UniData

UniData Commands Reference

Version 6.0
September, 2002
Part No. 000-9204

IBM Corporation
555 Bailey Avenue
San Jose, CA 95141

Documentation Team: Claire Gustafson

Table of Contents

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual contains an alphabetic listing of UniData commands and keywords and provides related syntax, options, and examples. This manual provides both ECL commands and system-level commands. All of the examples in this manual use the UniData demo account and its database files.

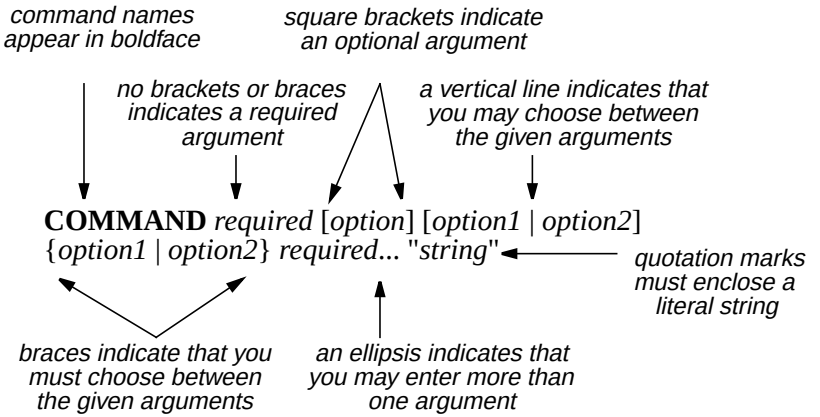
UniData provides the **Environment Control Language (ECL)**, a proprietary command language to handle database management functions. ECL commands execute from the UniData colon prompt (:).

ECL commands and keywords install when you install UniData. They are stored in the UniData Vocabulary (VOC) file. In this manual, these commands appear in uppercase. If you enter commands in lowercase, you invoke the UniData parser, regardless of the ECLTYPE setting.

UniData also provides system-level commands, which you execute from the shell prompt. System-level commands are stored in the *udtbin* directory. In general, these commands must be entered in lowercase. You can execute some system-level commands from the UniData colon prompt by entering the ! (bang) command first (e.g., :!**systest**).

Elements of Syntax Statements

This reference manual uses a common method for stating syntax for UniData commands. The syntax statement includes the command name, required arguments, and options that you can use with the command. Italics represents a variable that you can replace with any valid option. The following figure illustrates the elements of a syntax statement:


command names appear in boldface
square brackets indicate an optional argument
no brackets or braces indicates a required argument
a vertical line indicates that you may choose between the given arguments
COMMAND required [option] [option1 | option2]
{option1 | option2} required... "string"
quotation marks must enclose a literal string
braces indicate that you must choose between the given arguments
an ellipsis indicates that you may enter more than one argument

!

Syntax

! *system_command*

Description

The ECL **!** (bang) command gives a UniData process access to the operating system. With this access, you can execute operating system and UniData system-level commands.

Example

In the following example, the ! command executes the “pwd” UNIX command and the “showud” UniData system-level command:

```
:!pwd
/home/claireg
:!showud
UID     PID    TIME COMMAND
root   18126    0:00 /disk1/ud60/bin/aimglog 0 23192
root   18127    0:00 /disk1/ud60/bin/aimglog 1 23192
root   18121    0:00 /disk1/ud60/bin/bimglog 2 23192
root   18122    0:00 /disk1/ud60/bin/bimglog 3 23192
root   18114    0:04 /disk1/ud60/bin/cleanupd -m 10 -t 20
root   18123    0:53 /disk1/ud60/bin/cm 23192
root   18110    0:00 /disk1/ud60/bin/sbcs -r
root   18119    0:00 /disk1/ud60/bin/sm 60 6354
root   18103    0:02 /disk1/ud60/bin/smm -t 60
root   18145    0:00 /disk1/unishared/unirpc/unirpcd
```

ACCT_RESTORE

Syntax

ACCT_RESTORE [-D] [-E]
[-F *outputfile*]
[-H [DYNAMIC0 | DYNAMIC1]
[-O] [-S]
[-VREAL7] [-Z] [-U [0-9]] [-M [0-3]] [-X *char_list*] [-B [1 | 2 | 4 | 8]][-K *n*] [-L]
[-A *outputfile*] [-C *outputfile*] [-I *I_list*]
[-[X]R {ALL | *filelist*}] [-[X] Y *filelist*] [-YX *filelist*]
[*acct_name*]

Description

The system-level **ACCT_RESTORE** command restores Pick® R83-compatible accounts that were saved to tape in UniData format using the Pick® commands ACCT-SAVE and FILE-SAVE. The account must be compatible with Pick® R83 (it can contain no records larger than 32K and a minimum block size of 512). When you are restoring multiple accounts, UniData prompts for owner and group for each.

Tip: Use backward compatibility options with your save from the Pick® system, except with MCD Rev 7. When saving from Reality 7.0, use the -VREAL7 flag.

ACCT_RESTORE restores accounts, with their original names, to the current directory. If UniData cannot read the account name from tape, it uses *acct_name*. If no account of the same name exists in the current directory, UniData executes the newacct command to create one.

UniData loads Pick® DC-type files as UniData directory files with their dictionaries intact.

The executable for this command is located in your *udtbin* directory.

See "Preparing for Restoration" following the Parameters table for a recommended procedure for restoring files efficiently.

Parameters

You can enter ACCT_ RESTORE parameters in lowercase or uppercase. Some Pick® systems allow a hash type in the separation field in a file pointer.

Parameter	Description
-D	Overwrites the data portion of files with data files from the tape, but does not create new ones. The account must already exist, and all dictionary files must have been previously converted. Restores only hashed data files, not Pick® DC-type files (DC-type corresponds to UniData DIR-type).
-E	Clears each file on disk before restoring it from tape.
-F *outputfile*	Restores dictionary files using the list of files in *outputfile*. To restore data *and* dictionary files, use the -R option. Provide *filelist*, a list of files to be restored.
-H[DYNAMIC0 | DYNAMIC1]	Converts all restored files to dynamic with: DYNAMIC0 – hash type 0 DYNAMIC1 – hash type 1 (default)
-O	Overwrites all data in the account, including that in dictionary and DIR-type files, from tape. The files must already exist in the current directory. Execute ACCT_RESTORE -C to create the files on disk before executing ACCT_RESTORE -O to populate them.
-S	Truncates file names to 12 characters in length.
-VREAL7	Enables compatibility with REALITY 7.0, which allows for registration of items larger than 32K.

ACCT_RESTORE Parameters

Parameter	Description
-Z	Skips zero-length blocks on multireel tapes, floppy diskettes, or tape volume. When UniData encounters one or more zero-length blocks, it pauses at the end-of-file mark and prompts for user response before continuing. You must respond with one of the following: E — Terminate. F — Advance to EOR (end-of-reel). Use only when you are sure you are at the end or the tape or disk image. C — Go to the next file on the tape. Use when several files are saved on the tape and you want to load them all.
-U [0-9]	Indicates a tape unit to read from. The tape unit must be described in the tapeinfo file in *udthome*/sys. Default is 0. Use the ECL SETTAPE command first to set tape unit attributes.
-M [0-3]	Converts data based on one of the following options: 0 — Default. No conversion. Data is assumed to be ASCII. 1 — EBCDIC conversion. 2 — Invert high bit. 3 — Swap bytes.
-X [*char_list*]	*char_list* indicates characters to be considered invalid for file names account names record IDs in DIR-type files While restoring, UniData converts these characters to underscore (_). If the resulting name conflicts with an existing account name, UniData adds a character to the end of the name to make it unique. For example: A&B becomes A_B. If A_B is used by another file, the name becomes A_Ba. Default invalid characters are the following: space * ? / & '. You cannot specify nonprinting characters as invalid. Do not separate characters in *char_list* with spaces or commas.

ACCT_RESTORE Parameters (continued)

Parameter	Description
-B [1 | 2 | 4 | 8]	Each option corresponds to a block size (in bytes) of the data on the tape. 0.5 — 512 (default) 1 —1024 2 — 2048 4 — 4096 8 —8192
-K *n*	Defines the size of the internal memory buffer (in kilobytes). Default size is 8000 kilobytes. System restoration performs best when buffer size is large. Change the size to match the capacity of your operating system.
-L	Restores all files as type LF or LD.
-A *filename*	Creates *filename*, an ASCII text file, in the current directory, containing statistics about each file on the tape. -A does not restore files. (See "Preparing for Restoration" following this table).
-C *filename*	Reads the file created by a previous execution of ACCT_RESTORE with the -A *filename* option. Creates, in the current directory, the files listed in *filename*, but does not restore data.
-I *I_list*	Recovers the operation after an interruption. UniData prompts for names of files already loaded. See "Resuming after an Interruption" after this table.
X	Reverses the effect of -R or -Y. Syntax and effect is: -[X]R – Files in *filelist* are **not** restored. -[X]Y – Files in *filelist* remain static.

ACCT_RESTORE Parameters (continued)

Parameter	Description
-R *filelist* | ALL	Restores both data and dictionary portions of files listed in *filelist*. You create *filelist*, an ASCII file containing a single-line entry for each file to be ignored. Syntax for each entry is the following: [*filename*] [,*acct_name*] Include *filename* only to load all files from a single account. Include *acct_name* only to load all files from a specific account.
-Y *filelist*	Converts the files in filelist to dynamic. Used in conjunction with the HDYNAMIC0 or -HDYNAMIC1 option. You create filelist, an ASCII file containing a single-line entry for each file to be ignored. Syntax for each entry is the following: [filename] [,acct_name]
acct_name	New name for the restored account to be used if UniData cannot obtain a name from the account on tape.

ACCT_RESTORE Parameters (continued)

Preparing for Restoration

IBM recommends that you follow this procedure to make the restoration more efficient. Use the -A option in conjunction with -C and -O to determine file status before files are loaded. This decreases load time, because UniData then does not have to resize files during restoration.

1. Execute "ACCT_RESTORE -A *filename*" to generate a file containing statistics about the files on tape. Use these statistics to evaluate the suitability of the projected modulo, file type, and file separation.

   filename is stored in the current directory. For each file, UniData lists the following on a single line separated by commas:

   - The position of the file on the tape.
   - The type of UniData file.
   - The name of the UniData file.
   - File separation.
   - New or recommended modulo — Informix recommends a modulo based on the number of records and the size of the file. This recommended modulo is never smaller than the original modulo.
   - The original modulo of the file on tape.
   - The proposed key length for the UniData file.
   - The total record length for the file.
   - The number of records in the UniData file.

2. Use an ASCII text editor to modify the file generated in Step 1 as desired. For example, you might eliminate files from the list that you do not want UniData to restore.
3. Execute "ACCT_RESTORE -C *filename*" to create new UniData files in the destination directory. Remember, *filename* must be the name of the file created in Step 1. You can add options as desired.
4. Execute "ACCT_RESTORE -O *filename*" to load the data and dictionary records into the files created in Step 3. You can add options as desired.

Resuming after an Interruption

Follow this procedure if you are interrupted when restoring files with the -C, -R, or -O options.

1. Check the last 10 lines of the dispmsg file in the current directory, and record the message about the last reel. The following is an example of 10 lines from a dispmsg file:

```
D[2].flag=0
D[3].flag=0
D[1].count=1
D[2].count=194
D[3].count=195
D[2].rel.relname=DIFF
D[3].rel.relname=DIFF
D[2].sname=DIFF
D[3].sname=DIFF
IC3~01220~0011ABC~3
```

2. To ensure that no files are skipped, enter the last 10 statements into *I_list* file.
3. Remount the interrupted reel.
4. Execute "ACCT_RESTORE -I".

   UniData reads *I_list*, displays the name of each file loaded, and prompts you to skip or reload it (overwriting the existing copy).

ACCT_RESTORE Messages

UniData may display the following messages during the restore.

Message	Description
Create file modulo separator [---*newfile*]	UniData is loading the file using the modulo and separator found in the tape. If the file name contains invalid characters or if the file name is too long, UniData changes it to "newfile".
DUMP_MD	UniData is reading an MD file.
DICT	UniData is reading a dictionary file.
DATA	UniData is reading a single-level hashed data file.
DIR	UniData is reading a single-level sequential file.
LF	UniData is reading a multi-level hashed data file.
LD	UniData is reading a multi-level sequential file.

ACCT_RESTORE Messages

Message	Description
Loading (*filename*)...	UniData is loading the data into existing files rather than creating files. This is the default when you run ACCT_RESTORE with the -D, -F, or -O option.
Replace to multi-level success.	A single-level file changed to a multi-level file.
Replace to multi-level failure.	UniData failed to change a single-level file into a multi-level file.
Resize (*filename*) to new modulo --- (*modulo*)	The file called *filename* has an inadequate modulo; UniData resized the file to a more efficient modulo (*modulo*).
Create file failure.	UniData failed to create the file.
Open file failure.	UniData failed to open the file.

ACCT_RESTORE Messages (continued)

Files Created by ACCT_RESTORE

UniData creates the following files in the restored account by default.

File	Description
analyze_list	Lists number of records, total key length, and total record length for each file.
DUMP_MD	Hashed file. Contains the account's original Pick® MD file.
pgm_list	Text file. Record of program names altered by the load. UniData conversion tools use this file.
dispmsg	Text file. A compilation of all messages generated during the restore.
resize_list	Text file. Record of file names that may be resized at a later time.

ACCT_RESTORE-Related Files

acctrestore

Syntax

acctrestore [*n*]

Description

The system-level **acctrestore** command restores a UniData account from a tape backup. The account must have been saved with the ACCT.SAVE command. acctrestore operates on a single tape volume. *n* represents the tape unit number in the *udthome* / sys / *tapeinfo* file. Use the SETTAPE command to define the tape unit.

Note: acctrestore is supported on UniData for UNIX only.

You must have permission to read from and write to the tape device to use this command. For more information about managing UniData accounts, see *Administering UniData.*

This command does not function if the Recoverable File System is running. If you used the ACCT.SAVE command to save an account that contains recoverable files, acctrestore does not restore those files as recoverable. To convert them to recoverable, run the udfile command against them. See *Administering the Recoverable File System* for more information about udfile and recoverable files.

Warning: To avoid file corruption, do not use this command while UniData is running.

Note: acctrestore uses the UNIX cpio utility: cpio -iBvd < %s", raw

Example

In the following example, UniData restores a file and its subdirectories from a backup tape:

```
# $UDTBIN/acctrestore
Status: Tape unit 0 blocksize = 1024.
cpio -iBvd < /users/claireg/tape
.
BP
BP_SOURCE
BP_SOURCE/GPA1
BP_SOURCE/PHONE_FMT
BP_SOURCE/PSTLCODE_FMT
BP_SOURCE/UP_NAME
BP_SOURCE/_GPA1
BP_SOURCE/_PHONE_FMT
BP_SOURCE/_PSTLCODE_FMT
BP_SOURCE/_UP_NAME
CATEGORIES
...
650 blocks
#
```

Related Command

ACCT.SAVE

ACCT.SAVE

Syntax

ACCT.SAVE

Synonym

ACCT-SAVE

Description

The ECL **ACCT.SAVE** command saves the current UNIX directory and all of its subdirectories to the device defined as tape unit 0 in *udthome* / sys / *tapeinfo*.

Note the following before using ACCT.SAVE:

- Before you use this command, use the SETTAPE command to define the tape unit.
- This command does not function if the Recoverable File System is running.
- You must have permissions to write to the tape device to use this command.
- ACCT.SAVE uses the UNIX cpio utility: find . -print | cpio -oBv > %s",raw

Note: ACCT.SAVE is only supported on UniData for UNIX.

Example

In the following example, UniData saves the current UNIX directory and its subdirectories to tape unit 0. Notice how UniData displays a list of all subdirectories in the account. You must already have defined a device as tape unit 0 with the SETTAPE command.

```
:ACCT.SAVE
find . -print | cpio -oBv > /users/claireg/tape
.
BP
BP_SOURCE
BP_SOURCE/GPA1
BP_SOURCE/PHONE_FMT
BP_SOURCE/PSTLCODE_FMT
BP_SOURCE/UP_NAME
BP_SOURCE/_GPA1
...
650 blocks
```

Related Command

acctrestore

AE

Syntax

AE [*filename*] [*record*]

Description

The ECL **AE** command invokes the UniData Alternate Editor. You can use this line editor to edit UniData hashed files and UniBasic source programs. If you do not indicate the *filename* or *record*, AE prompts for them. See *Developing UniBasic Applications* for a brief introduction to the editor.

If you have an active select list, you can execute AE from the select list prompt rather than entering *record*, and UniData opens each record successively: when you close one record, the next one opens. To exit the select list without saving changes, enter QK at the command prompt in AE. See *Using UniData* for instructions on creating and using select lists.

UniData displays a warning message if a trigger prevents record update or deletion. See *Developing UniBasic Applications* or CREATE.TRIGGER in this manual for more information on UniData triggers.

Regarding other editors:

- The ECL ED command invokes the standard operating system editor supported by UniData. See ED in this manual for more information.
- UniData also supplies UniEntry for modifying UniData records.
- On UniData for UNIX, the ECL VI command invokes vi, the UNIX System V visual editor, from within UniData.
- You can edit UniData hashed files and DIR-type files with any ASCII text editor. Refer to your operating system documentation for more information on supported editors. Be aware, though, of any changes or conversions the editor might make to files it opens.

Tip: To display the ASCII code for control characters (including UniData delimiters and the null value) in AE, press SHIFT+6.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of the file to edit or create.
record	ID of the record to edit or create.

AE Parameters

Common AE Commands

The following table lists commonly used AE editor commands.

Command	Description
C / *old.string* / *new.string*	Changes the current character string to a new character string on the current line.
P	Displays one page of the record.
HELP	Displays online help for AE. You can also enter HELP followed by a topic or AE command. You can also access the UniData help system using the XEQ command. For example, “XEQ HELP SELECT”.
I	Enters insert mode to enter text.
EX or Q	Exits the record without saving changes made this editing session.
FI	Files the UniBasic program record, saving changes.
FIB	Files the UniBasic program record and compile it.
FIBR	Files the UniBasic program record, compile it, and run it. If the compile is unsuccessful, the last successfully compiled version is executed.

Common AE Commands

Command	Description
FIR	Files the UniBasic program record and run the compiled version. Be aware that the compiled version may differ from the one you are editing.
FIBCFN	The N option of the FI command equates to the ECL NEWPCODE command. FIBCFN compiles a program and catalogs it (locally) with NEWPCODE. You need to use F (force) in conjunction with the N option. Refer to the online help for the AE editor for more information.
LN*n*	Lists the number of lines indicated with no line numbers.
n	Goes to line number *n*.
T	Goes to the top of the record.
SPOOLHELP	Prints brief help.
SPOOLHELP -FULL	Prints extensive help.
<Return>	Returns to command mode.

Common AE Commands (continued)

Example

In the following example, the AE command opens record 9999 of the CLIENTS file for editing:

```
:AE CLIENTS 9999
Top of "9999" in "CLIENTS", 10 lines, 95 characters.
*--:
```

Related Commands

ED, VI

ANALYZE.FILE

Syntax

ANALYZE.FILE *filename*

Synonym

ANALYZE-FILE

Description

The ECL **ANALYZE.FILE** command displays information about a dynamic file. The output includes information about the file's size, split / merge type, and hash type. The output also lists all the groups in the file along with loading information for each. The output of this command differs depending on the split / merge type of the file being analyzed.

Examples

The following example displays file and group information about the dynamic file INVENTORY in the demo database:

```
:ANALYZE.FILE INVENTORY
Dynamic File name                       = INVENTORY
Number of groups in file (modulo)       = 19
Minimum groups of file                  = 19
Hash type = 1, blocksize = 1024
Split load = 60, Merge load = 40
Split/Merge type = KEYONLY

   Group      Keys       Key Loads        Percent
================================================
    0          6              84               8
    1          3              42               4
    2          5              70               6
    3         11             154              15
...
   15          8             112              10
   16         11             154              15
   17          8             112              10
   18          2              28               2
================================================
 Average       9             128              12
File has 175 records.
:
```

Notice that the INVENTORY file is a KEYONLY file. For purposes of splitting and merging, the loading factor for each group is computed (and shown) based on keys only.

The next example shows ANALYZE.FILE output if the split/merge type of the INVENTORY file is changed to KEYDATA.

```
:ANALYZE.FILE INVENTORY
Dynamic File name                       = INVENTORY
Number of groups in file (modulo)       = 19
Minimum groups of file                  = 19
Hash type = 0, blocksize = 1024
Split load = 95, Merge load = 40
Split/Merge type = KEYDATA

1   Group     Keys      Key Loads       Percent        Key+Data        Percent
1 ================================================================================
1     0         9            126            12             836             81
1     1         8            112            10             692             67
1     2         9            126            12             808             78
1     3         7             98             9             598             58
1 ...
1    15         9            126            12             812             79
1    16        10            140            13             839             81
1    17         9            126            12             769             75
1    18         7             98             9             651             63
1 ================================================================================
1  Average      9            128            12             783             76
1 File has 175 records.
1 :
```

Notice that the split / merge type is now KEYDATA. For purposes of splitting and merging, the load factor for each group is based on both keys and data. For KEYDATA files, ANALYZE.FILE reports load based on keys as well as load based on both keys and data.

auditor

Syntax

auditor

Description

The system-level **auditor** command detects certain types of error conditions that affect dynamic files.

