 'yyyymmdd', as determined by the optional *mode* parameter, without separators. If invalid, an error is generated.

The optional *mode* is any *num.expr* which, after evaluation is truncated to an integer and determines the number of digits returned to represent the year. A two digit year, i.e. "yymmdd", is returned if *mode* is unspecified or zero. A four digit year, i.e. "yyyymmdd", is returned if *mode* is non-zero.

The optional *result* is any *str.var* which, if included, receives the rearranged date leaving *date* unaffected.

The optional *result flag* is any *num.var* which, if included, receives the status of the verify operation; 0 for valid date, and 1 for an invalid date.

CALL 24 is also used to convert dates for input to **CALL 25**.

EXAMPLES

```
LET A$="JAN 23, 1990"

CALL 24,A$,B$,F \ IF F THEN STOP
```

ERRORS

Error detected in/by user CALL routine

String Expression must be used here

See also

CALL 25, CALL 27, CALL 28, CALL $TIME

CALL 25

SYNOPSIS

Convert to Julian Date.

SYNTAX

CALL 25, {*flag*,} *date* {, *result date* {, *status* }}

DESCRIPTION

flag is any optional *num.expr* used to specify the type of date format for input and output. If *flag* is not specified, zero is assumed. Flags are:

flag	Input Date	Output Date	Comment
0	yymmdd	yyddd	year and day of year; e.g. 98365
1	yymmdd	ddddd	days since January 1, 1968
2	yymmdd	yyyyddd	4 digit year and day of year; e.g. 1998365
4	yyyymmdd	yyddd	2 digit year and day of year; e.g. 98365
5	yyyymmdd	ddddd	days since January 1, 1968
6	yyyymmdd	yyyyddd	4 digit year and day of year; e.g. 1998365

date is any *str.var* in post **CALL 24** form, i.e. YYMMDD. If *date* is valid, it is rearranged to the selected Julian date form. If invalid, an error is generated.

The optional *result date* is any *str.var* which, if included, receives the converted date leaving *date* unchanged.

The optional *status* is any *num.var* which, if included, receives the error status; 0 for valid *date*, and 1 for invalid *date*.

CALL 27 is used to convert Julian dates back to printable dates.

EXAMPLES

```
CALL 25,A$,B$,S \ IF S STOP
```

ERRORS

Error detected in/by user CALL routine

String Expression must be used here.

See also

CALL 24, CALL 27, CALL 28

CALL 27

SYNOPSIS

Convert from Julian to Printable Date.

SYNTAX

CALL 27, {*flag*,} *jul date* {, *result date* {, *status* }}

DESCRIPTION

flag is any optional *num.expr* used to specify the type of date being passed to and returned from **CALL 27**. If *flag* is not specified, zero is assumed. Flags are:

flag	**Input Date**	**Output Date**	**Comment**
0	yyddd	mm/dd/yy	year and day of year; e.g. 98365
1	ddddd	mm/dd/yy	days since January 1, 1968
2	yyyyddd	mm/dd/yy	4 digit year and day of year; e.g. 1998365
4	yyddd	mm/dd/yyyy	2 digit year and day of year; e.g. 98365
5	ddddd	mm/dd/yyyy	days since January 1, 1968
6	yyyyddd	mm/dd/yyyy	4 digit year and day of year; e.g. 1998365

jul date is any *str.var* in Julian date form generated by **CALL 25**. If the date is valid, *jul date* is rearranged to "MM/DD/YY" or "MM/DD/YYYY" form. If invalid, an error is generated. The environment variable **DATESEP** is used as the separation character, and **EUROPEAN** specifies date conversion to the format: "DD/MM/YY" or "DD/MM/YYYY".

result date is any optional *str.var* which, if included, receives the converted *jul date* and *jul date* is unchanged.

The optional *status* is any *num.var* which if included, receives the error status; 0 for valid *jul date*, and 1 for invalid *jul date*.

EXAMPLES

```
CALL 27,A$,B$,S \ IF S STOP
```

ERRORS

Error detected in/by user CALL routine

String Expression must be used here.

See also

CALL 24, CALL 25, CALL 28

CALL 28

SYNOPSIS

Convert to Printable Date.

SYNTAX

CALL 28, *date* {,*result date* {, *status* {,*mode*}}}

DESCRIPTION

date is any *str.var* containing a date in post **CALL 24** form, i.e. "YYMMDD" or "YYYYMMDD". Date is converted to standard "MM/DD/YY" or "MM/DD/YYYY" output format as determined by the optional *mode* parameter.

An error is generated if the parameter is not a string variable dimensioned at least 8 bytes, or if *date* is invalid. If the environment variable **EUROPEAN** is set, the *date* is converted to the form "DD/MM/YY" or "DD/MM/YYYY" as determined by the optional mode parameter. The environment variable **DATESEP** is used as the separation character.

The optional *mode* is any *num.expr* which, after evaluation is truncated to an integer and determines the input and output date formats for the **CALL**. If *mode* is not specified, zero is assumed. Modes are:

mode	**Input Date**	**Output Date**
0	yymmdd	mm/dd/yy
1	yyyymmdd	mm/dd/yy
4	yymmdd	mm/dd/yyyy
5	yyyymmdd	mm/dd/yyyy

The optional *result date* is any *str.var* which, if included, receives the rearranged date leaving *date* unchanged.

The optional *status* is any *num.var* which, if included, receives the error status; 0 for valid *date*, and 1 for invalid *date*.

EXAMPLES

```
CALL 28, A$
```

ERRORS

Error detected in/by user CALL routine

See also

CALL 24, CALL 25, CALL 27

CALL 29

SYNOPSIS

Edit numeric Field.

SYNTAX

CALL 29, *string*, *mask*, *result*

DESCRIPTION

string is any *str.var* which contains data to be edited.

mask is any *str.expr* used as a field mask to edit the data supplied in *string*. This may consist of any combination of the following characters:

A	Fixed length alphabetic (A-Z). The current source byte must be alphabetic.
N	Fixed length numeric (0-9). The current source byte must be numeric.
X	Variable length alpha-numeric (any character). The current source byte may be any character.
V	Variable length alphabetic. The current source byte can be alphabetic. If not, comparison continues with the next mask byte.
Z	Variable length numeric. The current source byte can be numeric. If not, comparison continues with the next mask byte.
/	Field separator. The current source byte may be any one of "/", ".", or "-".
.	Decimal point. The current source byte must be a ".", unless followed by "V" or "Z" in the mask.
-	Minus sign. The current source byte must be "-", unless this is the first byte of the mask. If so, comparison continues with the next mask byte.

Any other character that appears in the mask must appear in the source string in the corresponding position.

result is any *str.var* defined to receive the edited *string*.

CALL 29 verifies that a given *string* conforms to the specifications of another string, termed a *mask*. The edit is performed by comparing the *string* with the *mask* , byte by byte.

The following table illustrates some typical editing examples:

MASK	EFFECT
`-ZZZ.ZZ`	Allows a number between -999.99 and 999.99 with a maximum of 2 fractional digits.
`ANA NAN`	This mask is used for the Canadian Postal Code. The source string length must be 7 bytes, with a space in the fourth position. Each letter and digit must be in its fixed place.
`NZZZ.NZ`	Allows a minimum of 1 digit before and after the decimal, and a maximum of 4 before and 2

after. The decimal point must exist. Note that "0.0" is allowed.

`VVVNZZ`	Source "A45" results in edit of "A045".

In a sequence of fixed and variable length numeric edit characters ("N" and "Z"), the fixed length character must appear before the variable length character.

In numeric fields, an edit results in left zero-filling of the field.

An error will occur if:

- Any parameter is not a string variable.
- Source does not conform to mask.
- Destination string dimension is too small.
- Same string used for source and destination.

EXAMPLES

```
CALL 29,S$,M$,D$
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

String Expression must be used here

See also

EDIT, USING

CALL 40

SYNOPSIS

Initialize access to User Message Files.

SYNTAX

CALL 40, *channel*, *filename*

DESCRIPTION

channel is any *num.expr* which, after evaluation, is truncated to an integer and used to specify an unused channel number for temporary use to **OPEN** the message file. Once opened, the channel is then cleared and free for use by the program. A channel which is not currently in use should be selected or it will be closed automatically.

filename is any *str.expr* specifying a *filename* or *pathname* to a user error message file to which you have read-permission. User error-message files conform to the structure defined in Error Message File.

CALL 40 works in conjunction with the **ERM** function to read user messages from a disk file. The CALL selects the file to be used for all

further **ERM** functions. By issuing secondary **CALL 40** statements, one may use different message files for different application packages or even within the same program.

The error message file is a text file with each message beginning with the desired message number (must have a message 0 defined as "No such Message Number), a colon, and the text (up to 80-characters).

EXAMPLES

```
CALL 40,3,"0/AR.MESSAGES"

PRINT ERM(12) ! Print Message 12
```

ERRORS

Not enough parameters passed to CALL

String Expression not allowed here

String Expression must be used here

See also

Error Message File, **ERM**

CALL 43

SYNOPSIS

Convert to Upper/Lower Case

SYNTAX

CALL 43, *mode*, *string* {, *position* }}

DESCRIPTION

mode is any *num.expr* which, after evaluation, is truncated to an integer and used to select one of the following operations:

Mode	Operation Performed
1 =	Convert all letters to upper case.
2 =	Convert first letter only to upper case.
3 =	Convert first letter of each word to upper case.
4 =	Convert all letters to lower case.
5 =	Convert first letter and any single 'I' to upper case.
6 =	Convert all letters to lower case, and any single 'I' to upper case.

string is any *str.var* to be converted. The conversion takes place within the specified *string*.

The optional *position* is any *num.expr* which, after evaluation, is truncated to an integer and used to specify the starting character position in *string*.

EXAMPLES

```
CALL 43,1,A$,5
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

String Expression must be used here

See also

CALL 60, CALL $STRING

CALL 44

SYNOPSIS

Miscellaneous String Functions & ECHO.

SYNTAX

CALL 44, *mode*, { *target*, *string*, *position*, {*step*} }

DESCRIPTION

mode is any **num.expr** which, after evaluation, is truncated to an integer to select one of the following modes of operation:

Mode	Operation Performed
0	Compare *target* to *string*.
1	Search *string* for first occurrence of *target*.
2	Search *string* for first non-occurrence of *target*.
3	Swap *target*. Reverses position of all bytes.
4	Disable terminal echo.
5	Enable terminal echo.

target is any *str.var* used in *modes* 0 thru 2 as the 'search for' string. In *mode* 3, *target* is the string to be swapped.

string is any *str.var* in *modes* 0 thru 2 to be searched for the *target* string.

position is any *num.var* which, after evaluation, is truncated to an integer to select the starting character position for search (*mode* 1 and 2). Returns position of *target* string, or zero if not found. *Mode* 0 causes *position* to return comparison status as follows:

-2 = *string* logically less than *target*

-1 = *string* shorter than *target*

0 = *target* and *string* exactly equal

1 = *target* shorter than *string*

2 = *target* logically less than *string*

The optional *step* is any *num.expr* which, after evaluation, is truncated to an integer and used as a counter for search (*modes* 1 and 2). Causes comparison to be performed every *step* bytes in search string. Default = 1.

CALL 44 is be used for string searching, comparison, and swap. A *step* counter is optional and provides for searching a string at 2, 3, or whatever byte intervals. Search or comparison is full eight-bit and terminates on null byte.

In addition, **CALL 44** may be used to enable or disable terminal echoing of input characters.