When a dynamic file expands outside the file system in which it was created, the "part files" are placed in a file system selected from a "part table" (a list of locations where the original file can expand). The original dynamic file directory contains UNIX symbolic links to the physical location of the data and overflow "part files." In each file system in which dynamic files expand, UniData maintains a UNIX hidden file called .fil_prefix_tbl that relates part file names back to their original dynamic file and account.

The auditor command reports inconsistencies between the symbolic links and the hidden files that should be resolved. If inconsistencies aren't resolved, users may encounter unexpected results (for instance, part files from the same dynamic file may be created in different directory structures for no apparent reason, or commands may fail unexpectedly due to naming conflicts). This command also reports an error if a part file is not found in the correct location. Your current working directory must be a UniData account. The auditor command checks all the dynamic files that have pointers in the current account directory's VOC file.

Note: auditor is supported on UniData for UNIX only.

The auditor command does not check all possible error conditions that can affect a dynamic file. After you resolve any conditions reported by auditor, use the guide command to verify the integrity of your files.

Examples

The following example shows auditor output from a UniData account:

```
:!auditor
In current account, VOC entry SAMPLE_FILE3, is a pointer to
SAMPLE_FILE3.
There is a mismatch between the symbol link for 'dat001'
of SAMPLE_FILE3 and /tmp/partfiles/.fil_prefix_tbl.

In current account, VOC entry SAMPLE_FILE3, is a pointer to
SAMPLE_FILE3.
There is a mismatch between the symbol link for 'over001'
of SAMPLE_FILE3 and /tmp/partfiles/.fil_prefix_tbl.

:
```

The next example shows auditor output when no inconsistencies are found:

```
:!auditor

auditor finished, no error was detected.

:
```

Related Commands

fixtbl, mvpart

AVAIL

Syntax

AVAIL

Description

The ECL **AVAIL** command displays the number of blocks the operating system is using and the number of free blocks. AVAIL is a UniData implementation of the UNIX “df” command. Results vary depending on the operating system type and release. Refer to your host operating system documentation for a detailed explanation of the output from the df command.

Note: AVAIL is supported on UniData for UNIX only.

You can execute df with options from the UniData colon prompt (:) by preceding the command with the UniData (bang) command.

Example

In the following example, the AVAIL command is executed. It displays information on the number of blocks used by UNIX and the number of blocks free.

```
:AVAIL
/usr            (/dev/vg00/lvol3):    44364 blocks         33380 i-nodes
                                     332592 total blocks   43008 total i-
nodes
                                     254968 used  blocks    9628 used i-
nodes
                                         10 percent minfree

/users          (/dev/vg00/lvol4):    79134 blocks        222993 i-nodes
                                    1860880 total blocks  241664 total i-
nodes
                                    1595658 used  blocks   18671 used i-
nodes
                                         10 percent minfree

/tmp            (/dev/vg00/lvol5):    41930 blocks          5989 i-nodes
                                      63860 total blocks    6144 total i-
nodes
                                      15544 used  blocks     155 used i-
nodes
                                         10 percent minfree

/               (/dev/vg00/lvol1):    12152 blocks         19071 i-nodes
                                     166000 total blocks   22528 total i-
nodes
                                     137248 used  blocks    3457 used i-
nodes
                                         10 percent minfree
 :
```

BASIC

Syntax

BASIC *filename* [TO *filename*] *prog.name1* [*progname2*...] [*options*]

Description

The ECL **BASIC** command compiles UniBasic source code into interpretive code to be used with the UniBasic interpreter. UniData names the resulting object code record *_prog.name*, where *prog.name* is the name of the source code record.

Tip: You can create a select list, then execute BASIC to compile all programs in the select list. For example, to select and compile all UniBasic source files in the BP directory, enter SELECT BP WITH @ID UNLIKE "_..." *Then, enter* BASIC BP *from the select prompt.*

Note: The UniBasic compiler returns nonfatal warning messages. If you run batch jobs to compile groups of programs, you need to code those jobs to terminate only if the compiler returns error messages. Messages beginning with "Warning:" should not terminate processing.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	UniData DIR-type file containing the source code to be compiled.

BASIC Parameters

Parameter	Description
TO *filename*	UniData DIR-type file to receive the object code record, if different from the location of the source code record.
program	Source code to be compiled. You can compile more than one program by separating the names with a space.
options	See “BASIC Options” in this section.

BASIC Parameters (continued)

BASIC Options

The following table lists the BASIC command options.

Option	Description
-D	Creates a cross-reference table for use with the UniBasic debugger.
-G	Generates a program that you can run with profiling.
-L -LIST	Generates a list of the program.
-X -XREF	Generates a cross reference table of statement labels and variable names used in the program.
-Z*n*	Creates a symbol table for use with the UniBasic debugger. UniData doesn’t recompile the program or expand $INCLUDE statements. Use one of the following options: ■ Z1 – for programs compiled on a UniData release earlier than release 3.1 ■ Z2 – for programs compiled on UniData Release 3.1 or later.
-I	If you compile a program with the -I option, all reserved words in UniBasic are case insensitive.

***options* Parameters**

Examples

In the following example, the BASIC command compiles the program TEST, found in the BP file, and stores the resulting object code as _TEST.

```
:BASIC BP TEST -D

Compiling Unibasic: BP/TEST in mode 'u'.
compilation finished
```

In the next example, the SELECT command saves in select list 0 the names of all programs in the BP file with names (record IDs) beginning with T. Then, the BASIC command compiles the selected program.

```
:SELECT BP WITH @ID LIKE "T..."

1 record selected to list 0.

>BASIC BP

Compiling Unibasic: BP/TEST in mode 'u'.
compilation finished.
```

The following example saves the executable in a DIR-type file different from the one that contains the source code. In the first line, the program, test, which resides in BP, is compiled, and the executable placed in PROGRAMS. Then the program is executed from PROGRAMS. The program prints "Hello".

```
:BASIC BP TO PROGRAMS test

Compiling Unibasic: BP/test in mode 'u'.
compilation finished
:RUN PROGRAMS test
Hello
```

BASICTYPE

Syntax

BASICTYPE ["U | P | R | M"] [*filename program*]

Description

The ECL **BASICTYPE** command selects the parser that UniData uses to interpret UniBasic commands for the duration of this session or until you execute BASICTYPE to select a different parser. This command is useful when compiling programs that need to be backwardly compatible.

If you do not include any parameters with this command, UniData returns the current BASICTYPE. If you do not select a parser option, but you do indicate a *filename* and *program*, UniData returns the BASICTYPE in which the program was compiled.

This ECL command performs the same function as the UniBasic $BASICTYPE command. For more information on the commands affected by BASICTYPE, refer to the individual commands in *Developing UniBasic Applications*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
"U"	UniData interprets commands and keywords consistent with the UniData parser. Must be enclosed in quotation marks.
"P"	UniData interprets commands and keywords consistent with the Pick® BASIC parser. Must be enclosed in quotation marks.
"R"	UniData interprets commands and keywords consistent with the Advanced Revelation® BASIC parser. Must be enclosed in quotation marks.

BASICTYPE Parameters

Parameter	Description
"M"	UniData interprets commands and keywords consistent with the McDonnell Douglas or Reality® BASIC parser. Must be enclosed in quotation marks.
filename	DIR-type file containing the program to be compiled. If you indicate a *filename* you must also name a *program*.
program	UniBasic program to be compiled.

BASICTYPE Parameters (continued)

Examples

In the following example, the BASICTYPE command, executed without any parameters, returns the current BASICTYPE (in this case standard UniData.

```
:BASICTYPE
BASICTYPE u
:
```

In the next example, the BASICTYPE command sets the BASICTYPE to P, for Pick® BASIC.

```
:BASICTYPE "P"
:
```

In the next example, UniData returns the BASICTYPE of the demo program PHONE_FMT in the directory file BP_SOURCE.

```
:BASICTYPE BP_SOURCE PHONE_FMT
Basic program 'BP_SOURCE/_PHONE_FMT'was compiled with mode 'u'.
:
```

Warning: Take care not to mix BASICTYPES in an application. For instance, do not call a P-type subroutine from a U-type program. Because the parsers interpret commands and keywords differently, using different BASICTYPEs may produce unexpected results.

Related Command

ECLTYPE

BLIST

Syntax

BLIST *filename record_ID* [([*lineM-lineN* [*option*]]]

Description

The ECL **BLIST** command lists and formats a UniBasic source code program for display to the terminal screen. When you issue the command without options, UniData displays the program. For more information about UniBasic, see *Developing UniBasic Applications*.

In UniBasic, comment lines begin with *, !, or REM. The BLIST command converts comments that begin with an exclamation point (!) to a row of asterisks (*). Two exclamation points (!!) at the beginning of a line produces a page eject. UniData does not convert comment lines that begin with * or REM.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	DIR-type file where the source code is stored.
record_ID	Designates a record that contains the UniBasic source code program.
(*lineM-lineN*	Indicates a range of line numbers. You must enter the single parenthesis and hyphen.
option	Formatting operations to be performed or output conditions to be met. Only one option is allowed on the command line.

BLIST Parameters

BLIST Options

The following table lists the BLIST options.

Option	Description
A	Indents all lines beginning with an asterisk (*) according to the original starting position.
B	The number of leading spaces for the first level of indentation. UniData indents subsequent levels in multiples of this number. When you use the B option, UniData prompts for a value. Valid values are 1 through 5. The default setting is 5. If you enter any other value, Unidata ignores it and uses the default value.
C	Places all comment lines at the left margin, regardless of their original staring position.
D	Output is double-spaced.
E	Expands all $INSERT code segments into the listing.
F	Prints the file name and program name on the first line of the listing.
K	Suppresses the printing of a line of asterisks (*) when the system encounters an exclamation mark (!) at the beginning of the line.
L	Prints a period (.) at each level of indentation.
M	Prints line numbers at the left margin.
N	The listing scrolls continuously, instead of stopping at each page.
P	Directs the listing to the printer that is assigned to your port or to a printer you assign through SETPTR command options. The default is to send the list to the display terminal.
X	Always used with the E option. Prints a level number for each $INSERT statement.

BLIST Options

Examples

In the following example, a segment of the PHONE_FMT demo program has been formatted so that lines 11 through 20 start at the left margin:

```
011:  RET_DATA = ""
012:  Counter = 1
013:  LOOP WHILE Counter <= TotalValues
014:  BEGIN CASE
015:  CASE COUNTRY = 'USA'
016:  IF LEN(PHONE_NUM<1,Counter>) = 7 THEN
017:  RET_DATA<1,-1> = FMT(PHONE_NUM<1,Counter>,"14R     ###-####")
018:  END ELSE
...
```

The next example shows how UniData reformats the program by double spacing the listing:

```
:BLIST BP_SOURSE PHONE_FMT (11-20 D

PAGE 1       Uni/Basic       Wed Jul 31 11:37:18 2000
PHONE_FMT

011:  RET_DATA = ""

012:  Counter = 1

013:  LOOP WHILE Counter <= TotalValues

014:  BEGIN CASE

015:  CASE COUNTRY = 'USA'
016:  IF LEN(PHONE_NUM<1,Counter>) = 7 THEN

017:  RET_DATA<1,-1> = FMT(PHONE_NUM<1,Counter>,"14R     ###-####")

018:  END ELSE
...
```

BLOCK.PRINT

Syntax

BLOCK.PRINT *expr*

Synonym

BLOCK-PRINT

Description

The ECL **BLOCK.PRINT** command prints the value of *expr* to the printer.UniData prints *expr* in large uppercase letters and cannot print more than ten characters on a single line. To depict an initial capital letter, UniData prints the initial capital letter in a slightly larger point size.

Note: In ECLTYPE U, this command prints to the printer. In ECLTYPE P, it prints to the terminal screen.

Example

In the following example, using BASICTYPE P, the BLOCK.PRINT command prints to the terminal:

```
:BLOCK.PRINT HELLO
#     # ####### #       #       #######
#     # #       #       #       #     #
#     # #       #       #       #     #
####### #####   #       #       #     #
#     # #       #       #       #     #
#     # #       #       #       #     #
#     # ####### ####### ####### #######

:
```

RELATED COMMAND

BLOCK.TERM

BLOCK.TERM

Syntax

BLOCK.TERM *expr*

Synonym

BLOCK-TERM

Description

The ECL **BLOCK.TERM** command displays the value of *expr* to the standard output device, usually the display terminal. UniData displays *expr* in large uppercase letters and cannot display more than 10 characters on a single line. To depict an initial capital letter, UniData displays the initial capital letter in a slightly larger point size.

Tip: If expr exceeds 255 characters, you can use the UniData continuation character (\ ) to enter the excess characters over 255 on the same line. For example, 1. Note erros...2. Correct 3. Balance ... \ 10 Record time.

Example

In the following example, UniData displays an expression with the BLOCK.TERM command:

```
1   :BLOCK.TERM HELLO
1   #     # ####### #       #       #######
1   #     # #       #       #       #     #
1   #     # #       #       #       #     #
1   ####### #####   #       #       #     #
1   #     # #       #       #       #     #
1   #     # #       #       #       #     #
1   #     # ####### ####### ####### #######
1
```

:

Related Command

BLOCK.PRINT

BUILD.INDEX

Syntax

BUILD.INDEX *filename* {*attribute* [*attribute*...] | ALL }

Synonym

BUILD-INDEX

Description

The ECL **BUILD.INDEX** command activates alternate key indexes and populates them with keys. If keys are already present in the index, UniData overwrites them. While the index is being built, users can access the related data file, but cannot update it.

You must create the alternate key index file with CREATE.INDEX before you can execute the BUILD.INDEX command. You must also execute BUILD.INDEX against the index before UniData can access it. This is true even if the data file is empty.

You cannot build an alternate key index when index updating has been disabled by the DISABLE.INDEX command.

When BUILD.INDEX completes successfully, UniData sets @SYSTEM.RETURN.CODE equal to the number of indexes built. A value of -1 in @SYSTEM.RETURN.CODE indicates an unsuccessful build.

If you specified NO.DUPS when you executed CREATE.INDEX against a nonrecoverable file, BUILD.INDEX does not populate the index if it encounters duplicate alternate key values. If you EXECUTE or PERFORM BUILD.INDEX from a UniBasic program and the command fails because the data file contains duplicate alternate key values, the UniBasic program aborts.

Using Indexes Created in an Earlier Release

Keep the following in mind when upgrading or using an index that was created with an earlier release of UniData:

- On UniData for UNIX, when upgrading from a release earlier than 3.3, you need to rebuild indexes. UniData added a time stamp feature at Release 3.3.
- Indexes created at Release 4.1 of UniData for UNIX or Release 3.6 of UniData for Windows NT are not backwardly compatible. Beginning with these releases, indexes were no longer compressed.

Tip: Use the UniBasic INDICES function to find out when an index was created.

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
filename	The name of the UniData file that is indexed.
attribute	The name of the attribute used as the alternate key. You can build more than one index at a time.
ALL	Builds all indexes associated with *filename*.

BUILD.INDEX Parameters

Tip: Use BUILD.INDEX ALL to build all of the indexes associated with a file at the same time. You cannot execute multiple BUILD.INDEX commands for individual attributes simultaneously.

Example

The following example creates an index on the COMPANY attribute of the CLIENTS demo file. Then the BUILD.INDEX command activates and loads keys into the index:

```
:CREATE.INDEX CLIENTS COMPANY
Alternate key length (default 20): 45
“COMPANY” created

:BUILD.INDEX CLIENTS COMPANY
Quick Build strategy is applied.
One “*” represents 1000 records

Building “COMPANY” ...

130 record(s) processed.
```

Related Commands

CREATE.INDEX, DELETE.INDEX, DISABLE.INDEX, ENABLE.INDEX, LIST.INDEX, UPDATE.INDEX

BYE

Syntax

BYE

Synonyms

LO, QUIT

Description

The ECL **BYE** command exits the UniData environment and returns the cursor to the host operating system prompt.

Example

In the following example, the user executes the BYE command to exit the UniData environment.

```
:BYE
%
```

Related Command

udt

CATALOG

Syntax

CATALOG *filename* [*catalog*] *program* [LOCAL | DIRECT] [FORCE]] [NEWVERSION | newversion]

Description

The ECL **CATALOG** command copies the compiled object code of a UniBasic program into a catalog space. By default, UniData catalogs a program globally and copies it into a subdirectory of *udthome*/sys/CTLG on UniData for UNIX, or *udthome*\sys\CTLG on UniData for Windows Platforms, the system catalog.

Multiple users can run globally cataloged programs simultaneously — UniData brings one copy of the program into shared memory.

You can use the CATALOG command in conjunction with a select list of UniBasic programs.

For more information about UniBasic programming, see *Developing UniBasic Applications*. For more information about shared memory and newversion, see *Administering UniData*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
catalog	The name of a global catalog where UniData copies the object code, if different from the default CTLG directory.
filename	The UniData DIR-type file that contains the program to be cataloged.
program	The UniBasic program that contains object code to be cataloged.
DIRECT	Catalogs the program locally without copying it to the local or system CTLG directory. Instead, UniData creates an entry in the VOC file that is a pointer to the directory where the program resides.
FORCE	Overwrites programs in a catalog that have the same name as *filename*. You can use the FORCE option in conjunction with the DIRECT or LOCAL option.
LOCAL	Catalogs the program locally and places a copy of it in a subdirectory of the local CTLG catalog (in the account where the user is running the program). UniData creates a VOC pointer to the subdirectory. **Note**: UniData creates the CTLG and the subdirectory, if they do not already exist.
NEWVERSION | newversion	Replaces the current version of a globally cataloged program in shared memory with the newly cataloged version. The UniData background process sbcs controls this activity. (See the next section, “Modifying Globally Cataloged Programs.”) **Root**: You can use this keyword only if you are logged in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

CATALOG Parameters

Modifying Globally Cataloged Programs

In order for multiple users to use a single program at the same time, UniData retrieves a copy of a globally cataloged program into shared memory. When you modify a program and recatalog it, any user who began using the program (the copy in shared memory) before you cataloged the new version continues to use the copy in shared memory.

Users who run the program after you recatalog it use the new version. When you return to the ECL prompt, you have access to the new version.

To force users to attach to the new version, use the ECL NEWPCODE command.

Note: Simply copying the executable to the global catalog space does not update the version of the program in shared memory.

Calling Programs

You can call a globally cataloged program from the ECL prompt or from any UniBasic CALL statement in any account. Locally and directly cataloged programs must be cataloged in each account where they are used.

Pointing to Directly Cataloged Programs

A program that is cataloged using the DIRECT option does not have to be recataloged when you recompile the program. This is because UniData creates a pointer in the VOC file that points to the program itself. If you change the location of the program, however, you must recatalog it to update the VOC pointer.

The following example shows a VOC file pointer for the PSTLCODE_FMT program in the demo database (PSTLCODE_FMT is called by the virtual attribute ZIP in both the CLIENTS and ORDERS demo files.) The CT command lists the record. Notice that the program resides in the BP_SOURCE directory.

```
:CATALOG BP_SOURCE PSTLCODE_FMT DIRECT
PSTLCODE_FMT has been cataloged, do you want to overwrite(Y/N)? Y
:CT VOC PSTLCODE_FMT
VOC:

PSTLCODE_FMT:
C
BP_SOURCE/_PSTLCODE_FMT
:LIST CTLG

No records listed.
```

Tip: To delete a VOC pointer for a cataloged program, use the ECL DELETE or AE commands, or use UniEntry or the .D command. For more information on UniEntry and the .D command, see Using UniData.

Examples

The following example lists the contents of the CTLG file in the demo database. Notice that it is empty. If any of the demo database programs had been locally or directly cataloged, a copy of the object code would reside in CTLG.

```
:LIST CTLG
No record listed.
```

In the next example, UniData catalogs the compiled object code of the PSTLCODE_FMT program locally. Afterward, notice the following:

- The local CTLG directory shows an entry for PSTLCODE_FMT.

- A VOC pointer exists that shows a path to a copy of the program and shows where the program actually resides (BP_SOURCE).

```
:CATALOG BP_SOURCE PSTLCODE_FMT LOCAL
:LIST CTLG
LIST CTLG 11:08:04 Jul 28 2001  1
CTLG......

PSTLCODE_F
MT
1 record listed

:CT VOC PSTLCODE_FMT
voc:

PSTLCODE_FMT:
C
/disk1/ud52/demo/CTLG/PSTLCODE_FMT
BP_SOURCE PSTLCODE_FMT
```

Note: On UniData for Windows Platoforms, the path in the previous example would be \disk1\demo\CTLG\PSTLCODE_FMT.

The next example directly catalogs the PSTLCODE_FMT program. Notice that the path to the program has changed from the previous example. DIRECT cataloging creates a VOC pointer to the object code, but does not place a copy of it in either CTLG directory.

```
:CATALOG BP_SOURCE PSTLCODE_FMT DIRECT
:CT VOC PSTLCODE_FMT
VOC:

PSTLCODE_FMT:
C
BP_SOURCE/_PSTLCODE_FMT
```

Tip: To remove a copy of a program from the local or system CTLG directory, use the ECL DELETE or DELETE.CATALOG commands.

Related Commands

DELETE.CATALOG, NEWPCODE, newversion

CENTURY.PIVOT

Syntax

CENTURY.PIVOT(*4-digit year* | *nn*)

Description

Prior to UniData 5.2, any 2-digit year entered from 1 through 29 defaulted to the next century. For example, UniData interpreted 12/31/29 as December 31, 2029. 1930 was the century pivot date.

You can set your own century pivot date. The century pivot date only applies to the ICONV function when using the D2 format, not D3 or D4.

The CENTURY.PIVOT ECL command overrides the systemwide century pivot date defined in the udtconfig file.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
4-digit year	The 4-digit year defining the century pivot date.
nn	The century pivot date code, indicating that the next *nn* years are in the next century.

CENTURY.PIVOT Parameters

You can change this value in one of the following ways:

- Enter a 4-digit year. UniData interprets the first 2 digits as the century, and the last 2 digits as the year. The last 2 digits of the year you enter, though 99, are considered to be in the century you specify. 0, through the year you entered -1, are considered to be in the next century. For example, if the century pivot date is 1950, years 50 through 99 are in the 1900's, and years 0 through 49 are in the 2000's. If the century pivot date is 2000, 0 through 99 are in the 2000's.
- Enter a code in the form of *nn*, indicating that the next *nn* years are in the next century. UniData calculates the century pivot date as:

  current_year - (100 - *nn*)

  For example, if the current year is 2000 and the century pivot code is 50, the century pivot date is 1950 (2000 - (100 - 50)).

If you enter CENTURY.PIVOT with no options, UniData returns the current setting for the century pivot date.

CHECKOVER

Syntax

CHECKOVER

checkover

Description

The ECL **CHECKOVER** command and the system-level **checkover** command list files in the current account that are in level 2 overflow. CHECKOVER also reports the number of groups that have overflowed.

Static hashed files are divided into a specific number of groups (the file's modulo). When you first write data to the file, UniData stores IDs and data in the same file block. When the block becomes full of data, a level 1 overflow occurs and data is written to a second block. If enough records are written to the same block, the primary keys also overflow — this is level 2 overflow.

Tip: Your system administrator should run this command for each UniData account and periodically resize files for optimal system performance.

Example

In the following example, UniData indicates that the CTLGTB file has overflowed. The last line of the display shows the file modulo (mod=17) and the number of level 2 overflowed blocks (overflow mod=111), including all level 2 overflowed headers.

```
:checkover
Current directory is '/home/claireg'
Overflowed files are listed in the file U_OVERFLOWED, which is
located in your current directory. Please resize files listed,
then rerun checkover again until no more overflowed files are
identified.
CTLGTB overflowed, mod=17, overflow mod=111
```

CLEAR.ACCOUNT

Syntax

CLEAR.ACCOUNT

Synonym

CLEAR-ACCOUNT

Description

The ECL CLEAR.ACCOUNT command deletes all records from the UniData system _PH_ and _HOLD_ directories.

Note: The _PH_ directory stores COMO files and phantom log records. The _HOLD_ directory stores print hold files

Example

In the following example, the CLEAR.ACCOUNT command clears the _PH_ and _HOLD_ directories:

```
:CLEAR.ACCOUNT
Clear _PH_ directory(Y/N)? Y
Clear _HOLD_ directory(Y/N)? Y
:
```

CLEAR.FILE

Syntax

CLEAR.FILE [DATA] [DICT] *filename* [FORCE]

Synonym

CLEAR-FILE

Description

The ECL **CLEAR.FILE** command deletes all records from the data or dictionary sections of *filename*, or both the data and dictionary portions. If you do not stipulate DATA or DICT in the statement, UniData deletes only the data records. You can clear only files for which you have adequate permission. After execution of CLEAR.FILE, the empty file remains.

The data portions of multifile and multidir files are defined in the dictionary as *@data.filename*. UniData does not remove these pointers when you specify the DICT keyword to clear a multifile or multidir file. UniData removes all dictionary records except those beginning with the @ sign.

Without the FORCE option, *filename* cannot be a synonym.

UniData displays an error message if unable to execute this command due to the presence of a trigger in the file header. For more information about UniData triggers, see *Using UniData*.

Warning: CLEAR.FILE deletes all data records in a file and, for dynamic files, returns the file to its original modulo and size.

You can use an active select list with this command. You can create a select list of file names by selecting VOC records of a particular type or by selecting VOC records by record ID. The following sample UniQuery statements assume ECLTYPE U.

- SELECT VOC WITH F1 LIKE “VOC_*type*”

- SELECT VOC WITH @ID = "*filename*" [[OR] WITH @ID = "*filename*"'...]

UniData handles multipart dynamic files in the following way with this command:

- Truncates dat001 and over001 and removes all other part files, including idx files, at the operating system level.
- Preserves the minimum modulo for the existing file and uses it as the modulo for CREATE.FILE logic, and so forth.
- Uses the current part file.
- May put new part files on different partitions from the original file system.

Warning: When you use a select list to clear files, UniData does not prompt for individual record IDs before deleting all records.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DATA	Deletes the data records in a file.
DICT	Deletes the dictionary records in a file.
filename	The name of the file to be cleared.
FORCE	Deletes the data and / or dictionary records in a file; accepts a synonym file name.

CLEAR.FILE Parameters

Examples

In the following example, UniData deletes all records in the data portion of the CLIENTS demo file:

```
:CLEAR.FILE CLIENTS
CLIENTS is cleared.
:LIST CLIENTS
LIST CLIENTS NAME COMPANY ADDRESS CITY STATE ZIP COUNTRY PHONE
PHONE_TYPE 16:32:47 Jun 14 2001  1

No record listed.

:
```

The next example demonstrates clearing files named in a select list. For this example, a select list was created that contains the names of the CLIENTS and ORDERS demo files. When this list is used with the CLEAR.FILE command, UniData deletes all of the records in the named files. The LIST statements that follow the example confirm this.

```
:SELECT VOC WTH F1 LIKE F AND F2 LIKE "INV..."

2 records selected to list 0.

>CLEAR.FILE
Use select list data(Y/N)? Y
Clear INV_FILE(Y/N)? Y
INV_FILE is cleared.
Next file(Y/N)? Y
Clear INVENTORY(Y/N)? Y
INVENTORY is cleared.
:LIST INVENTORY
LIST INVENTORY INV_DATE INV_TIME PROD_NAME FEATURES COLOR PRICE
QTY REORDER DIFF 15:47:27 Jun 29 2001 1

No records listed.

:LIST INV_FILE
LIST INV_FILE INV_DATE INV_TIME PROD_NAME FEATURES COLOR PRICE QTY
REORDER DIFF 15:47:30 Jun 29 2001 1

No records listed.
```

Related Command

DELETE.FILE

CLEAR.LOCKS

Syntax

CLEAR.LOCKS [*lock_num*]

Synonym

CLEAR-LOCKS

Description

The ECL **CLEAR.LOCKS** command clears semaphore locks previously placed by your UniData session using the LOCK, LINE.ATT, and T.ATT commands. *lock_num* is the number (0 through 64) of the semaphore lock you want to clear. If you do not indicate a lock number, UniData releases all locks you have placed.

Tip: To release locks set by your pid from other terminals or windows, execute SUPERCLEAR.LOCKS. You must be logged in as root on UniData for UNIX or Administrator on UniData for Windows Platforms to use that command.

Example

The following example sets a lock, then clears it, for system resource 4.

```
:LOCK 4
:LIST.LOCKS
 UNO   UNBR  UID UNAME TTY   FILENAME  INBR  DNBR  RECORD_ID M     TIME
DATE
   1  24775 1172 clair pts/0 semaphor    -1     0          4 X 15:03:52 Jun
08
:CLEAR.LOCKS
:LIST.LOCKS
```

Related Commands

LIST.LOCKS, SUPERCLEAR.LOCKS

CLEAR.ONABORT

Syntax

CLEAR.ONABORT

Synonym

CLEAR-ONABORT

Description

The ECL **CLEAR.ONABORT** command clears the setting of an ON.ABORT command.

With the ON.ABORT command, you can stipulate that a UniData command be executed if a subsequent UniBasic program aborts. CLEAR.ONABORT clears this setting.

For more information about creating and running UniBasic programs, see *Developing UniBasic Applications.*

Note: UDT.OPTIONS 105 determines whether to allow ON.ABORT to take effect from a PERFORMC or EXECUTE statement in UniBasic. For more information, see the UDT.OPTIONS Commands Reference.

Example

In the following example, UniData sets ON.ABORT to a paragraph called APOLOGY. Then, UniData clears the setting:

```
:ON.ABORT APOLOGY
:CLEAR.ONABORT
:
```

Related Command

ON.ABORT

CLEAR.ONBREAK

Syntax

CLEAR.ONBREAK

Synonym

CLEAR-ONBREAK

Description

The **CLEAR.ONBREAK** command clears the setting of the ON.BREAK command.

The ECL ON.BREAK command determines the actions UniData takes when a user presses the interrupt key during execution of a UniQuery statement. After CLEAR.ONBREAK executes, a user who presses the interrupt key during execution of these commands is returned to the environment from which he or she executed the command.

Example

After the first command in the following example, UniData executes the sentence MAIN_MENU when a user presses the break key during execution of a UniQuery statement. However, the CLEAR.ONBREAK command removes that setting so that the user is returned to the ECTL prompt after pressing the break key during execution of the previously mentioned UniQuery command.

```
:ON.BREAK MAIN_MENU
:CLEAR.ONBREAK
```

Related Command

ON.BREAK

CLEARDATA

Syntax

CLEARDATA

Description

The ECL CLEARDATA command clears the data stack. After the data stack is cleared, UniData displays subsequent input requests to the terminal screen.

The UniData data stack can be loaded by paragraphs or by the UniBasic DATA command, then they can be read by the UniBasic INPUT commands or paragraph inline prompts.

Examples

The following example shows a UniBasic program that clears the data stack:

```
Top of "CLEAR.PROCESS" IN "BP", 1 line, 19 characters.
001: EXECUTE 'CLEARDATA'
Bottom.
```

The next example shows a VOC sentence that creates select lists and loads the data stack:

```
     VOC RECORD ID==>LAST_NAMES

0 @ID=LAST_NAMES
1 F1=PA
2 F2=SELECT CLIENTS WITH LNAME LIKE "<<Enter first letter of last name:
>>..."
3 F3=DATA <<Enter first letter of last name: >>
4 F4=RUN BP CLEAR.PROCESS
```

In this example, we execute the LAST_NAMES paragraph more than once. If the data stack was not cleared by calling CLEAR.PROCESS, the second time you executed the paragraph, UniData would answer the inline prompt with input from the first execution.

```
:LAST_NAMES
Enter the first letter of last name:  M

11 records selected to list 0.

:LAST_NAMES
Enter first letter of last name:  T

3 records selected to list 0.

:
```

CLEARPROMPTS

Syntax

CLEARPROMPTS

Description

The ECL CLEARPROMPTS command clears all responses to inline prompts in paragraphs. Use this command within a paragraph after an inline prompt.

Note: Through UniData's Process Control Language (PCL), you can create paragraphs that require the user to respond before UniData continues executing the paragraph. For example, a prompt like "Enter a client number" might appear on the user's terminal screen. After the prompt appears, UniData waits for the user to enter a response. The device UniData uses to do this is called an inline prompt.

For more information on PCL and inline prompting, see *Using UniData*.

clearq

Syntax

clearq *qid*

Description

The system-level **clearq** command clears all message queues on the system of messages destined for processes that are no longer alive. *qid* represents the queue number. Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the colon prompt. For more information about clearq and clearing message queues, see *Administering UniData*.

Note: You must log in as root on UniData for UNIX or Administrator on UniData for Windows Platforms to execute the clearq command.

Tip: Execute the UniData system-level ipcstat command from the operating system prompt to get the queue number.

CLR

Syntax

CLR

Synonym

CS

Description

The ECL **CLR** command clears the terminal screen and places the cursor at the upper left side of the screen in the “home” position.

CNAME

Syntax

To change a file name:

CNAME *filename,new_filename*

CNAME *filename* TO *new_filename*

To change a record ID:

CNAME [DICT] *filename old_recordID,new_recordID*

CNAME [DICT] *filename old_recordID* TO *new_recordID*

To change a multilevel part name:

CNAME *filename,old_partname* TO *filename,new_partname*

Description

The ECL **CNAME** command changes the names of files and record IDs. You can change more than one record ID at a time.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	UniData file name. The file can be any hashed file, including multifiles and multidir files.
new_filename	New name assigned to the file.
DICT	Dictionary file. Used when changing dictionary file or record names. **Note**: When you use CNAME to change file names, UniData changes both the dictionary and data file names.
record_ID	Record ID in a file. You may change more than one record ID on the same command line.
new_recordID	New name assigned to the record ID.

CNAME Parameters

Examples

In the following example, UniData changes the name of the INVENTORY demo file to MERCHANDISE. The LIST command that follows demonstrates that the old file name no longer exists and that the name of the dictionary file for INVENTORY also changed.

```
:CNAME INVENTORY,MERCHANDISE
INVENTORY changed to MERCHANDISE.
:LIST INVENTORY
Not a filename “
 INVENTORY
:LIST DICT INVENTORY
Not a filename :
 INVENTORY
```

The next example changes two records IDs in the INVENTORY demo file:

```
:CNAME INVENTORY 53050,NEW53050 56060,NEW56060
53050 changed to NEW53050.
56060 changed to NEW56060.
:
```

The next example creates a multifile named multi_file and a subfile named sub_file, and then uses CNAME to change the subfile name to sub_one.

```
:CREATE.FILE MULTIFILE multi_file,sub_file
modulos for file multi_file,sub_file=4
4 is not a prime number, modulo changed to 5.
Create file multi_file/sub_file, modulo/5,blocksize/1024
Hash type = 0
Added "@sub_file" to DICT multi_file.
:CNAME multi_file,sub_file TO multi_file,sub_one
multi_file,sub_file changed to multi_file,sub_one.
```

cntl_install

Syntax

cntl_install

Description

The system-level **cntl_install** command reinitializes counters in the udt.control.file, the log files, the archive files, the system.status.file, the restart.fileend file, and the restart.newblk file, all located in /usr/ud60/include. cntl_install executed the log_install command, for use with recoverable files.

Warning: Since cntl_install reinitializes files needed for recovery, make sure none of these files are needed before executing cntl_install.

Note: To execute the cntl_install command, you must log in as root.

For more information about the Recoverable File System, see *Administering the Recoverable File System*.

COMO

Syntax

COMO [ON [HUSH] | OFF] [APPEND | DELETE | LIST | SPOOL [-T]] *record*

Description

The ECL COMO command creates a history of a UniData session by sending user input and system output to a designed record. UniData stores the COMO record in a UniData DIR-type file called _PH_ within the current account. UniData stores the COMO record by preceding the record name by _O.

Tip: Turn off COMO files when you finish recording your UniData session. If you do not, UniData continues to record input and output until you end the UniData session. This could cause the _PH_ file to become extremely large. Periodically review the _PH_ file and delete records that are no longer needed.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
record	The name you assign to the COMO session. If you do not indicate a record name, UniData prompts to name a record or quit the COMO session.
APPEND	Opens an existing COMO record and appends new information to the end of it.
DELETE	Deletes the COMO record from the _PH_ file.
HUSH	Directs output to the COMO record, suppressing output to the terminal.

COMO Parameters

Parameter	Description
LIST	Lists the COMO records in _PH_.
OFF	Ends a COMO session.
ON	Starts a COMO session.
SPOOL	Sends a copy of a COMO record to the printer. The COMO session must be turned OFF.
-T	Instructs the SPOOL option to send the output to the terminal, not the printer.

COMO Parameters (continued)

Note: When you use the COMO command with APPEND, LIST or SPOOL, record is the name of the COMO record without the O_ prefix.

Examples

In the following example, UniData starts a COMO session, lists five records in the CLIENTS demo data file, and then ends the COMO session:

```
:COMO ON save
/home/claireg/_PH_/O_save established
:LIST INVENTORY SAMPLE 5
LIST INVENTORY SAMPLE 5 INV_DATE INV_TIME PROD_NAME FEATURES COLOR
PRICE QTY REO
RDER DIFF 13:27:06 Jun 11 2001 1
INVENTORY        15001
Inventory Date   08/20/1995
Inventory Time     01:00PM
Product Name     Modem
Features         14.4K Internal V34
Color      Price      Quantity Reorder Difference
N/A           $119.00     7486      40       7446

INVENTORY        35000
Inventory Date   07/09/1995
Inventory Time     10:00AM
Product Name     Speaker
Features         250W, Direct/reflecting
Color      Price      Quantity Reorder Difference
Black         $198.93      148      50         98
Charcoal      $198.93      125      50         75

INVENTORY        15002
Inventory Date   08/12/1995
Inventory Time     07:00AM
Product Name     Modem
Enter <New line> to continue...Q
:COMO OFF
/home/claireg/_PH_/O_save closed
:
```

The next example prints the contents of the COMO file. Notice that you enter the como session name without the prefix of “O_”:

```
:COMO SPOOL save
:
```

Two COMO sessions can run at the same time. When you open first one session and then another, UniData nests the second session within the first. The first session is REC_1. The second session, REC_2, is initiated with REC_1 is still active.

Execute SPOOL to display the COMO record for REC_2 to the screen. Notice that this record consists only of the input and output from the time UniData established the session for REC_2 until the session ended:

```
:COMO SPOOL REC_2 -T
:LIST CTLGTB
LIST CTLGTB 09:34:49 Jun 30 2001 1
CATALOG NAME..............

SCHEMA_UPDATE_PRIVILEGES
SCHEMA_LIST_USERS
SCHEMA_VIEW_CHECK
...
Enter <New line> to continue...A
:COMO OFF REC_2
```

The next example shows the COMO session for REC_1. Notice that UniData recorded all input before, after, and including the session for REC_2:

```
:COMO SPOOL REC_1 -T
/home/claireg/demo/_PH_/O_REC_1 established

:LIST VOC WITH F1 LIKE "F"
LIST VOC WITH F1 LIKE "F" 09:34:05 Jun 30 2001 1
VOC........

privilege
INV_FILE
inv
_REPORT_
ENGLISH.MS
...
Enter <New line> to continue ...Q
:COMO ON REC_2
/home/claireg/demo/_PH_/O_REC_2 established
:LIST CTLGTB
LIST CTLGTB 09:34:49 June 30 2001 1
CATALOG NAME..........

SCHEMA_UPDATE_PRIVILEGES
SCHEMA_LIST_USERS
SCHEMA_VIEW_CHECK
...
Enter <New line> to continue...Q
:COMO OFF REC_2
/home/claireg/demo/_PH_/O_REC2 closed
:COMO OFF REC_1
:
```

COMPILE.DICT

Syntax

COMPILE.DICT *filename* [*attribute*]

Synonyms

CD, COMPILE-DICT

Description

The ECL **COMPILE.DICT** command checks the syntax of a virtual attribute. If you do not specify *attribute*, Unidata compiles all virtual attributes in *filename*. Compiling creates attributes 8 and 9 in the dictionary record for the virtual attribute.

UniData compiles a virtual attribute each time it is executed unless it is compiled in advance by COMPILE.DICT. Compiling in advance may speed execution.

You must compile virtual attributes before you can execute them in UniBasic programs (with the CALCULATE, {}, or ITYPE functions).

If COMPILE.DICT is unsuccessful, @SYSTEM.RETURN.CODE is set to -1, if it is successful @SYSTEM.RETURN.CODE is set to 0.

For more information about virtual attributes, see *Using UniData*.

Tip: Use AE (Alternate Editor) to display the dictionary record for a compiled virtual attribute. UniEntry does not display attributes 8 and 9.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of the file that contains the virtual attribute.
attribute	Virtual attribute name.

COMPILE.DICT Parameters

Examples

The ORDERS demo file contains the virtual attribute GRAND_TOTAL. In the next example, UniData compiles this virtual attribute:

```
:COMPILE.DICT ORDERS GRAND_TOTAL
GRAND_TOTAL=PRICE*QTY; SUM(SUM(@1))
Virtual field GRAND_TOTAL is syntactically correct.
```

The next example lists the dictionary record for the GRAND_TOTAL virtual attribute. Notice attributes 8 and 9, created by the compile process:

:Note: ü *and* y *are (nonprinting) UniData delimiters. The character used to display them varies with terminal or printer type.*

```
:AE DICT ORDERS GRAND_TOTAL
Top of "GRAND_TOTAL" in "DICT ORDERS", 9 lines, 107 characters.
*--: P
001: V
002: PRICE*QTY; SUM(SUM(@1))
003: MD2,$
004: Grand Total
005: 14R
006: S
007:
008: GRAND_TOTALyQTYü6üPRICEü7yPRICE*QTY; SUM(SUM(@1))
009: ORDERS
Bottom.
```

CONFIGURE.FILE

Syntax

CONFIGURE.FILE *filename* [SPLIT.LOAD *split_percent*] [MERGE.LOAD *merge_percent*] [MINIMUM.MODULO *modulo*] [KEYONELY | KEYDATA]

Synonym

CONFIGURE-FILE

Description

The ECL **CONFIGURE.FILE** command changes the split load, merge load, minimum modulo, and / or split / merge type for a dynamic file. A dynamic file is one that UniData automatically resizes when data is added or removed, according to the SPLIT.LOAD and MERGE.LOAD percentages.

For more information about dynamic files, see *Administering UniData* and *Using UniData*.

Tip: The default settings for split and merge thresholds are controlled by parameters in the UniData configuration file (/usr/ud60/include/udtconfig on UniData for UNIX or \udthome\include\udtconfig on UniData for Windows Platforms). The defaults are different between KEYONLY and KEYDATA dynamic files. To change the defaults for your system, edit the lines for SPLIT_LOAD and MERGE_LOAD (for KEYONLY files) or KEYDATA_SPLIT_LOAD and KEYDATA_MERGE_LOAD (for KEYDATA files) in the udtconfig file.