EXAMPLES

```
CALL 44, A$, B$, P

CALL 44,4 !Disable Echo
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

CALL $ECHO, IO Mnemonics, **SYSTEM, CALL $ECHO, CALL 56, SEARCH**

CALL 45/46

SYNOPSIS

Pack/Unpack Numeric Strings.

SYNTAX

CALL 45, {*flag*,} *source*, *dest* {,*status*}

CALL 46, *source*, *dest*

DESCRIPTION

flag is any *num.expr* which, after evaluation is truncated to an integer and used to select an optional mode for **CALL 45**. A zero (or omitted) invokes a packing operation. A value of 1 provides for unpacking operations within **CALL 45**.

source is any *str.var* which is to be operated upon (pack or unpack).

dest is any *str.var* to contain the resulting operation (pack or unpack).

The optional *status* is any *num.var* used to return an exception status from CALL 45. Zero is returned for a successful operation, 1 indicates an error. If status is not supplied, a normal BASIC error is generated if a conversion error occurs.

CALL 45 and **CALL 46** permit strings containing only numeric digits (0 thru 9) to be packed or unpacked into four-bit nibbles (two digits per byte). The packed string is then half the length of the original string. Generally, this call is used to reduce file size when numeric-only keys are used.

In addition to digits, the characters ",-./" and space are valid for packing. The following table depicts each digit's packed representation:

CHAR	PACKED	CHAR	PACKED
Space	0001	2	1000
,	0010	3	1001
-	0011	4	1010
.	0100	5	1011
/	0101	6	1100
0	0110	7	1101
1	0111	8	1110
		9	1111

EXAMPLES

```
CALL 45,0,A$,B$,F \ IF F STOP !PACK

CALL 45,1,B$,A$,F \ IF F STOP !UNPACK

CALL 46,B$,A$ !UNPACK
```

ERRORS

Error detected in/by user CALL routine

See also

CALL 20/21, CALL 15

CALL 47

SYNOPSIS

Perform Miscellaneous Functions.

SYNTAX

CALL 47, *mode*, *return*

DESCRIPTION

mode is any *num.expr* which, after evaluation, is truncated to an integer to select the operation for **CALL 47**.

return is any *num.var* used to return information, or any *num.expr* used to pass information to **CALL 47**.

Mode	Operation Performed
0	Pop top of **GOSUB** stack, place return statement number in return. Returns zero if no **GOSUB** pending. Same as **MSC(3)** and **SPC(14)** functions.
1	Push statement number in *return* onto **GOSUB** stack. Similar to **GOSUB** statement, without branching.
2	not supported. results in error.
3	Return current **TERM** type in return. Same as **MSC(32)** and **SPC(13)** functions.
4	Disable terminal echo.
5	Enable terminal echo.

EXAMPLES

```
CALL 47,1,2300 !Push onto GOSUB stack
```

ERRORS

Error detected in/by user CALL routine

GOSUBS Nested too deep

No such statement number

See also

IO Mnemonics, **SYSTEM, CALL $ECHO, SPC, MSC, GOSUB, RETURN**

CALL 48/49

SYNOPSIS

Pack/Unpack Radix 50 Characters.

SYNTAX

CALL 48, *ascii* , *packed*

CALL 49, *packed* , *ascii*

DESCRIPTION

The *ascii* is any *str.var* containing ASCII characters to be packed into Radix 50. Packing permits 3 characters to be stored into 2-byte positions reducing a strings length by 33%.

The *packed* is any *str.var* **DIM**ensioned at least 66% of the size of the

ascii and contains packed Radix 50 characters.

Radix 50 packing allows 3 bytes of string information to be packed into 2 physical bytes of storage using the formula:

$(C_1*40+C_2)*40+C_3$ where C is a character's pack value.

A 40-character (50_8) subset is utilized for this type of packing. Lower case letters will be converted to upper case automatically. The resulting packed string is 66% the length of the *ascii* string.

The following table depicts each valid radix 50 character and it's packed representation:

CHAR/PACK	CHAR/PACK	CHAR/PACK	CHAR/PACK
, 01	7 11	H 21	R 31
- 02	8 12	I 22	S 32
. 03	9 13	J 23	T 33
0 04	A 14	K 24	U 34
1 05	B 15	L 25	V 35
2 06	C 16	M 26	W 36
3 07	D 17	N 27	X 37
4 08	E 18	O 28	Y 38
5 09	F 19	P 29	Z 39
6 10	G 20	Q 30	SPACE 00

EXAMPLES

```
CALL 48,A$,B$ ! PACK

CALL 49,A$,B$ ! UNPACK
```

ERRORS

Not enough parameters passed to CALL

String Expression must be used here

See also

CALL 18/19

CALL 53

SYNOPSIS

ASCII/EBCDIC Conversion.

SYNTAX

CALL 53, *string*, {*flag*}

DESCRIPTION

string is any *str.var* to be converted to/from ASCII and EBCDIC.

The optional conversion *flag* is any *num.expr* which, after evaluation, is truncated to an integer. If omitted or zero, *string* is converted from EBCDIC to ASCII. If one, then *string* is converted from ASCII to EBCDIC.

The entire ASCII character set is convertible back and forth from ASCII to EBCDIC. There are many EBCDIC characters, however, which have no ASCII counterpart. These characters will be converted to nulls if encountered.

EXAMPLES

```
CALL 53,A$ ! CONVERT TO ASCII

CALL 53,A$,1 ! CONVERT TO EBCDIC
```

ERRORS

Data of wrong type (numeric/string)

See also

CALL $ETOA, CALL $ATOE

CALL 56

SYNOPSIS

External Subroutine to Provide String Searching

SYNTAX

CALL 56, {*flag*,} *string* {,*start*}, *target, position* {,*occur* {,*searchstep* {,*targetstep* }}}

DESCRIPTION

The optional ignore-null *flag* is any *num.expr* which, after evaluation is truncated to an integer. If omitted or zero, the search terminates at a null in the search string. 1 = search past null to dimensioned length.

string is any *str.var* to be searched.

The optional *start* is any *num.expr* which, after evaluation is truncated to an integer and used to specify the starting character position in *string*. *ABS*(*start*) indicates the position. If *start* is negative, a backwards search is performed.

target is any *str.expr* specifying the substring to locate in *string*.

position is any *num.var* used to returns the character position of *target* within *string*. If not found, *position* returns -1.

The optional *occur* is any *num.expr* which, after evaluation is

truncated to an integer and used to specify a search count. If *occur* is positive, *string* is searched for the *occur* occurrence of *target* . If negative, *string* is searched for the **ABS**(*occur*) non-occurrence of *target* .

The optional *searchstep* and *targetstep* are any *num.expr* which, after evaluation are truncated to integers and used to specify a 'step size'. If either option is used, **CALL 56** steps through the appropriate string by the specified number of characters. For example, if *searchstep* is set to 5, then **CALL 56** looks for a match at every fifth character of the *string*, starting at the offset specified in *start*. If *start* = 1, then the offsets used for a match would be *string*[1], [6], [11], etc. Note that in order to use *searchstep*, *occur* must be specified. Similarly, in order to use *targetstep*, both *occur* and *searchstep* must be specified.

EXAMPLES

```
CALL 56,1,A$,I+1,T$,P,3 !3rd occurrence at I+1
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

User CALL parameters out of order

See also

SEARCH, CALL 44, CALL 43

CALL 59

SYNOPSIS

Numeric BIT Manipulation.

SYNTAX

CALL 59, *mode, arg1, arg2* (,*flag*}

DESCRIPTION

mode is any *num.expr* which, after evaluation, is truncated to an integer to specify one of the following operations:

Mode	Operation Selected
0	Reset (zero) bit number *arg1* in variable *arg2* . *flag* returns bit *arg1* before reset.
1	Set bit number *arg1* in variable *arg2* to one. *flag* returns bit *arg1* before set.
2	Test bit number *arg1* in variable *arg2* . *flag* returns zero if the bit is zero or $2^{15-arg1}$ if the bit is one.
3	**AND** variable *arg1* to variable *arg2* and store result in *arg2* . A logical **AND** produces a one in each bit

position set in both *arg1* and *arg2* .

4 **OR** variable *arg1* to variable *arg2* and store result in *arg2* . A logical **OR** produces a one in each bit position set in either *arg1* or *arg2* or both.

5 **XOR** variable *arg1* to variable *arg2* and store result in *arg2* . A logical **XOR** (exclusive OR) produces a one in each bit position set in either *arg1* or *arg2* but not in both.

6 Complement (NOT) variable *arg1* and store result in variable *arg2* . Each one bit is set to zero and vice-versa.

arg1 is any *num.var* used to select one binary argument to the CALL.

arg2 is any *num.var* used to select a second binary argument to the CALL.

The optional *flag* is any *num.var* used to return information from the CALL.

CALL 59 provides bit manipulation on integer variables in the range 0 thru 65535 (177777_8). One-word arithmetic and logical operations are also provided.

The following table illustrates the effect of the logical operations:

X	Y	X AND Y	X OR Y	X XOR Y	NOT Y
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

EXAMPLES

```
CALL 59,M,A,B,F
```

ERRORS

Not enough parameters passed to CALL

Data of wrong type (numeric/string)

Illegal Parameter or Syntax for Command

Function Argument or Statement Mode out of range

See also

CALL $LOGIC

CALL 60

SYNOPSIS

Miscellaneous String Functions.

SYNTAX

CALL 60, {*mode*,} *string ...*

DESCRIPTION

mode is any *num.var* which, after evaluation, specifies an optional mode of operation for the CALL. If omitted, zero is assumed.

Mode	Operation Performed
0	Perform upper case conversion on all lower case letters.
1	Null the entire string (as in **CALL 57**).
2	Set the high-order bit of each data byte to one, excluding nulls. This is generally used when data is read from other operating systems. UniBasic internally stores all ASCII characters with their top bit zero to force them in the range 000_8 to 177_8.
3	Toggle all high-order bits of each character except for zero bytes and nulls (000_8 and 200_8). This mode is used when data is brought to UniBasic from IRIS, BITS or other high-bit string operating systems.

string is any *str.var* to be operated upon.

Multiple strings are permitted by **CALL 60**. Also, the occurrence of a numeric value resets the *mode* of operation for the following strings until another numeric value is specified.

EXAMPLES

```
CALL 60,1,A$,B$,C$ !NULL THE STRINGS

CALL 60,A$,B$ !UPPER CASE

CALL 60,1,A$,2,B$,3,C$
```

ERRORS

Error detected in/by user CALL routine

User CALL parameters out of order

See also

CALL $STRING, CALL 29, CALL 30, CALL 43, CALL 44, CALL 56

CALL 65

SYNOPSIS

Sort Keys in a String.

SYNTAX

CALL 65, *status, number, length, sort, work*

DESCRIPTION

status is any *num.var* to receive a return status from the sort operation:

Status	Description
0	Successful sort operation.
1	Parameter Error.
2	*number* or *length* was passed as zero.
3	*sort* string is too small; less than *number* * *length*.
4	*work* string is too small; less than *length* + *8*.

number is any *num.var* which, after evaluation, is truncated to an integer to specify the number of strings to be sorted.

length is any *num.var* which, after evaluation, is truncated to an integer to specify the length of each string.

sort is any *str.var* containing keys to be sorted.

work is any temporary work string **DIM**ensioned to a minimum of *length* + *8*.

The *sort* string may contain any number of fixed-length binary fields to be sorted. Sorting is based upon the supplied *length* of each item, up to *number* of items.

The resulting sorted string is returned in the *str.var sort*.

EXAMPLES

```
CALL 65,E,100,10,A$,W$
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

none

CALL 72/73

SYNOPSIS

Gather / Scatter Variables.

SYNTAX

CALL (**72** | **73**) , *string, var.list*

DESCRIPTION

string is any *str.var* from which to gather or scatter data. Its size must be large enough to load from or store to all of the variables in the *var.list*.

var.list is any list of *str.vars*, *num.vars*, *array.vars*, or *mat.vars* to be gathered from or scattered to. Only single elements of an *array.var* or *mat.var* may be specified. An entire *matrix* or *array* is not copied by supplying its variable name.

CALL 72 is used to gather a group of variables in the *var.list* and copy their contents (binary) into *string*.

CALL 73 scatters data from *string* into each variable in the *var.list* using a binary copy.

CALL 72/73 may be used with mixed class data (BCD/Base 10000), but numeric data in string must be of Base 10000. When numeric variables are gathered, BCD variables are automatically converted to their Base 10000 equivalent. During scatter, variables are then stored into the type of the variable in the *var.list*.

CALL 72/73 are typically used by applications designed prior to **COM** and **CHAIN READ/CHAIN WRITE**. Data would be gathered and written to a temporary file. Following a **CHAIN**, the data would be read and scattered into the appropriate variables in the new program.

EXAMPLES

```
CALL 72,2,A$[31,34],P !CONVERT 2%
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

See also

none

CALL 126

SYNOPSIS

Convert Decimal to Octal.

SYNTAX

CALL 126, *value*, (*string* | *number*)

DESCRIPTION

value is any *num.expr* which, after evaluation, is truncated to an integer to specify a decimal value to be converted. *value* must be in the range 0 to 2^{31}-1.

string is any *str.var* to receive the octal equivalent in ASCII form,

right-justified. The string variable should be **DIM**ensioned at least 12 to hold the result. The first position receives the sign, which are: space for positive, and "-" for negative. The remaining 11 positions receive the right-justified octal value with leading spaces.

If *number* is specified, it must be any *num.var* to receive the numeric value of octal equivalent.

EXAMPLES

```
CALL 126,a,a$

CALL 126,B,B
```

ERRORS

Not enough parameters passed to CALL

Data of wrong type (numeric/string)

See also

none

CALL 127

SYNOPSIS

Convert Directory Information.

SYNTAX

CALL 127, *directory, array, filename* { , *mode* , *information* , *encryption* }

DESCRIPTION

directory is any *str.var*, **DIM**ensioned at least 14 bytes, containing a BITS directory entry (used only for *mode* 0 BITS Conversion Package).

array is any *num.array* variable **DIM**ensioned for at least 25 entries at 2% or larger, used to return unpacked information about a file. Information returned is accessed by the element:

- [0] Account group (0-255).
- [1] Account user (0-255).
- [2] Attribute word as a numeric value Mode 0 only.
- [3] File type (0-9), represents “O$BACTSI”.
- [4] First disk address.
- [5] Record length in bytes. For non-UniBasic files, A[3]=0, returns 512 for text files and 65534 for non-text file.
- [6] File size in blocks (represents both halves of an indexed file).
- [7] Creation date in the form MMDDYY.

[8] Last access date in the form MMDDYY.