Note the following points about CONFIGURE.FILE:

- If you change the split / merge type, and you do not specify the split load or merge load in the command line, CONFIGURE.FILE sets the split and merge loads to the defaults for the split / merge type you specify. CONFIGURE.FILE displays a message to the screen if the split and merge load percentages are changed.

- CONFIGURE.FILE changes only the file's configuration parameters. This command does not redistribute the records in the file, and does not split or merge the file. After you run CONFIGURE.FILE, use ANALYZE.FILE and the guide utility to determine if you should rebuild your file with REBUILD.FILE.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of a UniData dynamic file.
SPLIT.LOAD *split_percent*	Load factor at which a group is eligible for splitting. The default splitting threshold is 60 percent for KEYONLY files and 95 percent for KEYDATA files.
MERGE.LOAD *merge_percent*	Load factor at which groups are eligible for merging. The default merging threshold for both KEYONLY and KEYDATA files is 40 percent.
MINIMUM.MODULO *modulo*	Minimum number of groups in the file.
[KEYONLY | KEYDATA]	Split / merge type for the target file. If this is not specified, CONFIGURE.FILE keeps the split / merge type of the source file.

CONFIGURE.FILE Parameters

Examples

The following examples use a copy of the INVENTORY demo file:

```
:ANALYZE FILE INVENTORY
Dynamic File name                    = INVENTORY
Number of groups in file (modulo)    = 19
Minimum groups of file               = 19
Hash type = 0, blocksize = 1024
Split load = 60, Merge load = 40
Split/Merge type = KEYONLY
...
```

In the following example, the split load and merge load are changed:

```
:CONFIGURE.FILE INVENTORY SPLIT.LOAD 70 MERGE.LOAD 45
:ANALYZE.FILE INVENTORY
Dynamic File name                    = INVENTORY
Number of groups in file (modulo)    = 19
Minimum groups of file               = 19
Hash type = 0, blocksize = 1024
Split load = 70, Merge load = 45
Split/Merge type = KEYONLY
...
```

In the next example, the split/merge mode is changed to KEYDATA:

```
:CONFIGURE.FILE INVENTORY KEYDATA
Split load has been implicitly changed to 95
Merge load has been implicitly changed to 40

:ANALYZE.FILE INVENTORY
Dynamic File name                    = INVENTORY
Number of groups in file (modulo)    = 19
Minimum groups of file               = 19
Hash type = 1, blocksize = 1024
Split load = 95, Merge load = 40
Split/Merge type = KEYDATA
...
```

Related Commands

ANALYZE.FILE, guide, memresize, REBUILD.FILE

confprod

Syntax

confprod

Description

The system-level command **confprod** displays and updates licensing information for your system.This command also provides a configuration code that you must supply to IBM after installing UniData. For more information about confprod and licensing products on UniData, see *Installing and Licensing UniData Products*.

Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the colon prompt.

Note: To execute the confprod command, you must be logged on as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

You have 30 days to authorize UniData after installation.

Example (UniData for UNIX)

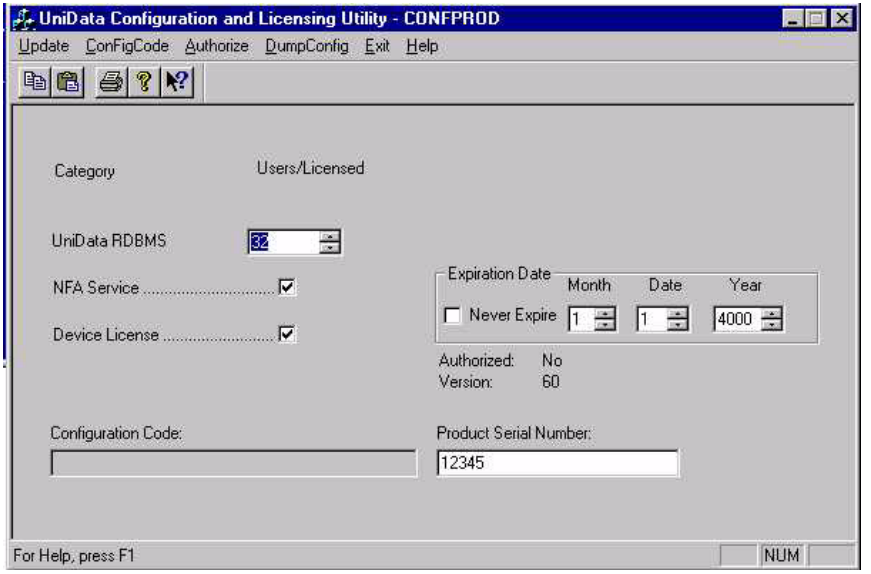
confprod displays the products licensed on your system and the number of UniData licenses authorized, as illustrated in the following example:

```
%confprod
```

For an explanation of the commands listed in the preceding example, see *Installing and Licensing UniData Products*.

Example (UniData for Windows Platforms)

confprod displays the products licensed on your system and the number of UniData licenses authorized, as shown in the next example:



Note: Use this command at the MS-DOS prompt.

CONNECT

Syntax

CONNECT *data.source* [*option setting* [*option setting*...]]

Description

Use the **CONNECT** command with UniBasic SQL Client Interface (BCI) to connect to a data source from a UniData client. You enter the CONNECT command at the ECL prompt. The CONNECT command enables you to submit SQL statements to the data source and receive results at your terminal.

While you are connected to a data source, you can enter any SQL statement understood by the DBMS engine on the data source, including SELECT, INSERT, UPDATE, DELETE, GRANT, and CREATE TABLE. ODBC data sources can use SQL language that is consistent with the ODBC grammar specification as documented in Appendix C of *Microsoft ODBC 2.0 Programmers Reference and SDK Guide*.

The CONNECT command runs in autocommit mode: that is, all changes made to the data source DBMS are committed immediately. Do not use transaction control statements such as TRANSACTION START, TRANSACTION COMMIT, and TRANSACTION ABORT when you are using CONNECT.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
data.source	The name of the data source to which you want to connect. The data source is an ODBC data source defined on your system. For example, on Windows platforms, a data source is defined in the ODBC Data Source Administrator.
options	You can specify any of the following options with the CONNECT command. See the following section for a detailed description of each option ■ BLOCK ■ NULL ■ PREFIX ■ UDOUT ■ VERBOSE ■ WIDTH

CONNECT Parameters

Command Options

You can specify any option with the CONNECT command. You must specify a setting for the option.

BLOCK

The BLOCK option defines how UniData BCI terminates input statements. *setting* is one of the following:

Setting	Description
ON	Enables BLOCK mode. In this mode, you can enter a series of SQL statements, ending with a ; (semicolon). To terminate the block of SQL statements, press RETURN immediately after an SQL+ prompt.
OFF	Disables BLOCK mode. In this mode if you type a semicolon at the end of a line of input, UniData BCI terminates your input and sends it to the data source. This is the default setting.
string	Enables BLOCK mode (see ON, above). *string* must be from 1 to 4 characters. To terminate the block of SQL statements, enter string immediately after an SQL+ prompt.

BLOCK Option Settings

For more details, see *Using the UniBasic SQL Client Interface (BCI).*

NULL

The way UniData BCI treats null values coming from the data source depends on the setting of the NULL_FLAG parameter in the udtconfig file.

NULL Flag	Description
0	Remote nulls are translated to or from the data source as an empty string.
1	Remove nulls are translated to or from the data source as the null value mark.

NULL FLAG Settings

The NULL option defines how to display the SQL null value. This option is only valid if NULL_FLAG is set to 1 in the udtconfig file, located in /usr/ud60/include. *setting* is one of the following:

Setting	Description
SPACE	Displays the SQL null value as a blank space.
NOCONV	Displays the SQL null value as defined by null value mark setting in UDTLANGCONFIG.
string	Displays the SQL null value as *string*. The string can be from 1 to 4 characters. By default, null is displayed as the 4-character string NULL.

NULL Option Settings

Prefix

The PREFIX option defines the prefix character for local commands. *setting* is any valid prefix character. The default prefix character is a period (.). You can use only the following characters as the prefix character:

Character	Description
!	Exclamation point.
@	At sign.
#	Hash sign.
$	Dollar sign.
%	Percent.
&	Ampersand.
*	Asterisk.
/	Slash.
\	Backslash.
:	Colon
=	Equal sign.

Valid Prefix Characters

Character	Description
+	Plus sign.
-	Minus sign.
?	Question mark.
(	Left parenthesis.
)	Right parenthesis.
{	Left brace.
}	Right brace.
[	Left bracket.
]	Right bracket.
'	Left quotation mark.
‘	Right quotation mark.
.	Period.
|	Vertical bar.
“	Double quotation mark.
,	Comma.

Valid Prefix Characters (continued)

For more details, see *Using SQL Client Interface (BCI).*

UDOUT

The UDOUT option specified how to handle output from SELECT statements executed on the data source. *setting* is either:

Setting	Description
filename	Stores output in *filename* on the client, then displays the output from *filename*. If the file does not exist, the CONNECT command creates it.
OFF	Displays output from the data source directly on the screen of the client. This is the default setting.

UDOUT Option Settings

For more details, see *Using SQL Client Interface (BCI).*

VERBOSE

The VERBOSE option displays extended column information and system messages. *setting* is either:

Setting	Description
ON	Enables verbose mode. In this mode, the name, SQL data type, precision, scale, and display size are displayed for each column definition when selecting data from the data source. Error messages are displayed in extended format that includes the type of call issued, status, SQLSTATE, error code generated by the data source, and the complete error text.
OFF	Disables verbose mode. This is the default setting.

VERBOSE Option Settings

WIDTH

The WIDTH option defines the width of display columns. *setting* is one of the following:

Setting	Description
col#,width	Sets the width of column *col#* to *width*. Do not enter a space after the comma. Specify *col#* as * (asterisk) to set the width of all columns. *width* can be from 4 to the maximum line length allowed by your terminal. The default width for all columns is 10.
T	Truncates data that is wider than the *width* you specify. This is the default setting.
F	Folds data that is wider than the specified *width* onto multiple lines.
?	Displays the current column width settings, and tells whether data will be truncated or folded.

WIDTH Options Settings

CONTROLCHARS

Syntax

CONTROLCHARS {OFF | ON | IGNORE}

The ECL CONTROLCHARS command determines UniData's response to user input of nonprinting characters (control or escape sequences) in response to UniBasic INPUT statements. You can:

- Allow nonprinting characters.
- Convert nonprinting characters to tilde (~).
- Ignore input of nonprinting characters.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
ON	Allows nonprinting characters.
OFF	Converts nonprinting characters to tilde (~).
IGNORE	Does not return nonprinting characters. Screens out the escape character and most of the ASCII codes between 000-031 and 127-255 inclusive. IGNORE does not screen out the following ASCII codes within those ranges: ■ 008—backspace ■ 010 and 013—line feed and carriage return ■ 009—tab

CONTROLCHARS Parameters

Note: UDT.OPTIONS 83 validates the escape character (ASCII code 027) as input to UniBasic INPUT statements. When this option in ON, UniBasic accepts the escape character as valid input when CONTROLCHARS is set to OFF and IGNORE, but screens out other control characters.

UDT.OPTIONS 103 determines how UniData treats the TAB character when CONTROLCHARS is set to off or ignore.

Examples

In the following example, CONTROLCHARS converts nonprinting characters to tilde (~).

```
:CONTROLCHARS OFF
```

In the next example, CONTROLCHARS allows nonprinting control or escape sequences as user response to the UniBasic INPUT statement:

```
:CONTROLCHARS ON
```

In the next example, CONTROLCHARS screens out nonprinting characters:

```
:CONTROLCHARS IGNORE
```

convcode

Syntax

convcode {*filename* | *directory* | -i}

Description

The system-level **convcode** command converts UniData object files from Motorola 68000 internal integer format. Format information is embedded within the file header. This command automatically determines if object files match the present machine integer format. If the files do not need to be converted, UniData displays a message that no files were converted.

You can run convcode more than once on a UniData file to convert between the two formats.

Execute this command from the system prompt, or use the ECL ! (bang) command to execute convcode from the colon prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	UNIX name of the file to be processed.
directory	Name of a dictionary that holds files, all of which are to be processed. The convcode command traverses the directory recursively.
-i	Run convcode interactively.

convcode Parameters

Related Commands

convdata, convidx

convdata

Syntax

convdata [-s] {*filename* [*filenameM*...*filenameN*] | [-r] *directory*}

Description

The system-level convdata command converts UniData hashed data files from Motorola 68000 internal integer format to Intel 386 internal integer format. Format information is embedded within the file header. This command automatically determines if files match the present machine integer format. If files do not need to be converted, UniData displays a message that no data files were converted.

You can run convdata more than once on a UniData file.

Execute this command at the system prompt, or use the ECL ! (bang) command to execute this command from the colon prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of a UniData file to convert. To use more than on file name, separate the names with spaces.
-s	Suppresses requests for operator action. Error messages still appear.
-r	Processes subdirectories recursively. Used only with the *directory* option.
directory	The name of a directory that contains file names to be processed by convdata.

convdata Parameters

Example

The following example illustrates an attempt to convert the format for several files. If the files do not need to be converted, UniData displays informational messages.

```
 % convdata -r .
./BP_SOURCE/GPA1: not a Unidata file
./BP_SOURCE/PHONE_FMT: not a Unidata file
./BP_SOURCE/PSTLCODE_FMT: not a Unidata file
./BP_SOURCE/UP_NAME: not a Unidata file
./BP_SOURCE/_GPA1: not a Unidata file
./BP_SOURCE/_PHONE_FMT: not a Unidata file
./BP_SOURCE/_PSTLCODE_FMT: not a Unidata file
./BP_SOURCE/_UP_NAME: not a Unidata file
./BP_SOURCE/AddRecord: not a Unidata file
./BP_SOURCE/DelRecord: not a Unidata file
./BP_SOURCE/EXAMPLE: not a Unidata file
./BP_SOURCE/EXAMPLE_C: not a Unidata file
./BP_SOURCE/EXAMPLE_CPP: not a Unidata file
./BP_SOURCE/EXAMPLE_DELPHI: not a Unidata file
./BP_SOURCE/FndRecord: not a Unidata file
./BP_SOURCE/UpdRecord: not a Unidata file
./BP_SOURCE/_AddRecord: not a Unidata file
./BP_SOURCE/_DelRecord: not a Unidata file
./BP_SOURCE/_EXAMPLE_C: not a Unidata file
./BP_SOURCE/_EXAMPLE_CPP: not a Unidata file
.
.
.
./D_BRI1234: converted
./BRI1234: converted (dynamic file)
41 data file(s) converted
#
```

Related Commands

convcode, convidx

convhash

Syntax

convhash [-T *targetdir*] [*filename1 ... filename(n)*]

Description

The system-level convhash command converts static hashed files to dynamic hashed files. This tool invokes the UniData memresize tool, creating a dynamic file with the following characteristics:

- Minimum modulo — the current modulo of the static file
- Hash type — the current hash type of the static file
- Split / merge type — KEYONLY (the UniData default)

Note: We recommend that you use the memresize command, rather than the convhash command, to convert a static file to a dynamic file.

Execute this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename1...filename(n)	The file or list of files to be converted. You can name more than one file by separating the file names with spaces. filename must be a static hashed file or multilevel file.
-T *targetdir*	Directory in which you want UniData to store the data portion of the converted file. If you do not name a directory, UniData stores the new dynamic file in the same directory as the static file. **Note**: If you specify the -T option, the DICT portion of the file remains in your current working directory. To access the data, you must edit the VOC pointer in your current account to add the path name for the data file.

convhash Parameters

Examples

In the following example, UniData converts a static hashed file called CONVHASH.TEST and the subfiles of a multilevel file called MULTI1 to dynamic files:

```
% convhash CONVHASH.TEST MULTI1
Converting 'CONVHASH.TEST'...
'CONVHASH.TEST' has been successfully converted
Converting 'MULTI1/FILE1'...
'MULTI1/FILE1' has been successfully converted
Converting 'MULTI1/FILE2'...
'MULTI1/FILE2' has been successfully converted
Converting 'MULTI1/FILE3'...
'MULTI1/FILE3' has been successfully converted
%
```

You can verify the file type for the converted file by displaying file statistics. The next example uses the ANALYZE.FILE command:

```
:ANALYZE.FILE MULTI1,FILE3
Dynamic File name = MULTI1,FILE3
Number of groups in file (modulo) = 11
Minimum groups of file = 11
Hash type = 0, blocksize = 1024
Split load = 60, Merge load = 40
Split/Merge type = KEYONLY
Group Keys Key Loads Percent
================================================
0 19 331 32
1 21 358 34
2 24 407 39
...
```

When you use convhash to convert a file, no splitting or merging takes place. This could result in a poorly sized file immediately after convhash. Use guide or ANALYZE.FILE to determine if you should rebuild your new dynamic file. The following example shows the output in the GUIDE_ADVICE.LIS (generated by the guide utility), indicating that a dynamic file should be rebuilt.

```
% pg GUIDE_ADVICE.LIST
FAMILY_FILE1
Management advice:
Running REBUILD.FILE may improve performance
for access to the file. This conclusion was reached
for the following reasons:
- File is in level two overflow.
- File has 101 groups over split load.
Files processed: 1
Errors encountered: 0
%
```

convidx

Syntax

convidx [-r] [-s] [*filename* [*filenameM*...*filenameN*] | *directory* [*directoryM*...*directoryN*]

Description

The system-level **convidx** command converts UniData index files from Motorola 68000 internal integer format to Intel 386 internal integer format. Format information is embedded within the file header. This command automatically determines if files match the present machine integer format. If files do not need to be converted, UniData displays a message to that effect.

You can run convidx more than once on a UniData file.

Static index files have a prefix of X_. Dynamic index files are named idx001, idx002,.... See the *Using UniData* manual for more information about working with index files and alternate key indexes.

Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL prompt.

Parameters

Parameter	Description
-r	Processes subdirectories recursively. Converts all index files in *directory*.
-s	Suppresses system messages.
filename	The index to be converted. Separate multiple index names with spaces.
directory	The UniData DIR-type file that contains indexes to be converted. Separate directory names with spaces.

convidx Parameters

Examples

In the following example, UniData attempts to convert the index file for the CLIENTS demo file. CLIENTS is a static file, so the index file has a X_ prefix. Since the index is already converted, UniData displays informational messages instead:

```
% convidx -r X_CLIENTS
X_CLIENTS: already been converted
0 index file(s) converted.
%
```

The next example shows an attempt to convert two dynamic file index files. Since they have already been converted, UniData displays informational messages instead:

```
% convidx -r INVENTORY ORDERS
INVENTORY/idx001: already been converted
ORDERS/idx001: already been converted
0 index file(s) converted.
%
```

Related Commands

convdata, convcode

convmark

Syntax

convmark [-t] *language_group_ID* [[*path1* [*path2*...]]

convmark [-t] -s *old_value* [-d *new_value*][[*path1* [*path2*...]]

Description

The system-level convmark command searches for and converts ASCII values in UniData files. *new_value* must be one that is not contained in the file to be converted.

Based on the option selected, UniData does one of the following:

- Displays the number of occurrences of a particular ASCII value.
- Counts the number of UniData delimiters in files.
- Converts a single ASCII character (ASCII values 128 – 255 only).
- Converts the UniData delimiters for your language group. (Be sure you have changed the language group with the system-level command udtlangconfig. For instructions, see *UniData International*.)

convmark Constraints

You cannot use the convmark command to convert in the following conditions:

- If your source file contains the new ASCII values the ones to which you are attempting to convert no data in the file is converted. UniData instead returns a message indicating that the data already contains the new mark, and returns the cursor to the ECL prompt. This does not mean that the file has been converted or that it does not require conversion. You must review and change the records manually.

- On UniData for UNIX, directories indicated by path1, and so forth, cannot contain any UNIX links (created with the UNIX ln command). If they do, convmark produces an error message and aborts.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-t	For use in test mode. Returns the number of files in the specified directory that need to be converted, but does not convert them. You can combine -t with any other options.
language_group_ID	The language group ID is made up of the ASCII values that represent the record mark, the cursor control escape sequence, and the null value for that language group: ■ 159 / 130 / 129 French, Japanese, and English ■ 255 / 192 / 129 English
path 1 [*path2*...]	The full path to files to convert. May be for a directory (all files are converted) or for a file name. On UniData for UNIX, these directories cannot contain UNIX links.
-s *old_value*	Used without *new_value*, counts the occurrences of *new_value*. Used with *new_value*, converts from *old_value*. Must be a single ASCII value from 128 through 255.
-d *new_value*	Replacement value. Must be a single ASCII value from 128 through 255. **Note**: If *new_value* already appears in the data, UniData does not execute the conversion. Instead, an informational message appears and the cursor returns to the environment from which you executed convmark.

convmark Parameters

Examples

In the following example, UniData counts the occurrences of ASCII value 254 in the ORDERS demo file:

```
:convmark -s 254 ORDERS
ORDERS: number of value 254: 1152
1 UniData file(s) need conversion.
%
```

In the next example, the -t option counts ASCII value 254 in all files in the current directory and in all subdirectories, but does not convert those characters. If the user in this example had not included the -t option, the command would have converted all ASCII values 254 to 129 (the null value in the English language group):

```
% convmark -t -s 254 -d 129
./BP/GREETING: not a UniData file
./BP/_GREETING: not a UniData file
./BP/TEST_PROG: not a UniData file
./BP/_TEST_PROG: not a UniData file
./BP/CLEAR.PROCESS: not a UniData file
./BP/_CLEAR.PROCESS: not a UniData file
./BP_SOURCE/GPA1: not a UniData file
./BP_SOURCE/PHONE_FMT: not a UniData file
./BP_SOURCE/PSTLCODE_FMT: not a UniData file
./BP_SOURCE/UP_NAME: not a UniData file
./BP_SOURCE/_GPA1: not a UniData file
./BP_SOURCE/_PHONE_FMT: not a UniData file
./BP_SOURCE/_PSTLCODE_FMT: not a UniData file
./BP_SOURCE/_UP_NAME: not a UniData file
./CATEGORIES: no conversion is need
./CLIENTS: need conversion.
./COURSES: need conversion.
./CUSTOMER: need conversion.
./D_BP: need conversion.
./D_BP_SOURCE: need conversion.
./D_CATEGORIES: need conversion.
./D_CLIENTS: need conversion.
...
40 UniData file(s) need conversion.
```

In the following example, convmark converts all ASCII values 129 (the null value in the English language group) to 193.