[9] Relative sector offset; Mode 0 only.

[10] Size of record map in sectors (INDX files Mode 0 only).

[11] Number of indices (Index files only).

[12] System time at last access in hours.

[13] Secondary attribute word as a numeric value; Mode 0 only.

[14] Logical unit number, as currently installed; Mode 0 only.

[15] DIRECTORY sector number; Mode 0 only.

[16] Word displacement into DIRECTORY sector; Mode 0 only.

[17] Unix Protection bits; Mode 1 only.

[18] Number of items per record; Mode 1 only.

[19] Revision of UniBasic at time file was created; Mode 1 only.

[20] First Real Data Record as built; Mode 1 only.

[21] Byte offset to Record 0; size of header; Mode 1 only.

[22] Returns the files creation time in hours-since-BASEDATE.

Record length in element A[5] is 512 bytes for a non-UniBasic text file and 65534 for a non-UniBasic file of type A[3]=0. The first block of the file is examined and is only considered text if all bytes are <0x80.

filename is any *str.var* specifying a file in *mode* 1, or to return an unpacked name for *mode* 0. The *filename* should be DIMensioned at least 31 characters (64 characters recommended). Returned in filename is the actual 14-character name. Supplemental attributes are returned in bytes 15-29; <PRWdsEOxFQUgabKY>. Lower-case letters refer to BITS attributes which are only returned when *mode* 0 is used on a BITS directory unpack.

mode is any *num.expr* which, after evaluation is truncated to an integer and used to specify the operational mode for the CALL. If omitted or 0, then a BITS DIRECTORY entry in *directory* is unpacked. *Mode* 1 is used to locate and return information about the file contained in *filename*. *Mode* 1 is used exclusively for **QUERY** and **SCAN** files. *Mode* 0 is used by the BITS Conversion Package.

information is any optional *array.var* used to return information about an Indexed or Formatted Item File. It must be **DIM**ensioned as *information[128,1]*.

If the file is Indexed, *information* returns the following information:

information[0,0] Record length in bytes for file.

information[0,1] Current actual active record count.

information[X,0] Key length for Directory X.

information[X,1] Active Keys in Directory X **.

** Most systems do not maintain the current key counts in Indexed files to increase performance of insertion and deletion operations, so this value is returned as zero. The *information[]* array is valid from 1 to the number of Indices returned in *array[11]*.

If the file is a Formatted Item file, *information* returns the following information:

information[X,0] Item Type

information[X,1] Item length in bytes.

(Release 9.1) *encryption* is any optional *array.var* used to return information about the Encrypted File. It must be **DIM**ensioned as *encryption[128,2]*.

If used, *encryption* returns the following information, where "X" is the segment number:

encryption[X,0] Segment fill type.

encryption[X,1] Segment item number.

encryption[X,2] Segment length in bytes.

The end of the array is marked by a segment definition with a length of zero. The seqment fill type has the following meaning.

0	Seqment cannot be read without a valid encryption key.
1	Segment is zeroed if read without a key ("Z").
2	Segment is space filled if read without a key ("S").
2	Segment is asterisk filled if read without a key ("*").

EXAMPLES

```
A$="DATAFILE"

CALL 127,D$,A,A$,1,T
```

ERRORS

Error detected in/by user CALL routine

Not enough parameters passed to CALL

File does not exist

See also

CALL $FINDF, CALL $RDFHD

Supplied Utilities

Your installation media includes a set of system utilities to assist the developer in application debugging, file maintenance, and system status.

The supplied utilities are documented as one of the following types:

System **BASIC** utilities, documented using upper-case names, are written in BASIC and may be viewed in source form. Most are supplied to simulate the familiar IRIS and BITS system processors. Utilities written in BASIC are only available during a UniBasic session.

Compiled 'C' **OBJECT** programs, documented using lower-case names, are written in 'C' and compiled to native binary on each platform supported by Dynamic Concepts. These utilities, as with UniBasic itself, are not generally portable between machines.

Utilities are launched in either *command mode* or directly within the Unix shell. Utilities written in BASIC are restricted to *command mode* and are identified in the synopsis and examples by the prompt # **SCOPEPROMPT**. C-Language Object utilities are identified in the synopsis and examples by the Unix shell prompt $.

When operating within the environment **BASICMODE=BITS**, certain utilities must be prefixed with a **!** to prevent misinterpretation by the BITS-style combined *command mode* and *program mode* command line interpreter.

For example:

#KILL filename	!Launch file delete utility from IRIS mode
*/KILL filename	!Launch file delete utility from BITS mode
*KILL "filename"	!BITS KILL statement in immediate mode
*!kill process	/!Force Unix process KILL command in BITS mode
#/!kill process	!Force Unix process KILL command in IRIS mode

BATCH

SYNOPSIS

Logon and execute commands on a different port.

SYNTAX

#**BATCH** { **/H** | {*port* {*command* | ^*commandfile* }}}

DESCRIPTION

The **BATCH** command 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 UniBasic *port number*. If *port* is not supplied on the command line, prompt mode is selected (see below). The *port* must be a valid UniBasic *port number*. If an interactive UniBasic 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 optional UniBasic 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. Starting a **DIR** or **LIBR** command in background is an example when this mode is used.

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 **[ESC]** terminates entry of *commands* and requests a new port number for another prompt-mode session. Pressing **[ESC]** a second time terminates **BATCH**.

When running in the environment **BASICMODE=IRIS**, the initial attachment of the port is in command mode (#).

EXAMPLES

```
#BATCH 1 ^COMMANDFILE

#BATCH 87 LIBR [OUTPUT] ^

#BATCH
```

ERRORS

Illegal Port Number

Cannot attach Port

See also

PORT, CALL 98, Port Numbering and Phantom Ports

BUILDXF

SYNOPSIS

Build a new IRIS Indexed File.

SYNTAX

#BUILDXF

DESCRIPTION

BUILDXF is the standard IRIS Indexed File creation utility. It is used

whenever a standard Indexed Data File (not a Polyfile) is to be created. The first real data record is always set to one (1) when building a file with this utility.

When creating a new file, simply enter one (1) for the number of data and indexed records since Indexed files typically expand dynamically. The Environment Variable: **PREALLOCATE** may be used to limit the number of records allowed during expansion, automatic pre-allocation and other special Indexed File features, controls and restrictions.

The *filename* must be given in the form *filename!* if an existing file is being replaced.

BUILDXF supports up to 62 directories, 122-byte keys and an unlimited number of records.

Prompt	**Information to be Entered**
Desired Filename	The desired pack, logical unit, or directory name and filename. An "!" must be appended to replace an existing file.
Number of Data Records	Total number of data records to be contained in the file when created. You must specify 1 record to create the file for, or the prompt for the record length is omitted. The Environment Variable **PREALLOCATE** may be used to limit the number of records allowed during expansion, automatic pre-allocation and other special Indexed File features, controls and restrictions.
Data Record Length (# words)	The maximum size, in words, of the data records to be used.
Number of Indexed Records	This value will be added to the Number of Data Records entry above. For normal use, press return.
Number of Directories	Total number of directories to be contained in the file. A maximum value of 62 may be entered.
KEY Length for each directory	The maximum length in words for each **KEY** for each directory. If more than one directory is defined, you will be prompted to enter the size of each directory's key length. A maximum value of 61 may be entered.

EXAMPLES

```
#buildxf

PROGRAM TO CREATE AN INDEXED DATA FILE
```

```
DESIRED FILENAME? datafile

NUMBER OF DATA RECORDS? 1

DATA RECORD LENGTH (#WORDS)? 512

NUMBER OF INDEXED RECORDS? [return]

NUMBER OF DIRECTORIES? 2

ENTER KEY LENGTH (#WORDS) FOR EACH DIRECTORY:

#1 ? 4

#2 ? 12

PLEASE WAIT . . .

FILE HAS 1 DATA RECORDS
```

ERRORS

File already exists, use "!" to replace

Value out of range

See also

MAKEIN, Indexed Data Files, **PREALLOCATE**

CHANGE

SYNOPSIS

Change attributes and/or filename.

SYNTAX

#**CHANGE** {*switches*} *filename* (IRIS)

*/**CHANGE** {*switches*} *filename* (BITS)

DESCRIPTION

switches represent the optional entry of either **/H** or **?**. Either will display instructions.

filename is any file in the form of *lu/filename*, or *dir/filename* only. Full Unix pathnames are not allowed.

CHANGE operates in a dialogue mode, displaying the current and requesting new information. 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 **[ESC]** to terminate the command.

When prompted for Protection, enter IRIS style 2-digit values or enclose within <> BITS, Supplemental, or Unix 3-digit permissions.

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

CHANGE utilizes the **MODIFY** statement to perform the operations.

Note When operating within the **BASICMODE=BITS** environment, **CHANGE** invokes the internal command with the same name (See **CHANGE** Command). To invoke this utility, the command must be entered in the form: **/CHANGE**

EXAMPLES

```
#CHANGE TEST

TEST IS A UNIX DATA FILE

IF NO CHANGE, PRESS RETURN - [CHANGE ? FOR HELP]

FILENAME = TEST

NEW NAME ? [RETURN]

COST = $0.00

NEW COST ?[RETURN]

PROTECTION: 60

NEW PROTECTION ? <666>
```

ERRORS

File does not exist

File is Read Protected

File is Write Protected

See also

Filenames and Pathnames, File Attributes, Protection and Permissions, Using IRIS Protections, Using Unix Permissions, BITS Attributes, Supplemental Protection Attributes

COPY

SYNOPSIS

Make an identical copy of any file.

SYNTAX

#**COPY** {*<attr>*} *destination* = *source*,{*source1*}

DESCRIPTION

<attr> are any optional protections or permissions for the new file. These may be specified in IRIS, BITS or Unix format.

destination is the name of a new file to create. It may be in the form *lu/filename*, *pack:filename*, or it may be any Unix full *pathname*. If destination begins with $, no file is created. Instead, the *destination* is

opened and data is copied a line at a time from the *source*. If the *destination* is executable, a pipe is opened to the *destination*. Otherwise, data from the *source* overwrites the *destination*.

source is the name of an existing file to which you have read-permission. If more than one *source* filename is given, data is merged into the *destination*.

If *destination* filename is to replace an existing filename, then *destination* must be given in the form of *filename!*.

COPY creates the *destination* using the supplied *attr* or, if no *attr* are supplied, using the default permissions *666*. These are affected by the current value of **umask**.

If the *source* is the name of a UniBasic Indexed Data File, and *dest* is not a named pipe, both the data portion, and the ISAM portion file is copied.

COPY utilizes the **DUPLICATE** statement to perform the operation.

EXAMPLES

```
#COPY /usr/ub/1/payrollbackup = /usr/ub/1/payroll

#COPY <644> programsave=program

#COPY $lpt=data2

#COPY $/usr/bin/pg=textfile
```

ERRORS

Filename does not exist

Illegal Pathname specified

See also

Filenames and Pathnames, File Attributes, Protections and Permissions, **DUPLICATE**, Pipes

DIR

SYNOPSIS

Produce an expanded listing of files in a directory.

SYNTAX

#**DIR** {*switches*}

DESCRIPTION

switches 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 following *switches* may be entered in any order, separated by spaces:

/H Print instructions for using **DIR**. An abbreviated list of commands and their formats is displayed.

***switches* controlling re-direction & output:**

/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.

***switches* controlling location and owner of files to display:**

path: Specify the Unix full *pathname*, or a *pathname* within the Environment Variable **LUST** from which to create the directory listing. *pathname* must be terminated by a colon.

[GRP-USR] List public files on the Unix 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.

***switches* controlling alphabetization of output:**

/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*.

***switches* restricting type & age:**

T=type Restrict listing to specific file types. These types are:

T Tree-Structured Data 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.

>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.