The following is a display of record 40008 in the demo INVENTORY file, previously modified by the addition of the null value to each multivalued and multi-subvalued attribute. Notice lines 5 – 8.

Note: The UniData-supplied editor AE is used here, and the user has pressed Shift-6 to display nonprinting characters.

```
...
*--: T
Top.
*--: P
001: 10026
002: 53760
003: Telephone
004: Cordless 9 # Memory
005: Burgundy^253Tan^253Black^253White^253^129
006: 350^253200^253300^253148^253^129
007: 6992^2536992^2536992^2536992^253^129
008: 70^25370^25370^25370^253^129
Bottom.
*--:
```

Next, after terminating the UniData session, the user changes directories to udthome, and executes convmark to accomplish the conversion:

```
% convmark -s 129 -d 193 /home/carolw/demo/INVENTORY
WARNING: All 129's in data of the given file(s) will be
replaced with 193. Are you sure (Y/N) ? y

/home/carolw/demo/INVENTORY: converted
1 UniData file(s) were converted successfully.
```

Here is the same record, 40008, redisplayed to show the converted characters: ASCII 129 has been converted to 193 for each multivalued and multi-subvalued attribute (lines 5 through 8):

```
...
*--: T
Top.
*--: P
002: 53760
003: Telephone
004: Cordless 9 # Memory
005: Burgundy^253Tan^253Black^253White^253^193
006: 350^253200^253300^253148^253^193
007: 6992^2536992^2536992^2536992^253^193
008: 70^25370^25370^25370^253^193
Bottom.
*--:
```

Related Command

udtlangconfig

CONVERT.SQL

Syntax

CONVERT.SQL [*filename*][*length*] [CHECKONLY | FORCE] [PUBLIC [*privilege*]]

Synonym

CONVERT-SQL

Description

The ECL **CONVERT.SQL** command checks the UniData file for conformance to ODBC's requirements. If it detects an inconsistency, UniData responds depending upon the CONVERT.SQL option selected. If you do not use the CHECKONLY, FORCE, or PUBLIC option, UniData displays each file and attribute name that does not conform to ODBC requirements, suggests an acceptable name, and waits for you to enter an acceptable name or press ENTER to accept the generated name.

Note: To execute the CONVERT.SQL command, you must be the owner of the file or a system administrator or another user with root access on UniData for UNIX or as Administrator on UniData for Windows Platforms.

In the conversion process, UniData takes the following actions:

- Checks the name of the file being converted. If filename is ODBC-compliant, UniData uses this name for the file. If filename is not ODBC compliant, UniData creates a new, duplicate dictionary file with a compliant name for use by ODBC / UniData SQL.
- Checks attribute specifications for missing value code and format specification.

- Creates synonyms (also called aliases) in the dictionary for attribute names that do not conform to ODBC's conventions. For each noncompliant attribute name, UniData creates or adds an entry in the attributes @SYNONYM and @ORIGINAL to link the new compliant attribute name with the original attribute name.
- Adds conforming names of the converted files to the UniData SQL privilege table.

CONVERT.SQL does not:

- Change the data portion of files being converted.
- Create 1NF schema (1NF views or subtables); therefore, converted tables are not necessarily accessible through UniDesktop tools. For more information on UniData ODBC, see *Developing UniData ODBC Applications*.

Note: Converted files are called base tables.

For a table to be accessible through UniData SQL, it must meet the following conditions:

- The table and attribute name must:
  - Not be longer than 30 characters.
  - Be made up of alphabetic characters, numbers, and special characters: _, @, #, $; the first character must be alphabetic. Be unique among table, subtable, and view names, and UniData SQL reserved words.
- If an attribute name is part of an association, the association name must exist in the dictionary as a PH attribute.
- An association may not contain a singlevalued (S) attribute.

For information about using UniData SQL, see *Using UniData SQL*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Specifies the name of the file to convert. If *filename* is omitted, CONVERT.SQL converts all file names contained in the active select list, if one exists.
length	Specifies the length of the input file name (maximum is 30 characters).
CHECKONLY | FORCE	CHECKONLY reports the problems found in the conversion process, but does not make any changes. FORCE makes necessary file changes during the conversion process and displays changes on the terminal. UniData does not prompt for user input.
PUBLIC	Automatically grants the privilege specified in *privilege* to all users. If PUBLIC is specified, but *privilege* is omitted, *privilege* defaults to ALL.
privilege	Specifies the privileges to grant. You may use the following options: ALL, INSERT, UPDATE, DELETE, or SELECT. For further information, see the "Granting Privileges" section in *Using UniData SQL*.

CONVERT.SQL Parameters

Example

In the following example, UniData converts a file named test.fil so it can be accessed in UniData SQL. During the conversion process, the system prompts the user for information. User responses appear in boldface type.

```
:CONVERT.SQL test.fil
default name length 30 is used
checking file 'test.fil' ...
single-valued field 'NUM-FLD' will be dropped from association
'NUM-DOLLAR'
association 'NUM-DOLLAR' has no corresponding PH field
PH field 'NUM-DOLLAR' has been created
invalid FMT '' in field 'account_no'
enter new FMT [number[R10R
FMT spec of field 'account_no' has been changed to '10R'
field name '@ID' will be changed to 'ID'
enter <CR> to accept, or enter a new synonym:
field name 'NUM-FLD' will be changed to 'NUM_FLD'
enter <CR> to accept, or enter a new synonym:
field name 'dollar$' will be changed to 'dollar_'
enter <CR> to accept, or enter a new synonym:
invalid FMT 'T' in field '@CHAR%'
enter new FMT [number[R10T
FMT spec of field '@CHAR%' has been changed to '10T'
field name '@CHAR%' will be changed to 'CHAR_'
enter <CR> to accept, or enter a new synonym: CHAR_FLD
invalid FMT 'T' in field '_FLDNAME'
enter new FMT [number[R8T
FMT spec of field '_FLDNAME' has been changed to '8T'
field name '_FLDNAME' will be changed to 'FLDNAME'
enter <CR> to accept, or enter a new synonym:
...
synonym 'ID' has been created for field '@ID'
synonym 'NUM_FLD' has been created for field 'NUM-FLD'
synonym 'dollar_' has been created for field 'dollar$'
synonym 'CHAR_FLD' has been created for field '@CHAR%'
synonym 'FLDNAME' has been created for field '_FLDNAME'
synonym 'NUM_DOLLAR' has been created for field 'NUM-DOLLAR'
7 conversions have been made to dictionary
file name 'test.fil' will be changed to 'test_fil'
enter <CR> to accept, or enter your own synonym name:
file synonym 'test_fil' has been added to VOC
1 file has been converted
:
```

COPY

Syntax

COPY FROM [DICT] *filename1* [TO [DICT] *filename2*][id [...] | *id, new_id* [...] | ALL] [DELETING | OVERWRITING | SQUAWK]

Description

The ECL **COPY** command copies individual records from one file to another or within the same file. If you include the DICT keyword, UniData copies dictionary records.

The dictionary and data files must already exist before you copy records into them. See CREATE.FILE for instructions on creating UniData dictionary and data files.

UniData displays an informational message if unable to execute a COPY...DELETING statement due to the presence of a trigger. For more information about UniData triggers, see *Using UniData* or *Developing UniBasic Applications*.

Warning: You cannot use system-level commands (such as cp and tar) to copy UniData recoverable files while UniData is running. If you use these commands on recoverable files, you could corrupt data.

In ECLTYPE P, the COPY command has the following syntax: COPY filename [*]

Notice the following:

- The FROM, DELETING, OVERWRITING, and SQUAWK keywords are not valid.
- A left (open) parenthesis must proceed a file name.

You do not enter a target file name; UniData prompts for it. The optional asterisk copies all records.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	A UniData file. *filename* must be a record in the VOC file. *filename1* and *filename2* can refer to the same file. If you are making a copy of a record in the same file, do not use the TO keyword.
DICT	The Dictionary file.
FROM	Copies records from a source file, *filename1*.
TO	Copies records to a target file, *filename2*.
id	The record ID to be copied. You can copy more than on record ID at the same time by separating multiple record IDs with a space. **Note**: Remember, when you copy records from a dictionary file, the record ID is the dictionary attribute name.
new_id	The new name you assign to a record ID you are copying.
ALL	Copies all records from the source file to the target file.
DELETING	Deletes records from the source file after they are copied to the target file.
OVERWRITING	Overwrites any record of the same name already present in *filename2*. **Warning**: UniData does not prompt to confirm that you intend to overwrite the record.
SQUAWK	Lists the records being copied to the display terminal.

COPY Parameters

Copying the Dictionary

After you copy a UniData file, you may want to copy the dictionary portion of the file you have copied to the new dictionary. If you created a new file to copy records to, the dictionary portion of the new file most likely contains the @ID record only. A UniQuery statement executed against the new file may look something like the following example, indicating that you have not copied the dictionary attributes.

```
:LIST MERCHANDISE ALL
LIST MERCHANDISE ALL 12:07:32 Jun 21 1999 1
MERCHANDISE
55040
51090
11020
...
```

When you copy the dictionary records, be sure to specify the DICT parameter with the target file name in the COPY statement. If you do not, UniData copies the dictionary records into the data portion of the file. The following example illustrates the results of a COPY statement when the DICT parameter with the target file name was not specified. It also illustrates copying dictionary records from the original file to the new file.

```
:COPY FROM DICT INVENTORY TO MERCHANDISE ALL
18 records copied
:LIST MERCHANDISE
LIST MERCHANDISE 10:29:29 May 29 1999 1
MERCHANDISE
PROD_NAME
13004
54030
40014
52060
40015
13005
36000
13006
50090
51040
DIFF
11110
INV_DATE
.
.
.
:COPY FROM DICT INVENTORY TO DICT MERCHANDISE ALL
@ID exists in MERCHANDISE, cannot overwrite
17 records copied
:LIST MERCHANDISE
LIST MERCHANDISE 10:32:29 May 29 1999 1
MERCHANDISE
52060
40015
13005
36000
13006
50090
51040
11110
...
```

ECLTYPE U Examples

In the following example, UniData copies a dictionary record (INV_DATE) to a new name (MORE_INV) in the same file. Notice that the TO keyword does not appear on the command line. It is not necessary, since the record is being copied from the source file to the source file.

```
:COPY FROM DICT INVENTORY INV_DATE, MORE_INV
1 records copied
:
```

The next example copies a dictionary record to a different file and gives it a new name. In this example, the TO keyword is required, since the target file differs from the source file.

```
:COPY FROM DICT INVENTORY TO DICT ORDERS PROD_NAME, ITEM_NAME
1 records copied
:
```

The next example demonstrates use of the SQUAWK keyword to display informational messages to the terminal during the copy process. The OVERWRITING keyword overwrites existing records of the same name without user verification:

```
:COPY FROM CLIENTS TO ORDERS 10011, C-10011 10013, C-10013 10015,
C-10015 OVER-WRITING
SQUAWK
10011 copied to C-10011
10013 copied to C-10013
10015 copied to C-10015
3 records copied
:
```

The following example copies ORDERS record 838 to record 10001:

```
:COPY FROM ORDERS 838, 1000
1 records copied
```

ECLTYPE P Examples

The following example illustrates a simple COPY statement. UniData makes a second copy of a record in the CLIENTS demo file:

```
:COPY CLIENTS 9999
TO: X-9999
1 records copied
:
```

In the next example, UniData copies a record from CLIENTS demo file to the ORDERS demo file. Notice UniData prompts for the target file name with TO:, and that the user proceeds the file name with a left parenthesis.

```
:COPY CLIENTS 10011
TO: (ORDERS
1 records copied
:
```

The next example shows a COPY statement that copies the dictionary record from the CLIENTS file to the dictionary of ORDERS file. The new dictionary record is called DISTRIBUTION.

```
:COPY DICT CLIENTS ZIP_CODE
TO: (DICT ORDERS DISTRIBUTION
1 records copied
:
```

In ECLTYPE P, you can display the all of the records in a file to the terminal by including an asterisk (*) and pressing ENTER at the TO: prompt, as shown in the following example:

```
COPY CLIENTS *
TO:
9999:
Paul
Castiglione
Chez Paul
45, reu de Rivoli
Paris
75008
France
3342425544y3342664857
WorkyFax
10034:
Fredrick
Anderson
Otis Concrete
854, reu de Rivoli
Paris
...
```

CREATE.FILE

Syntax

CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR] *filename* [,*subfile*] [*modulo* [,*block.size.multiplier*]] [TYPE *hashtype*] [DYNAMIC [KEYONLY | KEYDATA] [PARTTBL *part_tbl*]] [RECOVERABLE] [OVERFLOW]

Note: The PARTTBL and RECOVERABLE options are available on UniData for UNIX only.

Synonym

CREATE-FILE

Description

The ECL **CREATE.FILE** command creates a UniData file. If you do not indicate the kind of file to create (such as dictionary, data, or directory), UniData creates filename (both the data and dictionary files) as a static hashed file. If an operating system-level file of the same name already exists in the target account, CREATE.FILE fails.

See *Administering UniData* for more information on UniData file types, such as multifiles and part files.

Tip: The name you choose for a file must not exceed the length supported by your operating system. To view your operating system limitation, execute the ECL LIMIT command. The maximum operating system file name limit is the value of U_MAXFNAME. After you create the file, you can create a longer synonym in your VOC file to be used in UniData. For information about creating file name synonyms, see SETFILE.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
block.size.multiplier	The size, expressed as a multiplier, of each group in a hashed file. If you specify a block size multiplier of 0, UniData creates 512-byte groups. A block size multiplier of 1 represents 1024 bytes, 2 represents 2048 bytes, and so on. The maximum block size multiplier is 16. See "Estimating the File Size" in this section. If you specify a block size multiplier greater than 16, 16 is used.
filename	The name of the UniData file to be created.
hashtype	UniData supports two proprietary hashing algorithms (hash type 0 and hash type 1), which determine what data groups contain each record. The default hash type for both static files and dynamic files is 0. See *Administering UniData* for more information about the UniData hashing algorithms.
modulo	Number of groups allocated to *filename*. When hash type is 0, modulo must be a prime number. If the number you choose is not prime, UniData automatically increases the number to the nearest prime number. See "Estimating the Modulo" in this section.
part_tbl	The path and file name for a UNIX text file to be used as the part table for a dynamic hashed file. UniData copies the part table into the directory with the dynamic file. This option is only supported on UniData for UNIX. **Note**: UniData distributes part files across file systems by using ASCII files called part tables.
,subfile	Name of a subfile to be created when you use the MULTIFILE or MULTIDIR options. You must separate *filename* and *subfile* with a comma.
DATA	Creates only the data portion of *filename*.
DICT	Creates only the dictionary portion of *filename*. All UniData dictionary files are static hashed files. UniData prefixes dictionary file names with D_.

CREATE.FILE Parameters

Parameter	Description
DIR	Creates a file whose data portion is a directory, rather than a UniData hashed file. Records in a DIR-type data file are text and data files. **Note**: The DYNAMIC, KEYONLY, KEYDATA, PARTTBL, TYPE, and RECOVERABLE keywords are invalid for a DIR-type file.
DYNAMIC	Creates a dynamic hashed file. Dynamic files resize based on split and merge parameters. For more information on UniData dynamic files, see *Using UniData* or *Administering UniData*.
KEYONLY	Used only with the DYNAMIC keyword. Set the split / merge type for a dynamic file to KEYONLY, meaning that the load factor in each group is based on keys and pointers only. This is the default split / merge type.
KEYDATA	Used only with the DYNAMIC keyword. Set the split / merge type for a dynamic file to KEYDATA, meaning that the load factor in each group is based on keys and pointers plus data. For more information about split / merge types, see "Special Considerations for Dynamic Files" in this section.
MULTIDIR	Creates a multilevel directory file, consisting of multiple DIR-type files (*subfile*) under a directory (*filename*). The VOC entry for a MULTIDIR file is type LD.
MULTIFILE	Creates multiple DATA-type hashed files (*subfile*) under a directory (*filename*). The VOC entry is type LF. If you do not specify a subfile name, UniData creates a hashed file and names both it and the directory *filename*.
PARTTBL	Used only with the DYNAMIC keyword. Copies the specified text file (*part_tbl*) into the dynamic file directory. The text file you specify with the PARTTBL option must exist. The contents of this file are copied into the dynamic file directory in a file named parttbl. This option is supported on UniData for UNIX only.

CREATE.FILE Parameters (continued)

Parameter	Description
RECOVERABLE	Creates a recoverable file. You can define only the following types of files as recoverable: ■ Static hashed file or multilevel subfile ■ Dynamic hashed file or multilevel subfile This option is supported on UniData for UNIX only. For more information about recoverable files, see *Administering the Recoverable File System*.
TYPE *hashtype*	Hashing algorithm for the file. Hash type is 0 or 1. The default hash type for static and dynamic files is 0.
OVERFLOW	If specified, UniData creates a dynamic file with an overflow file for each dat file. For example, over001 corresponds to dat001, over002 corresponds to dat 002, and so forth. When the file is cleared, UniData maintains this overflow structure.

CREATE.FILE Parameters (continued)

Note: On UniData for UNIX, when you create a DIR, MULTIDIR, or MULTIFILE, UniData attempts to set permissions on the UNIX directory to 775 (rwxrwxr-x). These permissions allow users in the same UNIX group as the file owner add, modify, and delete records, subdirectories, and subfiles. UniData can set these permissions only if your umask allows. If your umask is more restrictive than 003, the umask rather than UniData determines the permissions setting for a DIR, MULTIDIR, or MULTIFILE.

Estimating the Modulo

UniData blocks a hashed file into a specific number of groups called the modulo. The best number of groups (modulo number) depends on variable factors, such as record size and length of the primary key. When you execute CREATE.FILE, the modulo and block size multiplier that you enter determine the size of the file. It is important to create a file that is adequate in size to store data efficiently. If you create a static file with only a few groups, the file can overflow quickly, which causes slow performance. When you create a dynamic hashed file, the modulo increases automatically when records are added to the file. However, you should still calculate the best initial modulo before you create the file. The following steps describe how to estimate a modulo number for a static hashed file (or initial modulo for a dynamic hashed file):

1. Estimate an average record size. The average record size (in bytes) is the sum of the size of the primary key, an estimated record size, and the integer 9. Suppose you're designing a file with the following characteristics:

   - Primary key Primary key is a 10-character field.
   - Estimated record length – There are 20 data attributes that are each 10 characters in length, for a record length of 200.

   Therefore, the average record size is: 10 + 200 + 9 = 219 bytes.

Note: If you are planning to resize an existing file or copy records from an existing file, you can use the FILE.STAT command (in ECLTYPE U) to display average number of bytes in a record and average number of bytes in a record ID. For an existing file, compute the average record size as the sum of the average number of bytes in the record, the standard deviation from average, the average number of bytes in the record ID, and 9 for overhead.

2. Compute the number of records per block as:

   (Block size in bytes - 32) / Average record size

   Note that the pointer array in each block requires 32 bytes. In the example, if you want to use 1024-byte blocks, then the number of records per block is (1024 -32) / 219, or 4.5.

3. Divide the number of records in the file by the number of records per block to compute the calculated modulo:

   1000 records / 4.5 records per block = 222 blocks

4. Add 10 –15% for optimum hashing, bringing the calculated modulo to 255.
5. Round this number up to the nearest prime number. This becomes the modulo for the file. For this example, the nearest prime number is 257. Use the ECL PRIMENUMBER command to find the prime number.

Estimating the File Size

UniData determines the size for a file by adding 1 to the modulo (for the group that contains the file header) and multiplying that sum by the block size.

Block size is the product of a block size multiplier (block.size.multiplier) times 1024. The block size multiplier is an integer between 0 and 16 inclusive. Except for 0, these integers represent multiples of 1,024 bytes. If you use 0 for block.size.multiplier, UniData interprets that as 512. If you use a number greater than 16, UniData uses 16K.

Note: A recoverable file must have a block size multiplier of at least 1 (1,024 bytes). A 512-byte block size is not supported.

For efficient I/O performance, we recommend that you use only the values of 0, 1, 2, 4, 8, and 16 for the block.size.multiplier. Do not use odd numbers for block sizes.

Special Considerations for Dynamic Files

If you are creating a dynamic hashed file, selecting an appropriate starting (minimum) modulo is critical to the future efficiency of the file. All subsequent splitting and merging operations are affected by the initial modulo. Starting with a modulo that is very small (for instance, 3) produces inefficient hashing and splitting as the file grows. Starting with a modulo that is very large produces a file that may take up more disk space than needed, but that impact is better than the slow performance and inefficiency that results if the starting modulo is too small.

When you create a dynamic file, estimate the initial modulo using the same procedure you would use to estimate the modulo for a static file.

KEYDATA Files and Block Size

If you are creating a KEYDATA dynamic file, make certain the block size is large with respect to the record length. We recommend that you choose a block size that is at least 10 times the average record length. Load factor in a KEYDATA file is based on the percentage of the space in each block that is occupied by both keys and data. If the block size is not large with respect to record size, the file will occupy a large amount of space and much of that space will be unused.

KEYONLY Files and Block Size

If you are creating a KEYONLY dynamic file, make certain the block size is large with respect to the average key length. We recommend that you choose a block size that is at least 10 times the average key length. Load factor in a KEYONLY file is based on the percentage of the space in each block that is occupied by keys and pointers. If the block size is not large with respect to the average key length and the hashing is not even, certain groups will be split over and over, resulting in an inefficient distribution of keys.

Example

In the following example, UniData creates a dynamic file. Notice the informational message related to modulo number. Also, notice that UniData creates both data and dictionary files, by default.

```
:CREATE.FILE CONTRACTS 4,2 DYNAMIC
4 is not a prime number, modulo changed to 5.
Create file D_CONTRACTS, modulo/1,blocksize/1024
Hash type = 0
Create dynamic file CONTRACTS, modulo/5,blocksize/2048
Hash type = 0
Split/Merge type = KEYONLY
Added “@ID”, the default record for UniData to DICT CONTRACTS.
:
```

Related Commands

CLEAR.FILE, DELETE.FILE

CREATE.INDEX

Syntax

CREATE.INDEX *filename attribute1* [*attributeM...attributeN*] [NO.DUPS] [NO.NULLS]

Synonym

CREATE-INDEX

Description

The ECL CREATE.INDEX command creates an index file for a UniData file and creates alternate key indexes for data attributes you indicate. The index file stores all of the alternate key indexes created on a file.

When you create alternate key indexes, you can screen out empty strings or duplicate values, or both (for nonrecoverable files).

If an alternate key index exists for the attribute you are indexing, UniData displays a message indicating that you cannot create more than one index for the same attribute (location).

UniData stores index files in two places:

- Static files – The UniData account directory. Static index files have a X_ prefix.
- Dynamic files – The UniData file directory. Dynamic index files are named idx001, idx002,....

The CREATE.INDEX command does not populate the alternate key index. To add keys to the index, use the ECL BUILD.INDEX command.

When CREATE.INDEX completes successfully, @SYSTEM.RETURN.CODE is set to the number of indexes created. If an error occurs, @SYSTEM.RETURN.CODE is set to -1.

IBM recommends that alternate key length be as large as the longest attribute being indexed to help prevent alternate key overflow. For example, if the indexed attribute is a virtual field that concatenates CITY (35 characters), STATE (2 characters), and ZIP (10 characters), the alternate key length should be 47.

Tip: Use the LIST.INDEX command to display a list of alternate key indexes for a UniData file.

Using Indexes Created in an Earlier Release

Keep the following in mind when upgrading or using an index that was created with an earlier release of UniData:

- When upgrading from a release earlier than 3.3, you need to rebuild indexes. UniData added a time stamp feature at Release 3.3.
- Indexes created at Release 4.1 of UniData for UNIX or Release 3.6 of UniData for Windows NT are not backwardly compatible. Beginning with these releases, indexes were no longer compressed.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of a UniData data file to be indexed.
attribute	Data attribute on which to base an alternate key index. You can name multiple data attributes to create multiple alternate key indexes simultaneously. You cannot create multiple alternate key indexes on the same location (*attribute*).
NO.DUPS	For nonrecoverable files, the parameter blocks creation of duplicate keys in an alternate key index. If UniData encounters duplicate data values when building the index or writing a record, the operation terminates. Here is a summary of the effect of NO.DUPS on other commands: BUILD.INDEX — If the nonrecoverable file contains duplicate values in the alternate key attribute, UniData displays an error message and does not build the index. UniData allows duplicates in indexes for RFS files. (UniBasic) WRITE / WRITEU / WRITEV / WRITEVU — For nonrecoverable files, the ON ERROR clause executes if you attempt to write a record that contains a duplicate alternate key value, and the STATUS return value is set to 10. For recoverable files, UniBasic writes the duplicate keys, but sets STATUS to 10 after the write.
NO.NULLS	Specifies that records that have an empty string as the alternate key **not** be included in an alternate key index. Key values that **are** the null value are included in indexes created with the NO.NULLS keyword specified and null value handling turned on.

CREATE.INDEX Parameters

Example

The following example creates an index file for the CLIENTS demo file and three alternate key indexes. When you create an index file, UniData prompts for an alternate key length. If you press ENTER instead of entering a key length, UniData uses the default (20).

```
:CREATE.INDEX CLIENTS LNAME COUNTRY ZIP_CODE
Alternate key length (default 20):
“LNAME” created
“COUNTRY” created
“ZIP_CODE” created
:
```

Related Commands

BUILD.INDEX, DELETE.INDEX, DISABLE.INDEX, ENABLE.INDEX, LIST.INDEX, UPDATE.INDEX

CREATE.TRIGGER

Syntax

CREATE.TRIGGER [DATA | DICT] *filename trigger* [BEFORE] {UPDATE |DELETE}

Synonym

CREATE-TRIGGER

Description

Use the ECL **CREATE.TRIGGER** command to place a trigger name in a file header. Depending on the kind of trigger, UniData references a UniBasic trigger subroutine with the trigger name whenever a user attempts to execute either update or delete operations on a file.

For detailed information about creating trigger subroutines, see *Developing UniBasic Applications*.

Note: To execute the CREATE.TRIGGER command, you must be the owner of the file at the operating system level or have root permissions on UniData for UNIX or Administrator permissions on UniData for Windows Platforms.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DATA	The data portion of a file.
DICT	The dictionary portion of a file.
filename	The name of the file that contains the header where the trigger name is inserted.
trigger	The name of the UniBasic trigger subroutine.
BEFORE	The type of trigger that UniData executes before processing an update or delete operation on the file.
UPDATE	The trigger related to updated operations on a file. A file header can reference only one UPDATE trigger.
DELETE	The trigger related to delete operations on a file. A file header can reference only one DELETE trigger.

CREATE.TRIGGER Parameters

Examples

The following example creates a BEFORE UPDATE trigger in the header of the INVENTORY file. The trigger calls the globally cataloged UniBasic trigger subroutine PRICE_UPDATE.

```
:CREATE.TRIGGER INVENTORY PRICE_UPDATE UPDATE
:
```

To find out if triggers are present in a file header, use the LIST.TRIGGER command. UniData indicates whether an UPDATE or DELETE trigger is defined and provides the trigger name:

```
:LIST.TRIGGER INVENTORY
BEFORE UPDATE TRIGGER: PRICE_UPDATE
BEFORE DELETE TRIGGER: not defined
:
```

Related Commands

DELETE.TRIGGER, LIST.TRIGGER

DATE

Syntax

DATE

Description

The ECL **DATE** command displays the current system date and time on the terminal screen.

Example

The following example displays the current system date and time.

```
:DATE
Wed Jul 30 10:20:50 MDT 1999
```

DATE.FORMAT

Syntax

DATE.FORMAT [2]

Synonym

DATE-FORMAT

Description

The ECL DATE.FORMAT command establishes the default display format for dates in output from ECL, UniQuery, and UniBasic statements for the current UniData session.

To reset the display to United States format, you must exit your current UniData session and open a new session.

This command has no effect on output from the DATE command.

Note: To display dates in all uppercase, set UDT.OPTIONS 4 ON.

The setting of UDT.OPTIONS 34 toggles the system date format between alphanumeric and numeric for the month display when you specify HEADING with the D option in a UniQuery statement. ON produces alphanumeric output. OFF produces numeric output. See the UDT.OPTIONS Commands Reference for more information about UDT.OPTIONS.

If you always want to display dates in the international format for all users, you can change the date value in the DEFAULTS record to 2. The DEFAULTS record is located in the language message file in udthome/sys on UniData for UNIX or udthome \ sys on UniData for Windows Platforms. The date value is the last value in attribute 1, and has a default setting of 0. For more information about the language message file, see UniData International.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no option	European: DD / MM / YY
2	International format: YY / MM / DD

DATE.FORMAT Parameters

Example

The following example executes DATE.FORMAT 2, and then a UniQuery statement that displays the system date in the header. Notice the international date format:

```
:DATE.FORMAT 2
:LIST INVENTORY QTY HEADING "'D'"
1999-07-30
INVENTORY. Quantity
10140 12000
149
13002 104
12006 396
11010 8781
3986
54090 575
...
```

dbpause

Syntax

dbpause

Description

dbpause is a UniData system-level command that blocks most updates to the database made in a UniData session. Any updates made from the operating system level are not blocked. You can use this feature to perform some tasks that normally require UniData to be stopped, such as backing up your data.

When the dbpause command is issued, all current writes and transactions complete before

UniData pauses. Updates are blocked until the system administrator executes the dbresume command.

System-level commands, such as cp or mv on UniData for UNIX or COPY or MOVE on UniData for Windows Platforms, are not blocked. In addition, updates to the _HOLD_ file and the _PH_ file are not blocked, and printing of reports is not interrupted.

If you execute dbpause when running the Recoverable File System (RFS), UniData forces a checkpoint, flushes the after image logs to the archive files (if archiving is enabled), and marks the next available logical sequence number in the archive file for use after the backup. UniData displays this information on the screen where you execute dbpause, and writes it to udtbin/sm.log.

Note: To execute the dbpause command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

For more information about dbpause, see *Administering UniData* and *Administering the Recoverable File System*.

Related Commands

dbpause_status, dbresume

dbpause_status

Syntax

dbpause_status

Description

The UniData system-level **dbpause_status** command returns information about the status of dbpause. If dbpause is in effect, dbpause_status returns the message DBpause is ON. If dbpause is not in effect, dbpause_status returns the message DBpause is OFF.

For more information about dbpause_status, see *Administering UniData* and *Administering the Recoverable File System*.

Related Commands

dbpause, dbresume

dbresume

Syntax

dbresume

Description

The dbresume system-level command resumes processing after the dbpause command is issued. When dbresume is executed, all writes that were blocked when dbpause was issued complete.

Note: You must log in as root on UniData for UNIX or Administrator on UniData for Windows Platforms to issue the dbresume command.

For more information about dbresume, see *Administering UniData* and *Administering the Recoverable File System*.

Related Commands

dbpause, dbresume

DEBUG.FLAG

Syntax

DEBUG.FLAG [ON | OFF]

Synonym

DEBUG-FLAG

Description

The ECL DEBUG.FLAG command enables the UniBasic DEBUG command. This flag is automatically on when UniData is installed.

For information about writing UniBasic programs, see *Developing UniBasic Applications*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
ON	Enables the UniBasic DEBUG command.
OFF	Suppresses the UniBasic DEBUG command.

DEBUG.FLAG Parameters

Example

The following program contains the UniBasic DEBUG command at line 002:

```
:AE BP convertit
Top of “convertit” in “BP”, 19 lines, 411 characters.
*--: p
001: PROMPT “”
002: DEBUG
003: LOOP
004: PRINT “Input or output [I/O]?” :
005: INPUT i_or_o
006: IF i_or_o = “” THEN STOP
...
019: END
Bottom.
```

As expected, when you execute this program, it exits to the debugger when this line executes:

```
:RUN BP convertit
***DEBUGGER called at line 2 of program BP/_convertit
```

If you execute DEBUG.FLAG OFF before running the program, the DEBUG command is ignored. Notice that the prompt is found on line 004, after the DEBUG command in the program displayed previously:

```
:DEBUG.FLAG OFF
:RUN BP convertit
Input or output [I/O]?
```

If we then turn the flag back on, the DEBUG command executes the next time we run the program:

```
:DEBUG.FLAG ON
:RUN BP convertit
***DEBUGGER called at line 2 of program BP/_convertit
```

DEBUGLINE.ATT

Syntax

DEBUGLINE.ATT

Synonym

DEBUGLINE-ATT

Description

The ECL **DEBUGLINE.ATT** command attaches a terminal for dual-terminal debugging with the UniBasic debugger. You must first initialize the communication line with SETDEBUGLINE.

For more information on UniBasic and the UniBasic debugger, see *Developing UniBasic Applications*.

Related Commands

DEBUGLINE.DET, SETDEBUGLINE, UNSETDEBUGLINE

DEBUGLINE.DET

Syntax

DEBUGLINE.DET

Synonym

DEBUGLINE-DET

Description

The ECL DEBUGLINE.DET command terminates dual-terminal debugging with UniBasic.

For more information on UniBasic and the UniBasic debugger, see *Developing UniBasic Applications*.

Related Commands

DEBUGLINE.ATT, SETDEBUGLINE, UNSETDEBUGLINE

DEFAULT.LOCKED.ACTION

Syntax

DEFAULT.LOCKED.ACTION [BELL [*interval*] | OFF]

Synonym

DEFAULT-LOCKED-ACTION

Description

The ECL **DEFAULT.LOCKED.ACTION** command turns on or off terminal beeping at intervals while the process waits for an exclusive file or record lock to be released.

Note: To avoid holding up a process when it encounters a lock, include the LOCKED clause in the UniBasic command that attempts to set an exclusive lock.

Some UniBasic commands that set exclusive locks include the following:

- READU
- READVU
- MATREADU
- RECORDLOCKU

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
BELL	Turns on the bell.
interval	The interval, in seconds, at which the bell sounds. The default is 10 seconds.
OFF	Turns off the bell.

DEFAULT.LOCKED.ACTION Parameters

Example

The following example sets the terminal bell to sound every 20 seconds when the process encounters a locked file or record:

```
:DEFAULT.LOCKED.ACTION BELL 20
```

DELETE

Syntax

DELETE [DICT] *filename* [*record_ID* [...]]

Description

The ECL DELETE command deletes one or more record IDs from a file. If you do not indicate a record ID, UniData steps through the file, prompting with each record key in turn.

You can execute this command against an active select list.

Warning: UniData deletes all data for record IDs listed in an active select list without prompting for confirmation.

UniData displays an informational message if unable to execute this command due to the presence of a DELETE trigger. For more information about UniData triggers, see *Using UniData.*

Note: UDT.OPTIONS 16 governs the kind of message that displays when you use an active select list to delete records. When this option is ON, UniData displays only the number of records deleted. When this option is OFF, UniData displays the record IDs, but not the number of records deleted.

Parameters

The following table lists the DELETE command parameters.

Parameter	Description
DICT	Deletes the dictionary. If you do not include the DICT keyword, UniData deletes records from the data file.
filename	File from which records are to be deleted.
record_ID	ID of a record to be deleted. Separate multiple record IDs with a space.

DELETE Parameters

Examples

In the following example, UniData deletes two records from the INVENTORY demo file:

```
:DELETE INVENTORY 31000 39300
‘31000’ deleted.
‘39300’ deleted.
:
```

In the next example, UniData prompts for a record to delete from the dictionary file of the INVENTORY demo file. You can enter only one record ID each time UniData prompts:

```
:DELETE DICT INVENTORY
Delete more records from file INVENTORY (Y/N)?Y
please type in key: INV_DATE
‘INV_DATE’ deleted from INVENTORY
Delete more records from file INVENTORY (Y/N)?N
```

In the next example, UniData deletes the records listed in an active select list. If you respond Y to the prompt UniData immediately deletes all records in the list.

```
:SELECT INVENTORY WITH @ID LIKE “5...”
83 records selected to list 0.
>DELETE INVENTORY
Do you want to delete records in select list?(Y/N)Y
‘56060’ deleted.
‘57030’ deleted.
‘53040’ deleted.
‘56070’ deleted.
‘55040’ deleted.
```

DELETECOMMON

Syntax

DELETECOMMON [/ *common.name* /]

Description

The ECL DELETECOMMON command deletes one or all named common areas. If you do not specify *common.name*, all named common areas are deleted.

If control returns to a UniBasic program after execution of DELETECOMMON, or if the specified common area does not exist, UniData displays a warning message and does not delete common.

Note: The UniBasic named common areas store variables that can be accessed from any subroutine or program. For information on declaring and using named common areas, see Developing UniBasic Applications, or COMMON in the UniBasic Commands Reference.

Examples of allowed and disallowed processes.

Allowed	Not Allowed
A user executes DELETECOMMON from the ECL prompt.	1. A UniBasic program 2. EXECUTEs DELETECOMMON
1. Paragraph or Proc 2. Executes DELETECOMMON	1. A UniBasic program. 2. EXECUTEs a paragraph or Proc 3. that executes DELETECOMMON
1. A UniBasic program 2. CHAINs to a paragraph or Proc 3. that executes DELETECOMMON	1. A UniBasic program 2. CHAINs to a UniBasic program 3. CHAINS to another UniBasic program 4. that EXECUTEs DELETECOMMON

Allowed and Disallowed Processes

Allowed	Not Allowed
1. A UniBasic program 2. CHAINs to a UniBasic program 3. that CHAINs to another UniBasic program 4. that CHAINs to a paragraph or Proc 5. that executes DELETECOMMON	1. A Paragraph or Proc 2. runs a UniBasic program 3. that EXECUTEs DELETECOMMON
1. A Proc or paragraph 2. runs a UniBasic program 3. that CHAINs to a paragraph or Proc 5. that executes DELETECOMMON	
UDT.OPTIONS 40 ON: 1. A Proc or paragraph 2. runs a UniBasic program 3. that EXECUTE a UniBasic program 4. that CHAINs to a paragraph or Proc 5. that executes DELETECOMMON	UDT.OPTIONS 40 OFF: 1. A Proc or paragraph 2. runs a UniBasic program 3. that EXECUTE a UniBasic program 4. that CHAINs to a paragraph or Proc 5. that executes DELETECOMMON

Allowed and Disallowed Processes (continued)

Example

The following example demonstrates passing and deleting named common. These two programs pass the variable VAR in the named common COMVAR.

Note: Named common remains in memory until deleted.

```
FIRST_PROG
COMMON /COMVAR/ VAR
VAR = VAR+1
PRINT “IN FIRST_PROG”
PRINT VAR
CALL NEXT_PROG
Program Example
NEXT_PROG
*Program NEXT_PROG
COMMON /COMVAR/ VAR
PRINT “IN NEXT_PROG”
VAR = VAR+1
PRINT VAR
```

Here is the output from these programs (the first time you execute FIRST_PROG):

```
:RUN BP FIRST_PROG
IN FIRST_PROG
1
IN NEXT_PROG
2
```

VAR remains in the named common area COMVAR, which remains in memory, getting incremented by two each time you execute FIRST_PROGRAM, or once each time you execute NEXT_PROG until you execute DELETECOMMON or until the operating system is rebooted. Here we execute FIRST_PROG a second time, execute DELETECOMMON, then execute FIRST_PROG a third time. Only after executing DELETECOMMON is VAR reset to 0.

```
:RUN BP FIRST_PROG
IN FIRST_PROG
3
IN NEXT_PROG
4
:DELETECOMMON
:RUN BP FIRST_PROG
IN FIRST_PROG
1
IN NEXT_PROG
2
```

DELETE.CATALOG

Syntax

DELETE.CATALOG *program*

Synonyms

DECATALOG, DELETE-CATALOG

Description

The ECL **DELETE.CATALOG** command deletes the object code and removes the VOC record for the program from the CTLG subdirectory in which it is cataloged.

Note: DECATALOG works only in ECLTYPE P.

Even though you delete a cataloged program, as long as the program resides in the DIR file in which it was created, you can run it from the UniData ECL prompt with the RUN command. It cannot, however, be called with a UniBasic external call.

If a program is cataloged locally **and** globally, you must execute DELETE.CATALOG once for each entry. UniData deletes the local program first.

UniData places a copy of globally cataloged programs in shared memory for all users to access. Therefore, when you delete the object code and the VOC entry with this command, users who may be running the program from shared memory are not affected.

UniData stores locally cataloged programs in the CTLG directory of the local account. UniData stores globally cataloged programs in a subdirectory of the CTLG directory in *udthome*/sys on UniData for UNIX or *udthome*\sys on UniData for Windows Platforms. For more information about programming in UniBasic, see *Developing UniBasic Applications.*Formore information about cataloging and shared memory, see *Administering UniData.*

Examples

The following examples are taken from UniData for UNIX. On UniData for Windows Platforms, the path contains backslashes rather than forward slashes.

The first example shows the VOC file pointer for a UniBasic program called PRICE_UPDATE, which has been locally and globally cataloged. When you catalog a program locally, UniData creates the VOC pointer:

```
:CT VOC PRICE_UPDATE
VOC:
PRICE_UPDATE:
C
/users/claireg/demo/CTLG/PRICE_UPDATE
BP PRICE_UPDATE
:
```

The next example shows the entries in the local and global catalogs for PRICE_UPDATE:

```
:!pwd
/users/claireg/demo
:
:LS CTLG
LS CTLG
PRICE_UPDATE
:
:!ls $UDTHOME/sys/CTLG/p
!ls $UDTHOME/sys/CTLG/p

PRICE_UPDATE
:
```

The next example deletes the catalog entries and the VOC pointer with the DELETE.CATALOG command. After UniData deletes the object code from the catalogs, this program is no longer available for subroutine calls or direct execution as a cataloged item.

```
:DELETE.CATALOG PRICE_UPDATE
:
:LS CTLG
:
:DELETE.CATALOG PRICE_UPDATE
:
:!ls $UDTHOME/sys/CTLG/p
:
:CT VOC PRICE_UPDATE
VOC:
PRICE_UPDATE is not a record in VOC.
:
```

DELETE.FILE

Syntax

DELETE.FILE [DATA] [DICT] *filename* [,*filename2*] [FORCE]

Synonym

DELETE-FILE

Description

The ECL DELETE.FILE command deletes a UniData file and all records in it. If you do not indicate DATA or DICT, UniData deletes both. If the file is multilevel, UniData deletes all part files unless you stipulate filename2.