***switches* restricting *filenames* :**

(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
```

ERRORS

No such file or directory

See also

LIBR, MAKECMND

FORMAT

SYNOPSIS

Create a Formatted Data File.

SYNTAX

#**FORMAT** {/*H* | {*<attr>* { *[X:Y] Filename* } } (IRIS)

*/**FORMAT** {/*H* | {*<attr>* { *[X:Y] Filename* } } (BITS)

DESCRIPTION

The **FORMAT** utility creates and defines fixed record Formatted Data Files. Options may be entered directly on the command line. Required parameters not entered on the command line will be requested as input.

If a single *Filename* is specified on the command line, **FORMAT** attempts to build it. If multiple *Filenames* are entered, they are not built until the record has been defined.

The utility requests all format information required to build the file. Enter **ESC** at any time to abort the file. For each field to be defined,

the following information is requested:

/H	Display instructions for using **FORMAT**.
<attr>	Specify the file's optional attributes. Attributes may be specified as 2-digit IRIS protections, BITS attributes, Supplemental attributes or 3-digit Unix permissions enclosed with < >.
{[X:Y]}	When specified on the command line, **FORMAT** creates a Contiguous data file with *X* records of record length *Y* words.
Filename	Optional name(s) of the file or files to be created. A filename must be specified on the command line when a Contiguous data file is being created.

When the command line does not specify creation of a Contiguous file, **FORMAT** enters a conversational mode. Information not supplied on the command line is requested, including:

Request	**Information to be Entered**
Filename	Enter the *filename* to create. To replace an existing file, append "!" to the end of the *filename*.
Attributes	Enter the desired attributes for the file being created. Attributes may be specified as 2-digit IRIS protections, BITS attributes, Supplemental attributes or 3-digit Unix permissions enclosed with < >. See also File Attributes and Permissions.
ITEM#	Enter the various types of fields and their lengths to be defined within the file. Valid types are: ***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 4. See also, Numeric Variable Precision. For example, N2 will create a 4-byte numeric field. ***Bn*** Binary strings or matrix data, where *n* is the length of the field in words. Valid lengths are greater than zero and less than 32768. For example, B20 will create a 40-byte binary field. **[Return]** End definition, create file. **FORMAT** continues to step to the next field until only **[RETURN]** is entered for **ITEM**, or until the field number exceeds 127.

EXAMPLES

```
#FORMAT <P> [10:10] FMTFILE

#FORMAT /H

#FORMAT FILENAME
```

ERRORS

Filename already exists; use "!" to replace

Invalid parameter or syntax for command

File 'filename' syntax error

Invalid! Precision must be 1 thru 4

Invalid! String item must be >0 and less than 65535

Invalid! Binary item must be >0 and less than 32768

See also

MAKEITEM Utility, Numeric Variable Precisions, Creating Formatted ITEM files

KEYMAINT

SYNOPSIS

Analyze/Maintain Indexed Data Files.

SYNTAX

#**KEYMAINT** {*filename*}

DESCRIPTION

filename is the name of an existing UniBasic Indexed data file. *filename* may be in the form *lu/filename*, *pack:filename*, or any Unix full *pathname*. If a *filename* is not entered on the command line, **KEYMAINT** prompts for its entry.

The following commands are available:

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 last 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 <RETURN> - or - filename-index number <RETURN> The filename may include a packname, logical unit or directory name.
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 Record String	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 Program	Allows you to exit KEYMAINT
Z	Get or Release Record	Get or release records.

(G)et or (R)elease Record

Record Enter G to get a record form 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.

General Guidelines:

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

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

/L	Sends the output to the system printer 'sys/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 /H

KEYMAINT
```

ERRORS

Filename does not exist

File is Write-protected

Selected data record is locked

Index file structure error or svar dim length < Key count

Illegal record number (past end of file)

Key not found

See also

Indexed Files

KILL

SYNOPSIS

Delete a single file or a list of files.

SYNTAX

#**KILL** *filename* {,*filename*} . . .

DESCRIPTION

filename is the name of any file to which you have write permission and wish to remove from the system. It may also include a logical unit, packname or directory specifier.

If a single *filename* is supplied, and it was deleted, the message, "DELETED", is displayed. When a list of *filenames* is specified and all files were deleted, the message, " ALL DELETED", is displayed.

If any *filename* in the list is invalid, **KILL** reports an error and attempts to delete the remaining files in the list. If any *filename* is delete protected, or you do not have write permission to the file, you are prompted with the error description and asked whether to delete the file. An answer of Y attempts to change the attributes and delete the file. Pressing [RETURN] or N skips the file and proceeds with the remainder of the *filenames* in the list.

When **KILL** is issued to an open in-use file, the filename entry is removed from the system directory immediately to prevent further access, but it remains open where in-use. The file is ultimately removed when the last user closes the file. This Unix behavior more closely resembles IRIS behavior. BITS systems did not permit the deletion of any file which was opened and in use.

KILL utilizes the **KILL** statement after parsing individual *filenames* from the command line. If attributes are to be changed on a prompted deletion, the **MODIFY** statement is used.

When operating in the environment **BASICMODE=BITS**, KILL must be preceded by a / (**RUN** command). Otherwise, UniBasic assumes the entry, and attempts execution of a **KILL** statement in *immediate mode*.

(Release 9.1) Can be used with encrypted files without an encryption key.

EXAMPLES

```
KILL ABC

KILL /usr/genled,/usr/PAYROLL

/KILL filename

filename is Delete Protected. Delete (Y/N) Y
```

ERRORS

File is write-protected

Filename does not exist

See also

KILL statement, **MFDEL** Utility

LIBR

SYNOPSIS

Generate an expanded listing of files in a directory.

SYNTAX

#**LIBR** {**switches**}

DESCRIPTION

LIBR is supplied for IRIS programmers to generate a directory listing. **LIBR** uses the Unix '**ls**' command to generate its output. **LIBR** only operates properly if **BASICMODE=IRIS** is enabled.

switches 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 following *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 ^
```

ERRORS

Logical unit not active

Filename in use and no "!" supplied

See also

DIR utility, File Attributes

loadlu

SYNOPSIS

IRIS/BITS Logical Unit to Unix Transfer Utility.

SYNTAX

$ **loadlu** *device dest type*

DESCRIPTION

Load a BITS or IRIS logical unit image from tape. A list of three arguments must follow the command. They are defined as:

device Selects the source containing an IRIS or BITS backup tape. Normally, device specifies the Unix no-rewind tape device name. Many systems prepend an "n" to the device name as the no-rewind device, such as "/dev/nrtp" or "/dev/nrct0"

dest Select the destination *filename* to store the BITS or IRIS tape image, i.e. /usr/ub/tapefile.

type Select the type or format from which the original backup was performed. Available options are:

iris IRIS media.

bits	BITS media.
starcopy	PCBITS media.
dump	Other foreign tapes.

If more than one logical unit resides on a single tape, *loadlu* will stop at the first filemark. To append subsequent logical units, re-issue the command using the no-rewind *device*

EXAMPLES

```
$ loadlu /dev/nrtp lu1 iris
```

```
$ loadlu /dev/nrStp1 pgms bits
```

ERRORS

INDEX header not found within first 99 blocks

Data does not appear to be in proper format

Read returns -1, tape appears to be in wrong format

Cannot find packname on disk

Cannot reopen /dev/xxx for reading

Cannot reopen /dev/

See also

none

lptfilter

SYNOPSIS

Filter ASCII data through a pipe.

SYNTAX

lptfilter [*-ffilename*] [-gn] [*input_byte replacement_string*]

DESCRIPTION

-f filename:	Specify a *filename* which contains multiple *input_byte replacement_string* pairs. *input_byte replacement_string* pairs must be separated by white space (tab, newline, or space). When identical *input_byte replacement_string* pair are specified in both the *filename* and the command line, the pair specified on the command line takes precedence.
$-$gn	Define the mode to translate extended graphical characters. *n* represents one of the following translation methods:

0: (default) No translation is performed on graphic mnemonics. Characters that fall

into the range of graphic mnemonics will be translated normally.

1: Conditional translation of graphic mnemonics. Graphic mnemonics are translated once lptfilter detects a 'BG' (\0236\) mnemonic in the data stream. Graphic mnemonic translation is suspended once lptfilter detects an 'EG' (\0237) mnemonic. All characters between sent between 'BG' and 'EG' and within the range of \0306\ and \0341\ are translated as graphic characters.

2: Unconditional translation of graphic mnemonics. **lptfilter** will translate graphic mnemonics without regard to the presence of BG' or 'EG' mnemonics.

input_byte = Input byte to be translated as one of the following:

1. Two-letter UniBasic mnemonic. For example: 'BX', 'BG', 'G1','BU', etc.
2. Single ASCII character such as A, X, G
3. One-byte octal, decimal, or hexadecimal value

Care should be exercised in the representation of input bytes. A single byte, (e.g. \306\), can have two identities - graphic and non-graphic. DCI recommends representing input bytes with the UniBasic mnemonics to avoid confusion.

replacement_string = Replace every occurrence of *input_byte* with this **replacement string**. The *replacement_string* may be a series of ASCII characters or octal, decimal, or hexadecimal values.

lptfilter is generally used to translate certain mnemonics for printers. The utility filters ASCII data, but does not conform directly to IRIS mnemonic values.

All octal, decimal, and hexadecimal values are represented by a backslash (\) followed by a value which determines base. For example:

Octal values are preceded by a "\0":	\020
Decimal values are preceded by a "\":	\16
Hexadecimal values are preceded by a "\0x":	\0x10

EXAMPLES

```
lptfilter BX \010

lptfilter \0x8D \0x0A

lptfilter -g1 -fptr1
```

ERRORS

lptfilter not found

See also

Configuring Printer Drivers, Installing & Configuring UniBasic, CRT mnemonics

MAKE

SYNOPSIS

Create multiple data files with the same attributes.

SYNTAX

#**MAKE** {*<attr>*} *filename* {,*filename* ,. . .}

DESCRIPTION

attr are any optional attributes for the file. The attributes must include the specification of a record count and the record length in the form: *count:length* may be included. The file structure selection must also be given. The structure is indicated with a "T" for tree structure, or "C" for contiguous structure. This indicator is used for the file type. The following other attributes can be included:

- P Public. A public file can be accessed by any account. If P is not selected, only the creator, or a user with a higher privilege access it.
- R Read protect. This attribute makes it impossible for any other accounts to read the file.
- W Write protect. No other account can write data to the file.

EXAMPLE

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

ERRORS

Filename already exist; use "!" to replace

See also

File Attributes, Protections and Permissions

MAKECMND

SYNOPSIS

Generate a command file for BATCH or EXEC.

SYNTAX

#**MAKECMND** { *switches file* **USING** ***DIR****file*} {/**H**}

DESCRIPTION

switches accepts the /**H** help option only.

file selects the *filename* to create and hold a series of commands. *file* is built as a standard Unix text file.

USING ***DIR****file* selects the *filename* of a **DIR /l=file** utility output previously executed.

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** listing.

If no options are present on the command line, the user is prompted for the **file** to create, ***DIR****file*.

The user is prompted to enter a series of commands to apply to each of the *filenames* in the ***DIR****file*. 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 DIR 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**). DIR listing must be /V 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

Create Command File

Press [RETURN] to exit

Line #1: GET ? [RETURN]

Line #2: DUMP TEMP! [RETURN]
```

```
Line #3: GET TEMP [RETURN]

Line #4: CHANGE ? <> [RETURN]

Line #5: (SAV) <?> ?! [RETURN]

Line #6: [RETURN]
```

This command file could be used to dump each file to ASCII type, then reload and save it with the original attributes.

ERRORS

Filename already exists; use "!" to replace

See Also

BATCH, DIR

MAKEHUGE

SYNOPSIS

Convert a Universal file into a Huge Universal file.

SYNTAX

$ **MAKEHUGE** *filename*

DESCRIPTION

MAKEHUGE converts an existing Universal file into a Huge Universal file that supports file sizes larger than 2 gigabytes. The **MAKEHUGE** utility does not change the index node size (ISAMSECT) of an Indexed file. The *ubcompress* utility can be used before or after converting a file to change the index node size. Huge files are not supported on some older operating systems..

EXAMPLES

```
$ MAKEHUGE CUSTHISTORY
```

ERRORS

Command format error

Not a Universal file

Cannot open file

Cannot write to file

See Also

File Attributes

MAKEIN

SYNOPSIS

Build a new UniBasic Index File.

SYNTAX

#MAKEIN

DESCRIPTION

MAKEIN is the standard BITS Indexed File Creation Utility. It is used whenever a standard Indexed File is to be created. The first real data record may be set to zero or 1.

When creating a new file, you may press return for the number of records to allocate, since the files are dynamic.

MAKEIN supports up to 62 indices, 122-byte keys and an unlimited number of records.

The user is prompted to enter the following information:

Prompt	**Information to be Entered**
Filename for indexed file	The desired *filename*, including any *logical unit*, *packname* or directory *pathname*. Append "**!**" to replace an existing file.
File attributes	Protections and controls, if any, that are to be applied to the file. The choices are: PRW. Default is to use the user's current protections. See BITS Attributes section for a complete description.
Number of Data Records	Total number of data records to be contained in the file when created. You may either specify an exact number of records or press return allowing a file to expand dynamically. The Environment Variable **PREALLOCATE** is used to limit the number of records during expansion, automatic pre-allocation and other special Indexed File features, controls and restrictions.
Record length in bytes	The maximum size, in bytes, of the data record. It must be an even number, an odd-sized record is rounded up automatically.
Number of Indices	Total number of indices to be contained in the file. A maximum value of 62 may be entered.
KEY Length (2-122 bytes)	The maximum number of bytes in each **KEY** for each index. If more than one index is defined, you will be prompted to enter the length of each index's key. It must be an even number, an odd-sized key-length is rounded up automatically.
Preallocate data portion	Specify whether to allocate the number of

	records specified during creation. Default is to not pre-allocate the records.
Allocate record zero	Specify whether you wish to use record number zero as the first real data record. Default is to set zero as the first real data record.