Note: UDT.OPTIONS 87 determines what UniData deletes when you execute DELETE.FILE against a file in a remote account. If UDT.OPTIONS 87 is on, UniData deletes the file pointer in the current directory and the file in the remote account. If UDT.OPTIONS 87 is off, UniData deletes only the VOC entry that points to the file.

You must have appropriate permissions to delete a UniData file.

Warning: You cannot use system-level commands (such as cp, rm,andtar) to operate on UniData recoverable files when UniData is running. If you use these commands on recoverable files, you could corrupt your data.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DATA	Deletes only the data file.
DICT	Deletes only the dictionary file.
filename	The name of the file to be deleted.
filename2	The multifile subdirectory to be deleted if *filename* is a multilevel file. UniData does not delete other LD or LF type files within *filename*.
FORCE	Deletes the file without prompting for confirmation.

DELETE.FILE Parameters

Examples

The following example deletes both the data and dictionary files of the CLIENTS demo file. Notice that UniData prompts before deleting the file.

```
:DELETE.FILE CLIENTS
Do you really want to delete file CLIENTS?(Y/N):Y
Deleting file D_CLIENTS.
Deleting file CLIENTS.
:
```

The next example displays a VOC pointer to the INVENTORY file in the demo directory on UniData for UNIX. Then DELETE.FILE deletes the VOC file pointer.

```
:CT VOC inventory
VOC:
inventory:
F
/disk1/ud60/demo/INVENTORY
/disk1/ud60/demo/D_INVENTORY
:DELETE.FILE inventory
inventory is a synonym, the real data file name is
/disk1/ud60/demo/INVENTORY
inventory has a synonym dict file, The real dict file is
/disk1/ud60/demo/D_INVENTORY
:CT VOC inventory
VOC:
inventory is not a record in VOC.
:
```

Related Commands

CLEAR.FILE, CREATE.FILE

DELETE.INDEX

Syntax

DELETE.INDEX *filename* {*attribute* [*attributeM...attributeN*] | ALL}

Synonym

DELETE-INDEX

Description

The ECL DELETE.INDEX command deletes an alternate key index from an index file. You can delete multiple indexes simultaneously.

If DELETE.INDEX executes successfully, UniData sets @SYSTEM.RETURN.CODE to the number of indexes deleted. If an error occurs, UniData sets @SYSTEM.RETURN.CODE to -1.

DELETE.INDEX fails if the index has been disabled (with DISABLE.INDEX).

Tip: Occasionally index files can become corrupted due to hardware or software failures. In these cases, we recommend that you use the ALL option with DELETE.INDEX to delete the index file and all alternate key indexes, and then rebuild the index file and the alternate key indexes.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of the data file that contains an index file.
attribute	The name of the alternate key index. You can name as many alternate key indexes as you want.
ALL	Deletes all alternate key indexes from an index file and deletes the index file itself. If the index is in an overflowed state, you can delete it completely with the ALL keyword, then re-create the index file with CREATE.INDEX. This allows UniData to prompt for a key length, at which point you can assign a longer key length.

DELETE.INDEX Parameters

Example

The following example removes all alternate key indexes in the CLIENTS demo file:

```
:DELETE.INDEX CLIENTS LNAME COUNTRY ZIP
"LNAME" deleted
"COUNTRY" deleted
"ZIP" deleted
:LIST.INDEX CLIENTS
No indices created on file "CLIENTS"
:
```

For more information and creating, building, and deleting indexes, see *Using UniData*.

Related Commands

BUILD.INDEX, CREATE.INDEX, DISABLE.INDEX, ENABLE.INDEX, LIST.INDEX, UPDATE.INDEX

DELETE.TRIGGER

Syntax

DELETE.TRIGGER *filename* [DATA | DICT] [BEFORE] {UPDATE | DELETE}

Synonym

DELETE-TRIGGER

Description

The ECL **DELETE.TRIGGER** command deletes a trigger name from a file header.

For more information about triggers, see *Using UniData* or *Developing UniBasic Applications*.

Note: To delete a trigger, you must be the owner of the file at the operating system level, or you must log in as root on UniData for UNIX or Administrator on UniData for Windows Platforms.

Parameters

The following table lists the parameters for the DELETE.TRIGGER command.

Parameter	Description
DATA	Deletes a trigger associated with a data file.
DICT	Deletes a trigger associated with a dictionary file.
filename	The name of the file from which the trigger is to be deleted.

DELETE.TRIGGER Parameters

Parameter	Description
BEFORE	UniData executes the trigger subroutine before processing an update or delete operation on the file.
UPDATE	Deletes an UPDATE trigger.
DELETE	Deletes a DELETE trigger.

DELETE.TRIGGER Parameters (continued)

Example

The following example creates, lists, and deletes a trigger on the ORDERS demo file:

```
:CREATE.TRIGGER ORDERS DEMO_RTN BEFORE UPDATE
:LIST.TRIGGER ORDERS
BEFORE UPDATE TRIGGER: DEMO_RTN
BEFORE DELETE TRIGGER: not defined
:DELETE.TRIGGER ORDERS UPDATE
:LIST.TRIGGER ORDERS
BEFORE UPDATE TRIGGER: not defined
BEFORE DELETE TRIGGER: not defined
:
```

Related Commands

CREATE.TRIGGER, LIST.TRIGGER

deleteuser

Syntax

deleteuser *pid*

Description

The system-level **deleteuser** command deletes a process, removing its identification number (pid) from the active UniData user list, and freeing up a UniData license. This command sends a signal to the process requesting that the process terminate in an orderly manner, then waits for five seconds to see if the process was terminated. If the process is still active, deleteuser forces immediate termination of the process.

deleteuser can be helpful to clean up orphaned processes after a system crash or when an active process aborts.

Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL prompt.

Warning: Killing a process that may be accessing a file may cause file corruption. Forcing a process to terminate interrupts writes in progress.

Note: To execute the deleteuser command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Example

The following example lists and identifies user processes with the LISTUSER command, then deletes user process 1976. The pid is found in USRNBR column (second column).

```
# listuser
Max Number of Users UDT SQL TOTAL
~~~~~~~~~~~~~~~~~~~ ~~~ ~~~ ~~~~~
32 2 0 2
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
1 1913 1283 carolw udt pts/1 17:01:14 Jul 30 1999
2 1976 1283 carolw udt pts/4 17:35:15 Jul 30 1999
# deleteuser 1913
# listuser
Max Number of Users UDT SQL TOTAL
~~~~~~~~~~~~~~~~~~~ ~~~ ~~~ ~~~~~
32 1 0 1
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
2 1976 1283 carolw udt pts/4 17:35:15 Jul 30 1999
#
```

Related Command

LISTUSER

DISABLE.INDEX

Syntax

DISABLE.INDEX *filename*

Synonym

DISABLE-INDEX

Description

The ECL **DISABLE.INDEX** command blocks automatic updating of alternate key indexes. When automatic updating is disabled, UniData writes updates to a log file. You must then execute ENABLE.INDEX to reactivate the index. This applies updates to RFS files. For non-RFS files, you must also execute UPDATE.INDEX to apply the updates.

If a data file is being accessed when you execute DISABLE.INDEX, UniData continues to update the alternate key indexes until the file is closed.

The index log file for static files is x_*filename* on UniData for UNIX and L_FILENAME on UniData for Windows Platforms. The files are located in the current account. The index log file for dynamic files is xlog001, xlog002, and so forth. The log files are located in the dynamic file directory, rather than the account.

Note: Depending on the number and size of alternate key indexes, automatic index updating may slow system performance.

Example

The following example disables automatic index updating for the CLIENTS demo file:

```
:DISABLE.INDEX CLIENTS
Automatic Updates have been disabled for CLIENTS
:
```

Related Commands

BUILD.INDEX, CREATE.INDEX, DELETE.INDEX, ENABLE.INDEX, LIST.INDEX

DISABLE.USERSTATS

Syntax

DISABLE.USERSTATS

Description

The **DISABLE.USERSTATS** command discontinues collection of statistics for a UniData session.

DTX

Syntax

DTX *decimal.number*

Description

The ECL **DTX** command translates a decimal number to its equivalent hexadecimal value. DTX performs the inverse operation of the XTD command. If you input invalid characters, DTX returns 0.

Valid decimal values range from -2,147,483,647 to 2,147,483,647. Hexadecimal values ranging from 80000001 (-2,147,483,647) to FFFFFFFF (-1) are negative.

Example

In the following example, the DTX command translates the decimal numbers to their equivalent hexadecimal value:

```
:DTX 2738
AB2
:DTX -2121
FFFFF7B7
:DTX 1996
7CC
:
```

Related Command

XTD

dumpgroup

Syntax

dumpgroup *filename group* [-d*outputfile*][-p]

Description

The system-level **dumpgroup** command extracts readable records from a specified group in a UniData file. If the file was corrupted, dumpgroup unloads only the complete, valid records, leaving behind any information it cannot read.

If you execute dumpgroup without specifying an output file, the output simply displays on the screen. You will not be able to use that output to verify records or repair the damaged group. If you do specify an output file, dumpgroup extracts readable records in uneditable form, suitable for reloading. dumpgroup also creates a directory in the /tmp directory on UniData for UNIX or the \TEMP directory on UniData for Windows Platforms for each dumped group. The directory is named FILE_GROUP, where FILE and GROUP are the file name and group number you specified. This directory contains an ASCII file for each record, so that you can check them for consistency before reloading the damaged file.

For more information about how to use dumpgroup to recover files, see *Administering UniData.*

Use this command at the system prompt, or use the ECL! (bang) command to execute this command from the ECL prompt.

Warning: When you use the -d parameter, make sure you name your output file with a name that does not already exist in your account name. If you specify a duplicate name, your data may be overwritten.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of the file that contains groups to be extracted.
group	Number of the group to be dumped. **Tip** – The output from guide and verify2 identifies damaged groups.
-d*outputfile*	Directs output to *outputfile*. Output file that contains the readable records from the dumped group. You cannot edit this file. If you do not include -d, dumpgroup displays readable records on the display screen. Do not insert a space between -d and *outputfile*. **Warning** – Make sure *outputfile* is not the name of another item in your account. If it is, UniData will overwrite it. **Tip** – This file is the input file for the fixgroup command.
-p	Converts nonprinting field markers to printable characters in output file. Makes *outputfile* editable. This option is valid only with -d.

dumpgroup Parameters

Related Commands

fixfile, fixgroup, guide, verify2

DUP.STATUS

Syntax

DUP.STATUS [ON | OFF]

Description

The ECL **DUP.STATUS** command turns on or off the UniBasic checking for duplicate alternate index keys when reading or writing records. The setting of DUP.STATUS affects only files for which an alternate key index exists.

DUP.STATUS with no option returns the current setting: ON or OFF.

With DUP.STATUS ON, the following commands set the UniBasic STATUS function return value to 10 when one of the following commands reads or writes a duplicate alternate index key:

- WRITE, WRITEU, WRITEV, WRITEVU
- READFWD, READFWDL, READFWDU
- READBCK, READBCKL, READBCKL

With DUP.STATUS turned off, the return value of the UniBasic STATUS function returns 0 after successful execution of the preceding commands, regardless of the presence or absence of duplicate alternate key values.

Note: When you create an index, you can specify NO.DUPS to prevent UniData from creating duplicate values in the alternate key index of a nonrecoverable file. This blocks completion of the ECL BUILD.INDEX command and all UniBasic write commands when they would result in duplicate values being written to the alternate key index.

Examples

The following program writes duplicate alternate key values to the index LNAME. With DUP.STATUS ON, the STATUS function returns 10 after the WRITE (see the WRITE command and STATUS function in bold typeface).

```
OPEN 'CLIENTS' TO clients ELSE PRINT "Open error"
SETINDEX 'LNAME', FIRST_ALT_KEY ON clients

LOOP
READFWD rec FROM clients THEN
   ID = @ID
   IF STATUS() = 10 THEN
    PRINT "Duplicate record ":ID:" ":rec<2>:", ":rec<3>
   END ELSE
    PRINT "NOT duplicate record ":ID:
    PRINT ",":rec<2>:",":rec<3>:" STATUS: ":STATUS()
    ID = ID + 1000
    WRITE rec TO clients,ID ON ERROR PRINT " STATUS: ":STATUS()
    PRINT "New record: ":ID:",":rec<2>:",":rec<3>:" STATUS:
":STATUS()
    READFWD rec FROM clients THEN CONTINUE
   END
 END ELSE EXIT
REPEAT
```

This program produces the following results with DUP.STATUS on:

```
:RUN BP DUPSTAT
NOT duplicate record 9968,Adams,United Hospital STATUS: 0
New record: 10968,Adams,United Hospital STATUS: 10
NOT duplicate record 10054,Alps,Weld Engineering STATUS: 0
New record: 11054,Alps,Weld Engineering STATUS: 10
NOT duplicate record 10034,Anderson,Otis Concrete STATUS: 0
New record: 11034,Anderson,Otis Concrete STATUS: 10
NOT duplicate record 10020,Andropolis,Calgary Aluminum STATUS: 0
New record: 11020,Andropolis,Calgary Aluminum STATUS: 10

NOT duplicate record 10008,Anitpoli,W Systems STATUS: 0
New record: 11008,Anitpoli,W Systems STATUS: 10
NOT duplicate record 9987,Asakawa,Pearl Security STATUS: 0
New record: 10987,Asakawa,Pearl Security STATUS: 10
NOT duplicate record 10074,Barry,Lyon Repair STATUS: 0
```

ECLTYPE

Syntax

ECLTYPE [P | U]

Description

The ECL command ECLTYPE determines the parser used to interpret UniData commands issued at the UniData colon (:) prompt.

If you enter the ECLTYPE without indicating P or U, UniData displays the setting for UDT.OPTIONS 2. When UDT.OPTIONS 2 is off, ECLTYPE is U. When it is on, ECLTYPE is P.

We recommend that you use ECLTYPE U. ECLTYPE P is available for backward compatibility with legacy Pick® databases.

Note: Another way to change ECLTYPE is to change the setting of UDT.OPTIONS 2. By default, UDT.OPTIONS 2 is off. See the UDT.OPTIONS Commands Reference for more information about UDT.OPTIONS.

The ECLTYPE command has no effect on UniBasic programs. The parser used to execute a UniBasic program is determined by the BASICTYPE in which the program is compiled. See the UniBasic $BASICTYPE command documentation for more information.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
P	UniData interprets commands consistent with the Pick® parser.
U	UniData interprets commands consistent with the UniData parser.

ECLTYPE Parameters

Example

In this example, UniData performs the following tasks:

- Displays the setting for UDT.OPTIONS 2 (OFF), indicating ECLTYPE U.
- Changes ECLTYPE to P.
- Displays the new setting for UDT.OPTIONS 2 (ON), which indicates ECLTYPE P.

```
:ECLTYPE
2 U_PSTYLEECL OFF
:
:ECLTYPE P
:
:ECLTYPE
2 U_PSTYLEECL ON
:
```

ED

Syntax

ED [DICT] *filename* [*record_ID*]

Description

The ECL **ED** command invokes the standard operating system editor supported by UniData. On UniData for UNIX, the default system editor is vi. On UniData for Windows Platforms, the default system editor is the MS-DOS editor. To select a system editor other than the default, set the environment variable UDT_EDIT or modify the VOC record ED. You can create and edit UniBasic programs, VOC records, and data and dictionary files with the system editor. The UniData interface to the operating system allows the system editor to work with active select lists and to interactively prompt for record IDs.

You can edit only one record at a time in a UniData hashed file or DIR-type file.

UniData displays a warning message if a trigger prevents record update or deletion. For more information on UniData triggers, see the CREATE.TRIGGER command in this manual or *Developing UniBasic Applications*.

Note: On UniData for Windows Platforms, the ED command invokes the MS-DOS editor. This editor requires a graphical user interface, and is therefore unusable in a Telnet session. If you log on to UniData via UDSerial or UDTelnet services and execute ED, UniData displays a message advising you to use AE.

Tip: To direct UniData to automatically invoke an editor other than the default when executing the ED command, set the UniData environment variable UDT_EDIT to the full path of the editor of your choice. On UniData for Windows Platforms, be aware that users logged in via the UDSerial or UDTelnet services will be unable to use ED unless you have purchased a third-party character-based editor. For more information on supported editors, see your operating system documentation.

Regarding UniData editors:

- The ECL AE command invokes the UniData Alternate Editor. You can use this line editor to edit UniData hashed files and UniBasic source programs.
- UniData supplies UniEntry for modifying UniData records.
- You can edit UniData hashed files and DIR-type files with any ASCII text editor. For more information on supported editors, see your operating system documentation. Be aware, though, of any changes or conversions the editor might make to files it opens.
- On UniData for UNIX, the ECL VI command invokes vi, the UNIX system V visual editor, from within UniData.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Indicates a UniData dictionary file.
filename	The name of the file to be edited. *filename* can be a hashed data file or a DIR-type file (such as _PH_ or _HOLD_).
record_ID	The primary key of the record within *filename* to be edited. If the item is not found, UniData creates a new record with this ID.

ED Parameters

UniData Delimiters

Before displaying a record through ED, UniData converts the UniData delimiters in hashed files (not DIR files) into symbols. The following table lists the symbols to which delimiters are converted.

Symbol	Delimiter Name	ASCII Character
}	Value mark	ASCII 253
|	Subvalue mark	ASCII 252

UniData Delimiters

During the ED session, you can use theses symbols to insert value and subvalue marks into a record. UniData converts the delimiters to the corresponding ASCII value when you save the edited record at the end of the session.

Examples

The following example retrieves an existing record in the INVENTORY demo file with the ED editor:

```
:ED INVENTORY
Please enter key: 52020
```

After the ID is entered, the user presses ENTER. UniData clears the screen and displays the record.

In the following example, taken from UniData on UNIX, the UniData environment variable UDT_EDIT was set so that the ED command invokes the system editor vi.

```
10236
28560
Printer
9 Pin Dot Matrix
Gray
56
19999
30
~
...
"/tmp/__ED7267" 8 lines, 54 characters
```

ENABLE.INDEX

Syntax

ENABLE.INDEX *filename*

Description

The ECL **ENABLE.INDEX** command turns on automatic updating of alternate key indexes for a data file.

For nonrecoverable files, ENABLE.INDEX does not apply updates that were deferred as a result of the DISABLE.INDEX command. To apply them, execute ENABLE.INDEX followed by UPDATE.INDEX.

For recoverable files, ENABLE.INDEX automatically applies updates that were deferred as a result of the DISABLE.INDEX command, so you do not have to update their indexes with UPDATE.INDEX.

Warning: Execute UPDATE.INDEX on a nonrecoverable file immediately after executing ENABLE.INDEX to avoid data integrity problems.

Tip: You can display the current state of index updating with the LIST.INDEX command.

Examples

In the following example, the ENABLE.INDEX command turns on automatic index updating:

```
:ENABLE.INDEX CLIENTS
Automatic Updates have been enabled for CLIENTS
```

In the next example, LIST.INDEX is used to find out if an alternate key index has been updated. In line 8 of the report, “Index updates,” UniData reports that the alternate key indexes require updating, indicating that updates were made to records in the data file between the time when updates were deferred (as a result of the DISABLE.INDEX command) and the point when ENABLE.INDEX was executed.

```
:LIST.INDEX CLIENTS
Alternate Key Index Details for File CLIENTS Page 1
File.................. CLIENTS
Alternate key length.. 20
Node/Block size....... 4K
OV blocks............. 1 (1 in use, 0 overflowed)
Indices............... 4 (4 D-type)
Index updates......... Enabled, Indices require updating
Index-Name...... F-type K-type Built Empties Dups In-DICT S/M F-
no/VF-expr....
FNAME D Txt Yes Yes Yes Yes S 1
LNAME D Txt Yes Yes Yes Yes S 2
COUNTRY D Txt Yes Yes Yes Yes S 8
```

Related Commands

BUILD.INDEX, CREATE.INDEX, DELETE.INDEX, DISABLE.INDEX, LIST.INDEX, UPDATE.INDEX

ENABLE.USERSTATS

Syntax

ENABLE.USERSTATS

Description

The **ENABLE.USERSTATS** command begins collection of detailed statistics about the current UniData session. Each time you issue the command, UniData zeros all of the statistics for your process.

FILE.STAT

Syntax

FILE.STAT [DICT] *filename* [LPTR]

Synonym

FILE-STAT

Description

The ECL FILE.STAT command displays statistical information on a data file, including hash type, split / merge type (for dynamic files), block size, number of records, overflow status, record size, and total bytes used.

Note: The output from FILE.STAT differs depending on ECLTYPE.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Displays information about the dictionary portion of a file.
filename	The name of the file to be analyzed.
LPTR	Directs output to the printer instead of the display terminal.