EXAMPLES

```
#MAKEIN

Index File creation package.

Filename for indexed File ICFILE

File attributes P

Number of user data records [return] Dynamic

Record length in Bytes 512

Number of Indices 2

#1: KEY Length (2-122 bytes) 10

#2: Key Length (2-122 bytes) 24

Pre-allocate the data portion of the file (Y-N/N) [RETURN]

Allocate record zero (Y-N/N) [RETURN]

Creating and structuring the file, Hold on;

Index File "icfile" has been created.
```

ERRORS

File already exists, use "!" to replace

Value out of range

See Also

BUILDXF, Indexed Data Files, **PREALLOCATE**

MAKEKEY

SYNOPSIS

Create or modify a key file for use with encrypted files.

SYNTAX

#MAKEKEY

DESCRIPTION

The **MAKEKEY** utility is an interactive program used to create or modify key files. Key files are used to supply the encryption keys used to open or create encrypted files (see Encrypted Files). Key files allow programs to transparently open, read, and write encrypted files without having each program explicitly define encryption keys. The

path of the key file is normally contained by the environment variable **UBKEYFILE** so that UniBasic can read the key file during initialization. There are two forms of key files: standard key files and master key files. Standard key files are locked to a specific system license number (see **SSN**) and can be loaded into the current key list automatically via the **UBKEYFILE** environment variable. Master key files are used to move key lists between systems and to create standard key files. Both types of key files can themselves be encrypted by passphrases.

The following commands are available:

Name	Description
LOAD	Read keys from a master key file into the current key list.
SAVE	Write the current key list to a master key file.
MAKE	Write the current key list to a key file.
ADD	Add a new encryption key definition to the current key list.
CHANGE	Modify an existing encryption key definition in the current key list.
DELETE	Delete an encryption key definition from the current key list.
DELETEALL	Delete all encryption key definitions from the current key list.
PRINT	Display the current key list.
HELP	Display a list of commands.
QUIT	Exit the **MAKEKEY** utility and restore the key list from **UBKEYFILE**.

All commands except **DELETEALL** can be abbreviated using the first letter of the command. Commands can be typed in lowercase or uppercase.

EXAMPLES

```
#MAKEKEY

Makekey - Key File Maintenance Utility

Key list is empty

Command (enter "help" to display commands)?a

Key name? examplekey

Supported ciphers:

1.   AES-256

2.   AES-128
```

```
3.   3DES
4.   DES
Cipher number? 2
Passphrase?
Repeat passphrase?
Name         Cipher      Passphrase
-----------------------------------
EXAMPLEKEY   AES-128     ***************

Command (enter "help" to display commands)?s
Name/path of master file? examplemaster
Passphrase?
Repeat passphrase?
Master file successfully written.
Name         Cipher      Passphrase
-----------------------------------
EXAMPLEKEY   AES-128     ***************

Command (enter "help" to display commands)?deleteall
Key list is empty.
Command (enter "help" to display commands)?l
Path of master file (type ENTER to abort)? examplemaster
Passphrase?
1 record added.
Name         Cipher      Passphrase
-----------------------------------
EXAMPLEKEY   AES-128     ***************

Command (enter "help" to display commands)?m
Name/path of key file? examplekeys
Passphrase (type ENTER if none)?
File label? Example of a key file
Key file successfully written.
Name Cipher Passphrase
Name         Cipher      Passphrase
-----------------------------------
EXAMPLEKEY   AES-128     ***************
```

```
Command (enter "help" to display commands)?q
```

ERRORS

Passphrase is too short (minimum 8 characters)

Passphrases do not match

Error writing master file

Unable to create file

Invalid cipher number

See Also

Encrypted Files, Indexed Files, **SYSTEM** statement

makeosn

SYNOPSIS

Create an OSN to activate runtime or listing of psaved programs.

SYNTAX

$ **makeosn** {*switches*}

DESCRIPTION

makeosn, which comes with the Passport product, is used in conjunction with a DCI supplied Product Description Number (PDN) to create multiple OSN (OEM Security Numbers). An OSN enables the runtime and/or listing of programs encrypted by **PSAVE** , as well as operation of the **PSAVE** command itself.

The *-m* switch provides for the creation of both a master and user OSN.

EXAMPLES

```
$ makeosn

$ makeosn -m
```

ERRORS

Makeosn not running as root

No company/product description entered

Invalid PDN encryption

See Also

PSAVE, OEM

makesp

SYNOPSIS

Create a system BASIC program.

SYNTAX

$ **makesp** *filename* {*filename . . .*}

DESCRIPTION

filename is the name of any saved BASIC program which is to become a system command.

makesp converts saved BASIC programs into system programs by changing their type from **SAVE** to **SYST**. System BASIC programs are treated as commands instead of programs when specified in a **CHAIN** statement. UniBasic closes all channels and exits to *command mode* copying the entire **CHAIN** command into the input buffer for execution as a command.

This feature is used to create system commands and utilities from BASIC programs. The **LIBR, COPY, KILL, PORT**, and **TERM** utilities included with UniBasic are a few example system programs written in BASIC.

Setting the <O> overlay attribute in system programs preserves the current program running in memory when entered directly from command mode.

EXAMPLES

```
$ makesp program1

$ makesp program1 program2 program3
```

ERRORS

Cannot open

No such file or directory

See Also

CHAIN, File Attributes

MFDEL

SYNOPSIS

Delete a list of files simultaneously.

SYNTAX

#**MFDEL** *command list*

DESCRIPTION

command 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 ***DIR****file*. Any *@dirname@* selection is overridden for the files within the ***DIR****file*.

EXAMPLES

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

ERRORS

File not found

Command format error

See Also

KILL utility, **KILL** statement

PORT

SYNOPSIS:

Query or Change a Port's Status.

SYNTAX

#**PORT** {*port-range*} {**EVICT**} {**MONITOR | M | ACTIVITY**}

DESCRIPTION

port-range Specifies the range of *port numbers* to operate upon. Valid range is 0 thru 4095. The *port-range* may be a single *port number*, *x-y* to select ports x through y inclusive and @ or ALL for all valid *port numbers*.

EVICT Evict - sign off all *port numbers* selected by port-range.

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

Port	The UniBasic port number.
Group	The Group Number a particular user is assigned to.
User	The User Number a particular user is assigned to.
Processor	The Unix Process running, UniBasic.
Program	The program running under UniBasic. If a port is at command mode or at SCOPE, the display is empty for that port's program.

EXAMPLES

```
#PORT ALL MONITOR

#PORT 20 EVICT

#PORT 132 ACTIVITY
```

ERRORS

Illegal Port; range allowed: (0-4095)

Illegal command

See Also

TERM Utility

QUERY

SYNOPSIS

Obtain detailed information about a file.

SYNTAX

#**QUERY** {/*switches*} {*Filename*}

DESCRIPTION

The optional *switches* may be used to as follows:

Switch	Meaning
?	Print instructions for using QUERY. Instructions are also printed when no options are entered on the command line.
@	Output the available disk space. Performs the Unix "df -t"
FILE	Scan the file and print historical information.
-L	If *FILE* is an Indexed File, compute and display the number of keys in each directory. This option may take several minutes to complete.

filename selects any filename, lu/filename, pack:filename, or full Unix pathname to be queried.

EXAMPLES

```
#QUERY ICFILE

#QUERY -L ICFILE

#QUERY @

#query ar.customers

AR.CUSTOMERS is a UniBasic INDEXED file with 500 records of 335
```

```
words each
Size of UniBasic header: 512 bytes.
There are: 2 indices, 340 active records of 670 bytes each
INDEX KEY LEN WORDS   KEY LEN BYTES   ACTIVE KEYS
    1             3               6   Issue QUERY -L FILE
    2             9              18   Issue QUERY -L FILE
Created on UniBasic Level: 5.1
Full path and filename: /usr/dci/files/ar.customers
Priv: none, Account Group: 102, User: 204
Protection: <60> Unix: <660> Additional Attr:
, Size: 701 Blocks</font>

Creation: 12-07-93
Accessed: 12-07-93 (0) hours ago
```

ERRORS

Command format error

Filename does not exist

Protected filesystem or directory

File is Read-protected

See Also

SCAN Utility, Files

SCAN

SYNOPSIS

Obtain detailed information about a file.

SYNTAX

#**SCAN** {*switches*} { *directory* } {*filename* | ***DIR****file* } . . .

DESCRIPTION

If no *switches* or *filenames* are entered on the command line, the user is prompted for *filename* to be interrogated. 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=$name	Re-direct all output to the named *pipe$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.
^DIR*file*	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

#SCAN 1/data1

#SCAN /H
```

ERRORS

Command format error

Filename does not exist

Protected filesystem or directory

File is Read-protected

See Also

QUERY Utility, Files

TERM

SYNOPSIS

Query or change a port's status.

SYNTAX

#**TERM** {*port-range*} **COMMAND** {*parameters*}

DESCRIPTION

port-range	Specifies the range of *port numbers* to operate upon. Valid range is 0 thru 4095. The *port-range* may be a single *port number*, *x-y* to select ports x through y inclusive and @ or ALL for all valid *port numbers*.
EVICT	Evict - sign off all *port numbers* selected by *port-range*. The letter ***E*** may replace the word **EVICT**.
MONITOR	Monitor the activity of all *port numbers* selected by *port-range*. The letter ***M*** may replace the word **MONITOR**. Monitor mode displays the following

information:

- UniBasic Port number
- tty name or number
- Default pathname
- User account number
- Operational mode
- Channel in use
- Program and files currently in use

The following two optional *parameters* may used in conjunction with the ***M* COMMAND**.

'F' Output all channels and files currently opened for each UniBasic port.

'C' Causes continuous monitoring, repeating every 10 seconds.

A port can be shown to be in one of three operating modes:

Mode	Description
Cmnd	Prompt mode, waiting for a command or statement
Run	BASIC program execution
List	BASIC program being listed

EXAMPLES

```
TERM ALL EVICT

TERM @ E

TERM 20 M

TERM @MF
```

ERRORS

Illegal command, "TERM /H" for list of commands

See also

PORT Utility

ubcompress

SYNOPSIS

Compress UniBasic Indexed Files.

SYNTAX

$ **ubcompress** [*-t tempdir*] [*-v*] *filenames*

DESCRIPTION

ubcompress reduces the size occupied by the index portion of a UniBasic index file. If any errors occur during compression, the original file is unaffected.

There must be sufficient disk space for the original and a temporary index portion at the same time. It may be useful to list the files in order of smallest-to-largest to avoid running out of disk space when processing the larger files.

-t tempdir specifies the directory for temporary files.

-v displays the file size and prompts user before replacing.

If your hard disk is separated into several file systems, you may need to specify the directory in which to build the temporary file, using either the environment variable **TMPDIR**, or the command line option "-t". File types other than UniBasic indexed files are ignored. The default temporary directory is */usr/tmp*.

This utility should be run periodically on files subjected to substantial insertion and deletion of keys. Files which insert and delete relatively sequential keys, such as order files, temp files etc. will benefit most from **ubcompress**.

EXAMPLES

```
ubcompress /usr/ub/1

ubcompress -t /u/tmp -v /u/ub/files

ubcompress /usr/*
```

ERRORS

Cannot generate temporary filename

Cannot open filename: c-tree error

Cannot create index file

Cannot initialize additional index number X

Cannot reopen index file

Cannot add key to index

Cannot close temporary file

Cannot close original file

Cannot replace original file

Cannot set original file mode

Cannot set original file ownership

See also

Indexed Files and Universal Files.

ubconvertfiles

SYNOPSIS

Convert UniBasic file(s) to Universal file(s).

SYNTAX

$ **ubconvertfiles** { *-h* | *-i x* | *-n* | *-o dir* | *-t x* | *-v n* | *-C h* | *-F* | *-V*} *filenames*

DESCRIPTION

ubconvertfiles converts non-Universal UniBasic Indexed, Contiguous, and Formatted file(s) to Universal file(s). The source file(s) must be UniBasic BCD file(s) and must be converted on the native platform. The Indices must not contain IRIS style keys (**PREALLOCATE** option 64) unless the "-i k" option is set and the keys consist only of printable characters. Binary data (e.g. packed data) should be avoided for maximum platform independence.

The converted file(s) may be read on a different hardware platform using UniBasic and dL4. The converted file(s) may also be read on a Microsoft Windows system using version 3.0 and higher of dL4 for Windows.

It is recommended that you have a current backup of the file(s) to be converted before processing. Also, verify that the file(s) to be converted are currently not in use by someone else. In other words, no one should have the file(s) open.

Note: You may use the ubrebuild utility to verify the integrity of an Indexed file's deleted record list before running this program.

---

filenames is a space separated list of files to convert to UniBasic Universal data files. Wildcards characters are accepted in *filenames*.

The command line options are:

-h print help.

-i x where 'x' is case insensitive and specifies the option(s) for Indexed Contiguous files. Options c, e, f, and p are mutually exclusive, but may be combined with d to produce the desired results.

- c converted file will have no deleted record list.
- e abort on corrupted deleted record flag; file is not converted.
- f ignore any deleted record flag error; convert file with a possibly corrupt deleted record list.
- k convert IRIS style keys ("k" attribute). The keys must not contain non-printable characters.
- p on deleted record flag error, stop building the deleted record list and retain the portion of the list already

processed. (default)

d disable "records-in-use" count for dL4 files (non-default). Disabling the count will increase file performance when deleting or adding records. It will also, however, cause **SEARCH** statement mode 1, index 0 to report an incorrect result for STATUS=1 (number of available records) and STATUS=7 (number of records in use). In both cases the result will assume that there are no deleted records in the file. Disabling the count does not prevent the reuse of deleted records; it only effects the **SEARCH** functions that return record counts. Note that the "number of records available" is, in fact, always inaccurate because Universal files are dynamically expandable up to the amount of disk space available. The **QUERY** utility uses the SEARCH statement and will display an incorrect active record count for any file with the "records-in-use" count disabled.

-n report information about the file(s) without performing a conversion. May be used to test file(s) and/or gather information on file(s). The information reported is the file name, file type, file format, UniBasic release level, workspace, convertible status, and summary of file(s) processed. For example:

```
cust.master

   Type: INDEXED

   Format: BCD

   Created on UniBasic Level: 5.5

   Work space needed: 2936

   Convertible

1 file(s) processed

All file(s) are convertible
```

The listing of *filenames* as generated by the ls command may be redirected to a file, the file inspected and edited as needed, then used as input to the ubconvertfiles utility. For example:

```
ls * >files
```

Creates the file '*files*' which contains the name of all the files in the present working directory. This file may be inspect and/or edited, then used as the source of *filenames* to the ubconvertfiles utility as follows:

```
cat files | xargs ubconvertfiles -v9
```

-o dir build the Universal file(s) in directory **dir** and keep the original file(s) unchanged.

-t x where **x** is case insensitive and specifies the file type to convert. Multiple options may be specified.

- i for Indexed
- c for Contiguous
- f for Formatted

-v n verbose mode 'n' where 'n' may be 0, 1, 2, 3, 4, 5, or 9. Each verbose mode outputs its own statistics plus those of the lower modes. Examples may be found in the Statistical Reports section of the ubconvert User's Guide.

-C h specify dL4 Character set 'h'

-F force conversion of non-BCD file(s).

-V print version number of this utility.

EXAMPLES

```
ubconvertfiles -h

ubconvertfiles -o /tmp /usr/ub/files/ar1

ubconvertfiles -t c /usr/ub/data/sales2
```

ERRORS

Error processing file 'filename': detected non-convertible file type: Unix

Error processing file 'filename': is a BITS file

Error processing file 'filename': file contains IRIS style keys

See also

Contiguous Files, Formatted Files, Indexed Files, and Universal Files in the UniBasic Reference Guide.

ubrebuild

SYNOPSIS

Rebuild a UniBasic Indexed File deleted record list.

SYNTAX

$ **ubrebuild** {*-c* } {*-r* } {*-s* } {*-f* } { *-v* } *filenames*

DESCRIPTION

ubrebuild rebuilds the deleted record list within a non-Universal Indexed file. When the deleted record list is damaged, your UniBasic application may receive a c-tree error 31.

A scan of the free record chain is performed and the file is rebuilt based on the actual deleted record marks within the file, thus ignoring the existing pointers in the free chain.

Several options are available in determining how to rebuild a file. The command line options are:

- -v verbose mode, displays data during operation.
- -c unconditionally clear the delete list. The next record allocated will extend (add) a new record to the file. Deleted records are orphaned.
- -r unconditionally clear the delete list. Scan all records in the file rebuilding the delete list
- -s scan delete list for errors. Report any errors and proceed to clear or rebuild only if errors were found and -r or -c is specified.
- -f force use of a temp file (instead of memory) for -s option.

File types other than UniBasic indexed files are ignored by **ubrebuild**.

EXAMPLES

```
ubrebuild –c /usr/ub/files

ubrebuild –s /usr/ub/files/ar1

ubrebuild *
```

ERRORS

Cannot gain exclusive access to filename

No record length

Unable to create temporary file

See also

Indexed Files, Universal Files, SEARCH Statement, C-tree errors

ubterm

SYNOPSIS

Create UniBasic Terminal Definition File.

SYNTAX

$ **ubterm** {*terminal ...*}

DESCRIPTION

terminal is the name of a terminal having a corresponding Unix Terminfo driver. An error is printed if the **TERMINFO** name is undefined, if the *sys* directory cannot be found in any path within **LUST**, or permission to create a file in the *sys* directory is denied.

To create a UniBasic term file from a Unix Terminfo driver, a minimum set of functions (cursor addressing & clear screen) must be contained in the Terminfo definition. A list of UniBasic mnemonics

not currently defined in the Terminfo driver is displayed. Specifically, the mnemonic **CU** (clear unprotect) is not defined by Terminfo and should be added manually.

UniBasic *term* files are maintained using a standard editor, like "**vi**".

EXAMPLE

```
$ ubterm tvi925 wyse60
```

ERRORS

Cannot locate the TERMINFO driver

No sys/ directory found in "LUST" environment variable

Cannot open output file sys/term.xxx

See also

Installing UniBasic, **CRT $TERM** files

ubtestlock

SYNOPSIS

Diagnostic Program to Test Record Locking.

SYNTAX

$ **ubtestlock** *filename byte-offset length* { ... }

DESCRIPTION

ubtestlock is a diagnostic program to test record locking in a network environment. It may be executed from the Unix command prompt. It performs the following five operations, any of which may report a system error:

1. open the file
2. seek to the *byte-offset*
3. read the *length* number of bytes
4. read-lock (without wait)
5. write-lock (without wait)

If the system is configured correctly for record locking no messages are returned from the program and the system prompt ($) is displayed. Any failures must be analyzed to determine if the read-lock or write-lock operation was the actual cause of the failure. The following is a typical message that indicates record locking is not available:

```
Read lock error: No locks available
```

filename is any absolute or relative path and name of a file that exist on the file system for which the user has read and write access permissions.

byte-offset is a numeric value specifying the starting location in filename at which to seek and perform the read, read-lock, and write-lock operations.

length is a numeric value specifying the number of bytes to read. A warning is reported if *length* specifies bytes beyond the end-of-file (EOF). For example, if test1 is a 40 byte file and the following command is issued:

```
$ ubtestlock test1 5 40
```

The following message is reported:

```
Warning: read request 40, actual read 36 in file test1
```

EXAMPLE

```
$ ubtestlock testfile 0 10

$ ubtestlock testa 0 10 testb 20 10
```

ERRORS

Unable to open file 'testfile': No such file or directory

Unable to open file 'testfile': Permission denied

Read lock error: Link has been severed

See also

none

WHO

SYNOPSIS

Displays information about your UniBasic process.

SYNTAX

#**WHO**

DESCRIPTION

WHO displays the following information about your UniBasic process:

Port	The UniBasic port number
CPU Secs	(not used)
Connect	The UniBasic 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)

EXAMPLE

```
*WHO : Port 7 CPU Secs: 0.0 Connect: 10:17 Time: 15 April 1993
16:20:47 Disk: User [101-101] Default: /usr/ub/sys Total Used:
-1, Limit: -1, Left: -1
```

ERRORS

None

See also

TERM Utility, **PORT** Utility

Appendix A - ASCII CODES

ASCII, an acronym for American Standard Code for Information Interchange, is a 7-bit representation for data transmission. ASCII characters stored internally conform to 7-bit ASCII industry standard. 8-bit ASCII characters are reserved for graphics, and *crt mnemonics*.

As discussed in Internal Representation of ASCII Characters(f), characters are toggled from the familiar IRIS/BITS 8-bit to the internal 7-bit form.

In the following table, **INT** refers to the internal storage of the character, **EXT** the external (program) format. ASCII codes are shown in both octal and decimal. **CT** is used to indicate a CONTROL character. All codes shown are for the printable character set. INT codes greater than 128 (200 octal) or EXT codes less than 128 (200 octal) represent *CRT mnemonics*.

INT/EXT

OCTAL	**DECIMAL**	**Key**	**Char**	**Comments**
000/200	000/128	CT-@	NUL	Null, tape feed.
001/201	001/129	CT-A	SOH	Start heading.
002/202	002/130	CT-B	STX	Start text.
003/203	003/131	CT-C	ETX	End text/message; EOM.
004/204	004/132	CT-D	EOT	End of transmission.
005/205	005/133	CT-E	ENQ	Enquiry.
006/206	006/134	CT-F	ACK	Acknowledge.
007/207	007/135	CT-G	BEL	Ring bell.
010/210	008/136	CT-H	BS	Backspace.
011/211	009/137	CT-I	HT	Horizontal tab.

012/212	010/138	CT-J	LF	Line feed.
013/213	011/139	CT-K	VT	Vertical tab.
014/214	012/140	CT-L	FF	Form feed.
015/215	013/141	CT-M	CR	Carriage return.
016/216	014/142	CT-N	SO	Shift out.
017/217	015/143	CT-O	SI	Shift in.
020/220	016/144	CT-P	DLE	Data link escape.
021/221	017/145	CT-Q	DC1	XON.
022/222	018/146	CT-R	DC2	AUX ON.
023/223	019/147	CT-S	DC3	XOFF.
024/224	020/148	CT-T	DC4	AUX OFF.
025/225	021/149	CT-U	NAK	Negative ack.
026/226	022/150	CT-V	SYN	Synchronous idle.
027/227	023/151	CT-W	ETB	End block/ medium;LEM.
030/230	024/152	CT-X	CAN	Cancel.
031/231	025/153	CT-Y	EM	End of medium.
032/232	026/154	CT-Z	SUB	Substitute.
033/233	027/155	CT-[	ESC	Escape.
034/234	028/156	CT-\	FS	File separator.
035/235	029/157	CT-]	GS	Group separator.
036/236	030/158	CT-^	RS	Record separator.
037/237	031/159	CT-_	US	Unit separator.
040/240	032/160	space	SP	Space
041/241	033/161		!	Exclamation point
042/242	034/162		"	Quotation mark
043/243	035/163		#	Pound sign
044/244	036/164		$	Dollar sign
045/245	037/165		%	Per cent symbol
046/246	038/166		&	Ampersand
047/247	039/167		‘	Accent acute or '
050/250	040/168		(	Left parenthesis
051/251	041/169		)	Right parenthesis
052/252	042/170		*	Asterisk
053/253	043/171		+	Plus sign
054/254	044/172		,	Comma

055/255	045/173	-	Minus sign or hyphen
056/256	046/174	.	Period
057/257	047/175	/	Forward slash
060/260	048/176	0	Numeral zero
061/261	049/177	1	Numeral one
062/262	050/178	2	Numeral two
063/263	051/179	3	Numeral Three
064/264	052/180	4	Numeral four
065/265	053/181	5	Numeral five
066/266	054/182	6	Numeral six
067/267	055/183	7	Numeral seven
070/270	056/184	8	Numeral eight
071/271	057/185	9	Numeral nine
072/272	058/186	:	Colon
073/273	059/187	;	Semi-colon
074/274	060/188	<	Less than symbol
075/275	061/189	=	Equals sign
076/276	062/190	>	Greater than symbol
077/277	063/191	?	Question mark
100/300	064/192	@	At sign
101/301	065/193	A	Uppercase letter A
102/302	066/194	B	Uppercase letter B
103/303	067/195	C	Uppercase letter C
104/304	068/196	D	Uppercase letter D
105/305	069/197	E	Uppercase letter E
106/306	070/198	F	Uppercase letter F
107/307	071/199	G	Uppercase letter G
110/310	072/200	H	Uppercase letter H
111/311	073/201	I	Uppercase letter I
112/312	074/202	J	Uppercase letter J
113/313	075/203	K	Uppercase letter K
114/314	076/204	L	Uppercase letter L
115/315	077/205	M	Uppercase letter M
116/316	078/206	N	Uppercase letter N
117/317	079/207	O	Uppercase letter O

120/320	080/208	P	Uppercase letter P
121/321	081/209	Q	Uppercase letter Q
122/322	082/210	R	Uppercase letter R
123/323	083/211	S	Uppercase letter S
124/324	084/212	T	Uppercase letter T
125/325	085/213	U	Uppercase letter U
126/326	086/214	V	Uppercase letter V
127/327	087/215	W	Uppercase letter W
130/330	088/216	X	Uppercase letter X
131/331	089/217	Y	Uppercase letter Y
132/332	090/218	Z	Uppercase letter Z
133/333	091/219	[	Left bracket.
134/334	092/220	\	Backslash.
135/335	093/221	]	Right bracket.
136/336	094/222	^	Up arrow, caret.
137/337	095/223	_	Underscore.
140/340	096/224	‘	Accent grave.
141/341	097/225	a	Lowercase letter A
142/342	098/226	b	Lowercase letter B
143/343	099/227	c	Lowercase letter C
144/344	100/228	d	Lowercase letter D
145/345	101/229	e	Lowercase letter E
146/346	102/230	f	Lowercase letter F
147/347	103/231	g	Lowercase letter G
150/350	104/232	h	Lowercase letter H
151/351	105/233	i	Lowercase letter I
152/352	106/234	j	Lowercase letter J
153/353	107/235	k	Lowercase letter K
154/354	108/236	l	Lowercase letter L
155/355	109/237	m	Lowercase letter M
156/356	110/238	n	Lowercase letter N
157/357	111/239	o	Lowercase letter O
160/360	112/240	p	Lowercase letter P
161/361	113/241	q	Lowercase letter Q
162/362	114/242	r	Lowercase letter R

163/363	115/243		s	Lowercase letter S
164/364	116/244		t	Lowercase letter T
165/365	117/245		u	Lowercase letter U
166/366	118/246		v	Lowercase letter V
167/367	119/247		w	Lowercase letter W
170/370	120/248		x	Lowercase letter X
171/371	121/249		y	Lowercase letter Y
172/372	122/250		z	Lowercase letter Z
173/373	123/251		{	Left brace
174/374	124/252		|	Vertical bar
175/375	125/253		}	Right brace
176/376	126/254		~	Alt mode, Tilde
177/377	127/255	RUB	DEL	Delete, rubout

Appendix B - CRT Mnemonics

The following table shows the mnemonic code, \xxx\ octal format used within an application as an equivalent to the Code, the Internal value transmitted to file or device for the mnemonic, and a brief description of mnemonic. For a complete description of the mnemonics, refer to CRT Expressions and Mnemonics.

Code	\xxx\	Internal	Description
ET	003	203	ETX Code.
RB	007	207	Ring Terminal Bell.
ML	010	210	Move Cursor Left.
TF	011	211	Tab Forward to next tab stop (BITS)
LF	012	212	Line Feed.
VT	013	213	Vertical Tab
FF	014	214	Form Feed
CR	015	215	Carriage Return.
MH	017	217	Move Cursor Home.
CS	020	220	Clear Screen.
S1	021	221	Special user code 1.
S2	022	222	Special user code 2.
S3	023	223	Special user code 3.

S4	024	224	Special user code 4.
ES	025	225	End Status Line definition.
SO	026	226	Status On.
SF	027	227	Status Off
WS	030	230	Write Status Line.
K0	031	231	Cursor Off.
K1	032	232	Cursor Blinking Block.
K2	033	233	Cursor Steady Block.
K3	034	234	Cursor Blinking Underline.
K4	035	235	Cursor Steady Underline
BG	036	236	Begin Graphics (Extended Graphics).
EG	037	237	End Graphics.
MR	040	240	Move Cursor Right one position.
RD	041	241	Read Current Cursor position.
EF	042	242	End Function Key definition.
CU	043	243	Clear Screen Unprotected.
CL	044	244	Clear to End-of-Line (unprotected).
CE	045	245	Clear to end-of-screen (unprotected).
P1	046	246	Program Function Key 1.
P2	047	247	Program Function Key 2.
P3	050	250	Program Function Key 3.
P4	051	251	Program Function Key 4.
MD	052	252	Move Cursor Down 1 line.
MU	053	253	Move Cursor Up 1 line.
P5	054	254	Program Function Key 5.
P6	055	255	Program Function Key 6.
P7	056	256	Program Function Key 7.
P8	057	257	Program Function Key 8.
BB	060	260	Begin Blink mode.
EB	061	261	End Blink mode.
BR	062	262	Begin Reverse Video mode.
ER	063	263	End Reverse Video mode.
BD	064	264	Begin Dimmed Intensity mode.
ED	065	265	End Dimmed Intensity mode.
BP	066	266	Begin Protected Field mode.

EP	067	267	End Protected Field mode.
BU	070	270	Begin Underline mode.
EU	071	271	End Underline mode.
BX	072	272	Begin Expanded Print mode.
EX	073	273	End Expanded Print mode.
FM	074	274	Enter Format mode.
FX	075	275	Exit Format mode.
LK	076	276	Lock Keyboard.
UK	077	277	Unlock Keyboard.
BT	100	300	Begin Transmission from CRT memory.
MP	101	301	Use Memory Pointer instead of cursor for next positioning command.
IL	102	302	Insert Line at current position.
DL	103	303	Delete Line at current position.
IC	104	304	Insert Character at current position.
DC	105	305	Delete Character at current position.
CT	106	306	Clear Tabs {all}.
ST	107	307	Set Tab at current position.
AE	110	310	Auxiliary Port Enable.
AD	111	311	Auxiliary Port Disable.
SL	112	312	Send Line {all}.
LU	113	313	Send Line {unprotected}.
SP	114	314	Send Page {all}.
GR	115	315	Set Color Green.
TB	116	316	Tab Backward to last tab stop.
PI	117	317	Position Indicator Character.
RE	120	320	Set Color Red.
PU	121	321	Send Page {unprotected}.
YE	122	322	Set Color Yellow.
BL	123	323	Set Color Blue.
MA	124	324	Set Color Magenta.
CY	125	325	Set Color Cyan.
WH	126	326	Set Color White.
XX	127	327	Initialize Terminal defaults.
SA	130	330	User-defined code A.

SB	131	331	User-defined code B.
SC	132	332	User-defined code C.
SD	133	333	User-defined code D.
BV	134	334	Box Vertical; Vertical line character.
BH	135	335	Box Horizontal; Horizontal line character.
			136-141 Reserved for future use.
WD	142	342	Set 80-column mode.
NR	143	343	Set 132-column mode.
RF	144	344	Reset Function Keys to defaults.
TL	145	345	Transmit line {unprotected}.
TP	146	346	Transmit line {protected} or Toggle Page.
TR	147	347	Transmit screen {unprotected}.
TS	150	350	Transmit screen {protected}.
PS	151	351	Print contents of Screen.
RS	152	352	Reset Terminal to defaults.
BA	153	353	Begin Transparent Print.
EA	154	354	End Transparent Print.
RV	155	355	Display Reverse Video as light on dark.
NV	156	356	Display Reverse Video as dark on light.
BO	157	357	Begin non-transparent Print.
EO	160	360	End non-transparent Print.
BK	161	361	Perform return without line-feed, Or Set Color Black.
IO	162	362	IOxx mnemonic prefix code.
BPW	173	373	Begin Protect Window Display replaces use of BD dimmed intensity.
EPW	174	374	End Protect Window Display.
PC1	175	375	PC1 Cursor secondary (coordinate separator).
PC2	176	376	PC2 Cursor tertiary (sequence terminator).
PC	177	377	'@' Start of Cursor Address Sequence.

Only the preceding mnemonics and Extended Graphics Mnemonics on the following pages may be defined within a term file. The user cannot define custom mnemonic names.

Combination **IO** mnemonics are represented by the **IO** mnemonic byte followed by an additional byte for 4-letter mnemonics, and two bytes for six letter mnemonics. They have no definition within the *term* file, but are shown here when it may be desirable to use octal form instead of the mnemonic:

Code	**\xxx**	**Internal**	**Description**

IOBE	001	201	Begin Input Echo mode.
IOEE	002	202	End Input Echo mode.
IOBI	003	203	Begin Transparent Input.
IOEI	004	204	End Transparent Input.
IOBO	005	205	Begin Transparent Output.
IOBD	031	231	Enable Destructive Backspace.
IOED	032	232	Disable Destructive Backspace.
IOB\	033	233	Begin Echoing \ on Escape.
IOE\	034	234	End Echoing \ on Escape.
IOCI	035	235	Clear Type-ahead Buffer.
IOBC	036	236	Begin Activate on Control Character.
IOEC	037	237	End Active on Control Character.
IOBX	041	241	Begin XON/XOFF protocol.
IOEX	042	242	End XON/XOFF protocol.
IORS	043	243	Reset IOxx parameters.
IOIH	044	244	Set Input Handler to next byte./

The following mnemonics are accepted, but perform no-operation:

Code	\xxx\	Internal	Description
IOIHIR	001	201	Standard Input Handling
IOIHSM	002	202	SM Basic Input Statement.
IOIHSR	003	203	SM Basic Read Style Input.
IOISSI	004	204	Simple Input; CTRL+S/Q.

Three additional Debugging mnemonics are supported; **HX**, **OC**, and **AS**. They affect the output of the next string variable presenting the entire string (including zero-bytes) in either Hex, Octal or ASCII respectively. They are not definable as output *replacement strings* within the *term* file.

For a complete list of terminal **mnemonics**, see CRT Mnemonics and Expressions.

Appendix C - Error Numbers

When an error is detected during program execution, and error branching using **IF ERR**(s), **ERRSET**(s), or **ERRSTM**(s) is not enabled, the program is terminated to debug mode (files open) and the following message is displayed:

```
Error in stn stn;sub-stn / text
```

where *stn* is the statement number, *sub-stn* the sub-statement number at which the error occurred, and *text* is a message describing the error.

The following table represents the internal UniBasic error returned by the **ERR(0)**(e) function. BASIC Error numbers are positive in the range 1 to 255. Negative error numbers are used to return special Unix **errno** errors which do not map directly to a BASIC error. The corresponding IRIS **SPC(8)**(e) numbers and System Errors may be found on subsequent pages.

The numbering of errors divides them into six groups to aid in program error branching.

Group 1--Encoding Syntax and Command Errors

Error Number	Text Description
1	Unrecognizable word
2	Format error
3	Incorrect parenthesis closure
4	Incorrect subscript closure
5	Line (stn) number is missing or invalid
7	IFs without 'ENDIF' \
8	'ELSE' without 'IF') Only issued by RUN or SAVE
9	'ENDIF' without 'IF' /
10	Too many variables defined, limit is 348
11	Statement not executable in keyboard mode
12	No program in partition
13	Non-existent lines referenced which overlap renumbered lines
15	Invalid character
16	Invalid speed, or invalid command from your port
17	ENTER statement is illegal if not in a subprogram
18	The ENTER statement can only be executed once in a subprogram
19	Program has been corrupted - cannot execute

Group 2--Syntax and Program Structure Errors

Error Number	Text Description

20	Syntax error
21	Syntax error in DEFined function
22	No such line (stn) number
23	Variable not specified
24	User function not defined
25	Illegal function usage
26	'COM' statement out of order
27	'FOR' without a matching 'NEXT'
28	'NEXT' without a matching 'FOR'
29	'RETURN' without a prior 'GOSUB'
30	Number/types of arguments do not match parameter list
31	'Function argument' or 'Statement Mode' out of range
32	String expression not allowed here
33	Syntax error in 'DATA' statement or CRT control string
34	Formatted output overflows output string
35	Variable in CHAIN READ not passed by CHAIN WRITE
36	Variable from CHAIN WRITE not in this program
37	String expression must be used here
38	Variable in CHAIN READ already contains data
39	Variable in CHAIN WRITE contains no data

Group 3--Complexity and Limit Errors

Error Number	Text Description
40	'FOR' statements nested too deep
41	'GOSUB' statement nested too deep
42	User DEFined functions nested too deep
43	Expression too complex for evaluation
45	Arithmetic error - (X/zero, overflow, LOG(0), or SQR -X)
46	User partition space exhausted
47	Execution prohibited from this account
48	Format string is invalid or too complex for evaluation

Group 4--Array and String Errors

Error Number	Text Description
50	Variable precision cannot be changed
51	Attempt to DIMension an existing simple variable
52	Variable name not DIMensioned
53	Array size exceeds initial DIMension
54	Subscript exceeds DIMension
55	Illegal subscript specified
56	Strings can have only one (1) DIMension
57	Parameter variable in ENTER statement has already been allocated
58	String or array variable has not been DIMensioned
59	A string may not be re-DIMensioned

Group 5--Matrix Errors

Error Number	Text Description
60	Same matrix on both sides of 'MAT' is illegal here
61	Matrices have different DIMensions
62	Matrix has zero DIMension; Argument is not a matrix
63	Matrix DIMensions are not compatible for this operation
64	Matrix is not square
65	Matrix cannot be INVerted - has zero DETerminant

Group 6--File and I/O Errors

Error Number	Text Description
70	Filename does not exist
71	Filename already exists; use '!' to replace

72	File in use; cannot CREATE, DELETE, EOPEN or MODIFY
74	File is in use and locked
75	File is delete protected
76	Out of DATA
77	Extra INPUT numeric items; warning only
78	INPUT of wrong type or insufficient
79	DATA of wrong type (numeric/string)
80	Illegal pathname or filename
81	Illegal channel number specified (or ISAMFILES value too small)
82	Protected Directory or file system, access not granted
85	System is out of channels - notify Manager
86	Not a loadable program file - wrong revision, protected or corrupted
87	Selected channel is not open
88	Illegal record number (past end of file)
89	Assigned channel limit exceeded; too many OPEN files
90	File size is too large for system; cannot expand
91	File is Read-protected
92	File is Write-protected
93	Invalid parameter or syntax for command
95	No such logical unit/pack
96	Program is Re-SAVE/COPY protected
98	File system has no available disk space
100	Selected data record is locked
101	File is not Indexed or Mapped
102	Invalid or non-existent Index number selected
104	Invalid or un-implemented user CALL ID number
105	Parameter list overflow
106	Error detected in/by user CALL routine
107	Not enough parameters passed to user CALL
108	User call parameters out of order
110	C-Tree Index File error; print ERR(8) for details, or var DIM < key len
112	CRT X,Y coordinate out of range
114	CRT Type not selected for your port
120	No communication file '/tmp/comm.listx.y'
121	Communication buffer is full

122	Illegal port number selected
131	Program Channel not OPEN; cannot resave until SAVE/PSAVE issued
132	Channel is already OPEN and in-use
133	Illegal item number selected
134	Data does not match item specification and cannot be converted
136	File is being built or deleted
138	Item number is not sequential
139	Subprogram file not found
140	Subprogram file is read protected
141	Subprogram file is not a BASIC program
142	Not a data file (can't OPEN or replace)
144	Cannot cross ISAM record boundary
150	WINDOWS environment variable not defined or count exceeded
151	No open windows
152	Window tracking not enabled
156	Record Not Written to Formatted Item File
157	Data Read error
158	Input timed out
159	File is encrypted
160	Unrecognized encryption key
161	Unsupported encryption method
162	Inaccessible or corrupt key file
253	Not licensed to use this feature
254	ESCape has been pressed and no ESCape branching enabled
255	Security Failure - Grace or Demonstration period has expired

IRIS Error Numbers

When an error is detected during program execution, and error branching using **IF ERR**(s), **ERRSET**(s), or **ERRSTM**(s) is not enabled, the program is terminated to debug mode (files open) and the following message is displayed:

```
Error in stn stn;sub-stn / text
```

where *stn* is the statement number, **sub-stn** the sub-statement number at which the error occurred, and *text* is a message describing the error.

The following table represents the error returned by the **SPC(8)**(e) function. In some cases, several UniBasic errors map to the same **SPC(8)**(e) value. BASIC Error numbers are positive in the range 1 to 255. Negative error numbers are used to return special Unix

errno errors which do not map directly to a BASIC error.

Error Number	Text Description
1	Syntax error
2	Illegal string operation
3	User partition space exhausted
4	Format error
5	Invalid character
6	No such line (stn) number
8	Too many variables defined, limit is 348
9	Unrecognizable word
14	Out of DATA
15	Arithmetic error - (X/zero, overflow, LOG(0), or SQR -X)
15	Matrix cannot be INVerted - has zero DETerminant
16	'GOSUB' statement nested too deep
17	'RETURN' without a prior 'GOSUB'
18	'FOR' statements nested too deep
19	'FOR' without a matching 'NEXT'
20	'NEXT' without a matching 'FOR'
21	Expression too complex for evaluation
23	Array size exceeds initial DIMension
23	Variable precision cannot be changed
23	Attempt to DIMension an existing simple variable
24	Strings can have only one (1) DIMension
25	String variable has not been DIMensioned
27	Syntax error in DEFined function
28	'Function argument' or 'Statement Mode' out of range
28	Assigned channel limit exceeded; too many OPEN files
28	Illegal subscript specified
28	Parameter list overflow
28	Subscript exceeds DIMension
29	Illegal function usage
30	User function not defined

31	User DEFined functions nested too deep
32	Matrices have different DIMensions
33	Matrix has zero DIMension; Argument is not a matrix
33	Variable name not DIMensioned
34	Matrix DIMensions are not compatible for this operation
34	Same matrix on both sides of 'MAT' is illegal here
35	Matrix is not square
36	Invalid or un-implemented user CALL ID number
38	Error detected in/by user CALL routine
38	Not enough parameters passed to user CALL
38	User call parameters out of order
39	Formatted output overflows output string
40	Channel is already OPEN and in-use
41	Illegal pathname or filename
42	Filename does not exist
43	Invalid parameter or syntax for command
44	Not a data file (can't OPEN or replace)
45	File is Read-protected
46	File is delete-protected
49	Selected channel is not open
51	Illegal record number (past end of file)
52	Record Not Written to Formatted Item File
53	Illegal item number selected
53	Item number is not sequential
54	Data does not match item specification and cannot be converted
55	Statement not executable in keyboard mode
56	No program in partition
57	A string may not be re-DIMensioned
58	Format string is invalid or too complex for evaluation
62	Communication buffer is full
62	Illegal port number selected
67	Filename already exists; use '!' to replace
70	DATA of wrong type (numeric/string)
70	Data Read error
76	File is in use and locked

79	Invalid speed, or invalid command from you port
89	Execution prohibited from this account
91	Variable in CHAIN READ not passed by CHAIN WRITE
92	Variable from CHAIN WRITE not in this program
93	Variable in CHAIN READ already contains data
94	Variable in CHAIN WRITE contains no data
95	Input timed out
98	INPUT of wrong type or insufficient
99	ESCape has been pressed and no ESCape branching enabled
100	Illegal channel number specified (or ISAMFILES value too small)
123	Selected data record is locked
150	Program has been corrupted - cannot execute
151	'COM' statement out of order
152	System is out of channels - notify Manager
153	Not a loadable program file - wrong revision, protected or corrupted
154	No communication file '/tmp/comm.listx.y'
155	Program Channel not OPEN; cannot resave until SAVE/PSAVE issued
156	Cannot cross ISAM record boundary
157	C-Tree Index File error; print ERR(8) for details, or var DIM < key len
158	File size is too large for system; cannot expand
159	WINDOWS environment variable not defined or count exceeded
160	No open windows
161	Window tracking not enabled
162	File is not Indexed or Mapped
163	Invalid or non-existent Index number selected
164	CRT X,Y coordinate out of range
165	CRT Type not selected for your port
201	IFs without 'ENDIF'
202	'ELSE' without 'IF'
203	'ENDIF' without 'IF'
206	Subprogram file not found
208	Number/types of arguments do not match parameter list
209	ENTER statement is illegal if not in a subprogram
212	Subprogram file is read protected
213	Subprogram file is not a BASIC program

216 Parameter variable in ENTER statement has already been allocated

217 The ENTER statement can only be executed once in a subprogram

273 Not licensed to use this feature

283 File is encrypted

284 Unrecognized encryption key

285 Unsupported encryption method

286 Inaccessible or corrupt key file

System Error Numbers

When an error is detected during program execution, and error branching using **IF ERR**(s), **ERRSET**(s), or **ERRSTM**(s) is not enabled, the program is terminated to debug mode (files open) and the following message is displayed:

```
Error in stn stn;sub-stn / text
```

where *stn* is the statement number, *sub-stn* the sub-statement number at which the error occurred, and *text* is a message describing the error.

All system errors are returned as negative numbers. Each error represents the Unix **errno** value returned from a system call or function. If possible, **errno** is converted into a standard BASIC error. If no error matches the condition, **errno** is negated and returned for either **ERR(0)** and **SPC(8)**. For further information, refer to your system documentation.

Appendix D - Port as Device

The IRIS Port as a Device Driver (PDn), has been implemented as a user call under UniBasic. All five functions, **Open, Close, Read, Write** and **Print** are available. These functions are used to Read or Write to another serial port.

The PDn routine uses a circular buffer to capture data from the port. The default size for this circular buffer is the same as the controlling port's **INPUTSIZE** environment variable. The circular buffer size can be changed explicitly by setting a new environment variable, **PDNBUFSIZE**, to a value different than **INPUTSIZE**.

The UniBasic PDn driver works with both interactive and non-interactive ports. However, it is the user's responsibility to assure the port being used is disabled and the port is not in use by a non-PDn process prior to accessing, such as **uucp** or **cu**.

CALL $DEVOPEN

The **PD**n port must be opened prior to any other operations. The syntax for opening a port is:

CALL $DEVOPEN, *chan.num*, "*$PDn*"

chan.num is a pseudo device channel number. This channel number is different from the UniBasic file channel number in that UniBasic statements that affect channels, such as

CLOSE, will not affect this channel. The pseudo channel number must be between 0 and 99 inclusive.

$PD*n* has the same syntax as IRIS. For example, $PD2 will open $PD2 and $PD3 will open $PD3. An equivalent environment variable without the leading dollar sign ($) must define the associated UNIX port. This environment variable should be set in user's .profile file. An example of PDn port configuration follows:

```
PD4=/dev/tty004; export PD4
```

Failure to set the appropriate environment variable results in a UniBasic error.

The PDn open routine creates a **LCK..ttyxxx** file to indicate a busy port. This lock file is created in /tmp directory. Users can change the default /tmp directory by setting **LOCKDIR** environment variable. The user's process ID is written in the lockfile.

CALL $DEVCLOSE

Unlike UniBasic, the user must explicitly call the close routine. The syntax to close a pseudo channel is:

CALL $DEVCLOSE,{*chan.num*}

chan.num is an optional pseudo device channel number used during **OPEN**. All pseudo channels are closed if channel number is omitted.

The PDn close routine removes the lock file in addition to closing the port. If the **CALL $DEVCLOSE** is not issued, the lock files will not be removed when UniBasic is terminated or a UniBasic process is killed, including "kill -15 PID".

CALL $DEVREAD

The syntax to read a port is:

CALL $DEVREAD, *chan.num*, *rec #*, *offset #*, *time-out*, *str.var*

chan.num pseudo device channel number

rec # record number, must be set to -1

offset # byte offset number, must be set to -1

time-out time-out value, must be set to -1

str.var string variable to store the contents of data coming from *chn.num*

rec #, *offset #*, and *time-out* must all be set to negative one (-1), as the **CALL** does not allow these parameters to be defined.

CALL $DEVREAD supports the various modes that may be set using **CALL $DEVPRINT**, such as: echo on (IOBE), echo off (IOEE), binary input (IOBI), activate on control characters (IOBC), input len and input time out. Only string variables are allowed since numeric variables are not supported for PDn driver under IRIS.

Like IRIS, a Basic error 95 is returned if a read request cannot be satisfied in the allocated time.

Input character translations, similar to UniBasic Input handling, are also done during a **READ**. However, Cursor tracking and hot-key swapping are not available. A hot-key swap character is disregarded in the event it is read.

CALL $DEVPRINT

The syntax to print to a port is:

CALL $DEVPRINT, *chan.num*, *rec #*, *offset #*, *time-out*, *str.var*

chan.num pseudo device channel number

rec # record number, must be set to -1

offset # byte offset, set to either -1 or 225

time-out time out, must be set to -1

str.var any string variable to be sent to specified *chan.num*

rec #, *offset #* and *time-out* are usually set to negative one (-1), as IRIS does not allow these parameters to be defined. An exception to this rule for PDn under UniBasic is while printing a terminal mnemonic. The *offset #* should be set to 225 to indicate CRT mnemonics. For example,

```
CALL $DEVPRINT,1,-1,225,-1,'IOCI'
```

CALL $DEVPRINT does not allow **PRINT USING** or **TABS**. Only string variables are allowed, although IRIS allowed numeric data to be printed.

CALL $DEVWRITE

The **$DEVWRITE** call has two separate modes of operation. One for actual writing to a port, and another for setting various parameters on a port.

The syntax to write to a port is:

CALL $DEVWRITE, *chn.num*, *rec#*, *offset#*, *time-out*, *str.var*

chan.num pseudo device channel number

rec # Must always be set to -1

offset # Must always be set to -1

time-out Time-out must always be set to -1

str.var Any string variable to be sent to specified *chan.num*

Record number, byte offset and time-out must all be set to negative one (-1) while writing a string variable, as the **CALL** does not allow these parameters to be defined.

A numeric variable cannot be written to a port.

The syntax to change different modes of a port are as follows:

CALL $DEVWRITE, *chn.num*, *rec#*, *mode*, *time-out*, *num.var*

chan.num Pseudo device channel number

rec # Must be set to 0 when setting various port characteristics

mode Acceptable modes are 0 thru 7 inclusive, as described below

time-out Must always be set to -1 when setting various port characteristics

num.var Value to set based on *mode*

The following are the various modes acceptable for use with **CALL $DEVWRITE**:

MODE 0: Set input time-out value in tenths of a second (in this case 1 second). Example:

```
CALL $DEVWRITE,1,0,0,-1,10 ! set input time-out to 1 second
```

MODE 1 : Set a specific input length. Example:

```
CALL $DEVWRITE,1,0,1,-1,2 ! set input length to 2 bytes
```

MODE 2 : IRIS allows polling mode, but is not implemented for UniBasic.

```
CALL $DEVWRITE,1,0,2,-1,xx ! not supported
```

MODE 3 : Set port baud rate. Example:

```
CALL $DEVWRITE,1,0,3,-1,9600 ! set port to 9600 baud
```

MODE 4 : Set port word length. Example

```
CALL $DEVWRITE,1,0,4,-1,8 ! set word length of port to 8
```

MODE 5 : Set no parity, odd parity or even parity. A zero equals no parity, a one equals odd parity, and a 2 equals even parity. Example:

```
CALL $DEVWRITE,1,0,5,-1,2 ! set parity bit to even
```

MODE 6 : Set to one or two stop bits. A one will set it to one stop bit, anything else will set it to 2 stop bits. Example:

```
CALL $DEVWRITE,1,0,6,-1,1 ! set port to 1 stop bits
```

MODE 7 : Set modem control for a port. A non-zero sets modem control, i.e. clear CLOCAL and set HUPCL. A zero sets CLOCAL and clears HUPCL. Example:

```
CALL $DEVWRITE,1,0,7,-1,1 ! set modem control hupcl
```