FILE.STAT Parameters

Examples

The following example shows FILE.STAT output for the CLIENTS file in the demo database, in ECLTYPE P and in ECLTYPE U:

```
:ECLTYPE P
:FILE.STAT CLIENTS
15:51:48 Apr 28 1999
FILE MOD OV HTY ITEMS BYTES MNI/G MXI/G MNB/I MXB/I
CLIENTS 19 0 0 130 14452 6 8 93 140
------- ------- -------
19 130 14452
:
:ECLTYPE U
:FILE.STAT CLIENTS
File name = CLIENTS
Number of groups in file (modulo) = 19
Static hashing, hash type = 0
Block size = 1024
File has 1 groups in level one overflow.
Number of records = 130
Total number of bytes = 14452
...
```

In the next example, the convhash command changes CLIENTS to a dynamic file. Notice that FILE.STAT displays the hash type and also the split/merge type:

```
:ECLTYPE U
:!memresize CLIENTS DYNAMIC
Resize CLIENTS mod(,sep) = 0(,-1) type = -1 memory = 8000 (k)
dynamic
KEYONLY PARTTBL=DEFAULT
RESIZE file CLIENTS to 101.
134 record(s) in file.
CLIENTS RESIZED from 101 to 101
Total time used =1 (sec)
:FILE.STAT CLIENTS
File name(Dynamic File) = CLIENTS
Number of groups in file (modulo) = 101
Dynamic hashing, hash type = 0
Split/Merge type = KEYONLY
Block size = 1024
Number of records = 134
Total number of bytes = 14585
Average number of records per group = 1.3
Standard deviation from average = 0.6
Average number of bytes per group = 144.4
Standard deviation from average = 62.7
Average number of bytes in a record = 108.8
Average number of bytes in record ID = 5.8
Standard deviation from average = 16.1
Minimum number of bytes in a record = 14
Maximum number of bytes in a record = 140
Minimum number of fields in a record = 2
Maximum number of fields in a record = 16
Average number of fields per record = 9.9
Standard deviation from average = 1.0
:
```

Related Commands

ANALYZE.FILE, GROUP.STAT

FILELIMIT

Syntax

FILELIMIT

Description

The ECL FILELIMIT command displays the maximum file size, in blocks, that the current process can write.

Standard block sizes vary depending upon the host machine and the operating system version.

Tip: To determine the maximum modulo number for a UniData file, multiply the number of blocks by the standard block size (512) and divide by 2048 or by a block size supported by your operating system.

Example

In the following example, UniData displays the maximum file size available to create a new file on one particular installation:

```
:FILELIMIT
File size limit for this process is 4194304 blocks
:
```

FILEVER

Syntax

FILEVER [*filenameM...filenameN*]

filever [*filenameM...filenameN*]

Description

The ECL FILEVER command and the system-level filever command display the following information on UniData files:

- high-byte or low-byte (also provided by the system-level filever command)
- recoverable or nonrecoverable
- static or dynamic

filename is the name of a UniData file.

Example

The following example shows FILEVER output for three demo database files:

```
:FILEVER INVENTORY CLIENTS ORDERS
This machine is a high byte machine.
Recoverable INVENTORY is high byte machine 2.0 dynamic version.
Non-recoverable CLIENTS is high byte machine 2.0 static version.
Recoverable ORDERS is high byte machine 2.0 dynamic version.
```

Note: Although output from the filever command indicates if a file is recoverable or nonrecoverable, the Recoverable File System (RFS) is not supported on UniData for Windows Platforms. All files on UniData for Windows Platforms are treated as nonrecoverable.

fixfile

Syntax

fixfile {[-d*outputfile*]-f | -t | -k | -p]} [-m*messagefile*][-w*directory*][-i*inputfile* | *filename group*]

Description

The system-level fixfile command repairs a damaged group in a UniData file by extracting and reloading readable records.

fixfile with the -i option accepts as input a file created by the system-level guide command.

UniData operates differently depending on whether the file is static or dynamic, and whether one group is damaged or multiple groups are damaged. For detailed information about using fixfile to repair damaged groups, refer to *Administering UniData.*

To repair files, you must include the -d and -f options.

Warning: Do not let users access UniData files while fixfile is running you could lose records.

Before creating new output files, the guide utility renames all files it processes by appending a date. We recommend you remove the original (old) versions of these files after fixfile finishes running.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-d*outputfile*	For each readable record, UniData creates an ASCII file in a directory in the current UniData account. UniData also takes the following actions for static and dynamic files: Static files – Stores readable records in (uneditable) *outputfile*. Dynamic files – Stores readable records in (uneditable) *outputfile* and in a subdirectory in the /tmp directory named *filename_groupno* on UniData for UNIX, or in the \TEMP directory on UniData for Windows Platforms. **Note**: To repair files, you must include both the -f parameter (to clear the group) and the -d parameter (to restore readable records).
-f	Clears damaged groups. Must be combined with the -d or -t parameters.
-k	Does not clear records before reloading them, so that damaged records are retained in the file. Must be combined with the -d or -f parameters. ■ To copy readable records to another file, include the -k and the -d parameters. ■ To copy readable records to another file and return them to the file, include the -k, -d, and -f options.
-o[*filename*]	Stores output in *filename*. If *filename* is not specified, sends output to the standard output device. Default output device is the display terminal. Specify output device at the operating system level.
-p	Combine with the -d option to convert UniData delimiters and nonprinting characters in the ASCII files as follows: ■ Attribute mark – New line ■ Value mark – "}" ■ Subvalue mark – " | " ■ Text mark – "{" ■ Nonprinting – "."

fixfile Parameters

Parameter	Description
-t	Record key and the record length are reported for each readable record. Directs output to the terminal only. All attributes in the record are listed, indented by two spaces. In the display, UniData delimiters and nonprinting characters are represented as follows: ■ Attribute mark – New line ■ Value mark – "}" ■ Subvalue mark – " | " ■ Text mark – "{" ■ Nonprinting – "." **Note**: The -t and -d options are mutually exclusive.
-m*messagefile*	Writes error messages and statistics to *messagefile* instead of the terminal.
-w*directory*	Specifies directory for storing work files.
-i*inputfile*	The file containing names of files and groups to be repaired. *inputfile* is produced by the guide command. If you do not designate *inputfile* with guide, fixfile reads damaged file and group names from GUIDE_FIXUP.DAT in the current directory. The following describes the format of GUIDE_FIXUP.DAT: *filenameM* *group_num* ... *filenameN* *group_num* *group_num* *group_num* Note: -i*inputfile* and *filename group* are mutually exclusive.
filename	The name of the damaged file.
group	The number of the damaged group.

fixfile Parameters (continued)

How fixfile Works with Static Files

When you execute fixfile with the -t parameter against a static file, UniData displays the readable records from the file and group to the terminal. The group is not cleared or repaired. You can supply the names of the damaged files and groups from the command line or from an input file. The default input file is GUIDE_FIXUP.DAT, created if the guide utility detects damaged groups.

When you execute fixfile with the -d parameter on a static file, UniData creates:

- On UniData for Windows Platforms, an NTFS directory named FILE_dir, where FILE is the name of the static file. Each FILE_dir contains a subdirectory for each damaged group in FILE. The name of each subdirectory is the group number of the damaged group. Each subdirectory contains a text file for every readable record in the damaged group. Each filename is the key for the corresponding UniData record. These group records are in a format suitable for editing.
- A file, with the name you specified on the command line, containing the records fixfile could read in uneditable format. This file is used to reload the records into the damaged groups after the groups are cleared.

Note: If you specify the -p parameter, fixfile translates nonprinting characters in the records when it creates the editable files. Otherwise, only attribute marks are translated to new lines.

When you run fixfile with the -d and-f parameters against a static file, UniData reloads the records into the damaged groups, taking them from the file you specified on the command line. Unless you specify the -k parameter, fixfile clears the groups, removing all contents, before reloading the data. If you specify the -k parameter, UniData adds the records back, but does not clear any data from the group.

How fixfile Works with Dynamic Files on UniData for UNIX

When you execute fixfile with the -d option against a dynamic file, UniData creates the following:

- Each FILE_GROUP directory contains a text file for every readable record in the damaged group. Each records name is the key for the corresponding UniData record. These records are in a format suitable for editing.
- A file containing the records fixfile could read, in uneditable format suitable for reloading into the group after it has been cleared. This file is located in /tmp (or in the directory identified by the tmp environment variable) and is names ud_dp_pid. pid is the process ID of the process that executed fixfile.

When you execute fixfile with the -d and -f parameters against a dynamic file, UniData reads the file you specify with the -d parameter on the command line, and also reads the uneditable file of dumped records. UniData then reloads the records from that file into the damaged groups. Unless you specified the -k parameter, fixfile clears the groups, removing all contents, before reloading the data. Otherwise, UniData adds the records back, but does not clear any data from the group.

How fixfile Works with Dynamic Files on UniData for Windows Platforms

When you execute fixfile with the -d option against a dynamic file, UniData creates the following:

- An NTFS directory located in \TEMP for each file/group combination being repaired. The directories are named FILE_GROUP, where FILE is a damaged file (created from the guide utility) and GROUP is a damaged group. If several groups in a file are damaged, UniData creates a directory for each damaged group.
- Each FILE_GROUP directory contains a text file for every readable record in the damaged group. Each records name is the key for the corresponding UniData record. These records are in a format suitable for editing.
- A file containing the records fixfile could read, in uneditable format suitable for reloading into the group after it has been cleared. This file is located in \TEMP (or in the directory identified by the tmp environment variable) and is named ud_dp_pid. pid is the process ID of the process that executed fixfile.

When you execute fixfile with the -d and -f parameters against a dynamic file, UniData reads the file you specify with the -d parameter on the command line, and also reads the uneditable file of dumped records. UniData then reloads the records from that file into the damaged groups. Unless you specified the -k parameter, fixfile clears the groups, removing all contents, before reloading the data. Otherwise, UniData adds the records back, but does not clear any data from the group.

Examples

```
:!fixfile -ddump -f
Fixing dynamic file /usr/udt60/demo/INVENTORY, group 0
6 records dumped for group 0
The records can be found under directory /tmp//INVENTORY_0
Check them before fixing the file
1 block(including the group header) of group 0 was made empty
6 records written to file /usr/udt60/demo/INVENTORY.
```

In this case the user can look in the /tmp/INVENTORY_0 directory for copies of readable records. The file name suffix represents the group number from which the records were extracted. In this example, records were extracted from group 0. The user could compare this version of INVENTORY with recent backups to find out if records are missing in the new version.

After this execution of fixfile, guide reveals that the INVENTORY file is repaired.

```
:!guide INVENTORY -o
INVENTORY
Basic statistics:
File type.............................. Recoverable Dynamic
Hashing
File size
[dat001]............................... 20480
[over001].............................. 9216
File modulo............................ 19
File minimum modulo.................... 19
File split factor...................... 60
File merge factor...................... 40
File hash type......................... 1
File block size........................ 1024
File integrity:
No errors were found
Group count:
Number of level 1 overflow groups...... 8
Primary groups in level 1 overflow..... 8
Record count:
Total number of records................ 175
Average number of records per group..... 9.21
Standard deviation from average......... 3.58
Record length:
Average record length.................. 71.17
Standard deviation from average......... 18.25
Key length:
Average key length..................... 5.00
Standard deviation from average......... 0.00
Data size:
Average data size...................... 86.17
Standard deviation from average......... 18.25
Total data size........................ 15080

Predicted optimal size:
Records per block...................... 10
Percentage of near term growth......... 10
Scalar applied to calculation.......... 0.00
Block size............................. 1024
Modulo................................. 19
Files processed: 1
Errors encountered: 0
```

Related Commands

dumpgroup, fixgroup, guide, verify2

fixgroup

Syntax

fixgroup *filename group* [-i*inputfile*][-k]

Description

The system-level fixgroup command reloads a single hashed file group from the output file generated by the dumpgroup command.

Warning: If you run fixgroup without including an input file (using the -i parameter), UniData clears the damaged group and leaves it empty. Be sure that you have previously saved the readable records with the dumpgroup command. If you clear the damaged group and you have not saved the readable records, the data in that group is lost. The syntax for clearing a group without reloading it is:

> **fixgroup** *filename group*
>
> %fixgroup INVENTORY 5
>
> Fixgroup INVENTORY 5 will make group 5 empty,
>
> do you wish to do it? [y / n]

Execute this command at the system prompt, or use the ! (bang) command to execute this command at the ECL prompt.

Tip: Some types of file corruption (for example, file corruption that is not associated with a group number) can be repaired with the memresize command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of the file to be repaired.
group	The damaged group.
-i*inputfile*	Uses *inputfile* to replace *group*. *inputfile* is generated by the dumpgroup command. If you do not name an input file, UniData clears group without reloading it. **Note**: No space is allowed between -i and *inputfile*.
-k	Reloads damaged records from *inputfile* without clearing the group first. This option may be useful if the group has updated since dumpgroup was executed. **Tip**: Do not allow user access while a file is being repaired. We suggest that you clear damaged groups to ensure that damage is removed before reimporting records (in other words, do not use -k option) on the final executing of fixgroup.

fixgroup Parameters

Example

To prepare for this example, group 0 in the demo file INVENTORY was damaged. Then dumpgroup was executed to create the output file d_group. In this example, fixgroup first clears group 0, then copies repaired records from d_group into the group.

```
:!dumpgroup INVENTORY 0 -dd_group
6 records dumped for group 0
The records can be found under directory /tmp//INVENTORY_0
Check them before fixing the file
:!fixgroup INVENTORY 0 -id_group
1 block(including the group header) of group 0 was made empty
6 records written to file INVENTORY.
:
```

Related Commands

dumpgroup, fixfile, guide, verify2

fixtbl

Syntax

fixtbl [-fix]

Description

The system-level **fixtbl** command detects and optionally repairs certain error conditions that can affect dynamic files. Execute fixtbl from the UNIX prompt. This command is supported on UniData for UNIX only.

Note: fixtbl is an offline tool. If you attempt to execute fixtbl while UniData is running or paused, an error message displays and the command fails. This tool is intended for system administrators performing maintenance functions. It is not intended for end users.

When a dynamic file expands outside the file system where it was created, the part files are placed in a file system selected from a part table (a list of locations where the original file can expand). The original dynamic file directory contains UNIX symbolic links to the physical location of the data and overflow part files. In each file system where dynamic files expand, UniData maintains a UNIX hidden file called .fil_prefix_tbl that relates part file names back to their original dynamic file and account. The symbolic links may become out of sync with.fil_prefix_tbl if users manipulate dynamic part files with the UNIX mv, cp, or rm command. The fixtbl tool detects the following error conditions:

- .fil_prefix_tbl is missing. If a dynamic file directory contains links to another partition, but there is no .fil_prefix_tbl at that location, fixtbl can create a new one.
- A prefix in .fil_prefix_tbl references a different directory than the symbolic links from a dynamic file in the current account. fixtbl can select a new prefix, then move and relink the part files for consistency.

- There are symbolic links from a dynamic file to another partition, but there is no entry in the .fil_prefix_tbl that matches the links. Assuming the prefix in the links is not used by another directory, fixtbl can create an entry in .fil_prefix_tbl that is consistent with the links from dynamic files in the current account directory.

See *Administering UniData* for more information about part tables and per-file part tables.

Parameters

The behavior of fixtbl depends on whether you specify the optional parameter [-fix]. If you specify -fix, fixtbl creates or modifies the .fil_prefix_tbl in the target partition. Otherwise, fixtbl creates or modifies a working copy of .fil_prefix_tbl, called .fil_prefix_tbl.new. The following table summarizes the behavior of fixtbl with and without -fix.

Error Condition	fixtbl	fixtbl -fix
.fil_prefix_tbl missing	Creates / updates .fil_prefix_tbl.new.	Creates new .fil_prefix_tbl.
Naming inconsistency	Displays information messages on the screen.	Adds necessary entries to .fil_prefix_tbl; move and relink part files; display no messages.
Missing entry in .fil_prefix_tbl.	Creates / updates .fil_prefix_tbl.new.	Creates / updates .fil_prefix_tbl.

Behavior of fixtbl Command

Examples

The following examples show fixtbl output.

In the first example, there is a naming conflict between .fil_prefix_tbl and the symbolic links in the dynamic file directory:

```
% fixtbl
Creating new /tmp/partfiles/.fil_prefix_tbl.new file
Error: Problem entry in prefix table
/tmp/partfiles/.fil_prefix_tbl. Prefix AA
in /tmp/partfiles/.fil_prefix_tbl corresponds to /disk1/ud41/demo
but the dynamic file /home/terric/SAMPLE/SAMPLE_FILE/dat001 is
located in
/home/terric/SAMPLE. Please resolve the inconsistency.
Error: Problem entry in prefix table
/tmp/partfiles/.fil_prefix_tbl. Prefix AA
in /tmp/partfiles/.fil_prefix_tbl corresponds to /disk1/ud41/demo
but the dynamic file /home/terric/SAMPLE/SAMPLE_FILE/over001 is
located in
/home/terric/SAMPLE. Please resolve the inconsistency.
```

Notice that in the previous example fixtbl was run without the -fix option. Executing fixtbl -fix adds a new entry to .fil_prefix_tbl and moves and relinks the part files.

In the next example, the dynamic file contains links to /tmp/partfiles/BBSAMPLE_FILE3, but the prefix table does not match:

```
% fixtbl
Creating new /tmp/partfiles/.fil_prefix_tbl.new file
Error: File /home/terric/SAMPLE/SAMPLE_FILE3/dat001. Inconsistency
between
the symbolic link (/tmp/partfiles/BBSAMPLE_FILE3/dat001) and
/tmp/partfiles/.fil_prefix_tbl. Please locate the part file, and
either rename/relink it or change /tmp/partfiles/.fil_prefix_tbl.
Error: File /home/terric/SAMPLE/SAMPLE_FILE3/over001.
Inconsistency between
the symbolic link (/tmp/partfiles/BBSAMPLE_FILE3/over001) and
/tmp/partfiles/.fil_prefix_tbl. Please locate the part file, and
either rename/relink it or change /tmp/partfiles/.fil_prefix_tbl.
```

Notice that the -fix parameter was not used in the previous example, so updates were made to the working file .fil_prefix_tbl.new. Executing fixtbl with -fix moves and relinks the part files to resolve the inconsistency.

In the next example, a user attempts to execute fixtbl while the UniData daemons are running:

```
:!fixtbl
fixtbl has detected that the UniData daemons are running.
The system administrator must stop the daemons (with stopud)
before fixtbl can execute.
```

FLOAT.PRECISION

Syntax

FLOAT.PRECISION [0 | 1 | 2 | 3 | 4[,*round*]]

Synonym

FLOAT-PRECISION

Description

The ECL **FLOAT.PRECISION** command controls how UniData applies truncation and rounding for the following operations:

- Arithmetic calculations
- Display or printing (numbers are always converted from decimal to string)
- Comparisons
- UniBasic INT function

When you execute an arithmetic operation, UniData invokes the appropriate host operating system command, which performs the operation in floating point. When the results are converted to string format for print or display, the rounding that is automatically applied may produce unexpected results, so FLOAT.PRECISION provides a mechanism for controlling this conversion and rounding.

Points to Remember

FLOAT.PRECISION influences UniData in the following ways:

- Modifies operation of the UniBasic INT function based on the option you select:
  - 0 – UniData truncates all digits after the decimal point; no rounding occurs.
  - 1, 2, and 3 – UniData rounds numbers before converting them to integers.
  - 4[,round] – Arithmetic operations in UniBasic truncate results at the level of precision set by the UniBasic PRECISION function. round further refines this option.
- C internal double – UniData does not round the results of a C function that performs internal double calculation.

Note: The UniBasic PRECISION command sets the number of decimal places expressed for the current UniData session. The default is 4. For more information, see the *UniBasic Commands Reference*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no option	Displays the current FLOAT.PRECISION setting.
0	Default setting. UniData rounds numbers after conversion to string format and after comparisons are made.
1	UniData rounds results after each calculation or comparison.

FLOAT.PRECISION Parameters

Parameter	Description
2	UniData rounds numbers at these times: ■ After conversion to string format ■ After relational operations. ■ Before executing the UniBasic INT (integer) function.
3	UniData converts the results of calculations to integers (executes the UniBasic INT function). UniData rounds numbers before comparisons. ■ If PRECISION is set to 5 or less, UniData adds 1 to the eighth digit after the decimal point before rounding. ■ If PRECISION is set to a number greater than 5, UniData adds 1 to the digit two decimal places to the right of the precision setting before rounding. **Note**: See the example program run at the end of this section for an illustration.
4[,*round*]	Arithmetic operations in UniBasic truncate results at the level of precision set by the UniBasic PRECISION function. *round* further refines this option for compatibility with Pick®. The point at which the number is rounded is calculated as PRECISION + *round*. Default is 3. See "Rounding Before Truncating with FLOAT.PRECISION 4,*round*" following this table, for a complete description.

FLOAT.PRECISION Parameters (continued)

Rounding Before Truncating with FLOAT.PRECISION 4, round

Because of the way the operating system represents floating point numbers, FLOAT.PRECISION with option 4 may occasionally return unexpected results, especially for users accustomed to Pick ® processing. Therefore, you can specify round to round the number before truncation.

The point at which the number is rounded is calculated as PRECISION + round. The default is 3.

For example, when PRECISION is set to 1, and round is 3, UniData rounds up at the fourth position after the decimal point.

Here is another illustration: Because of the operating systems previously mentioned floating point representation, 4.7 may actually be represented internally as 4.699999999999. Because of this, FLOAT.PRECISION 1 causes UniBasic to return 4.6 rather than 4.7. Use FLOAT.PRECISION 4,*round* to correct this, as shown in the following examples:

PRECISION 1 and FLOAT.PRECISION 4, 4

```
rounding point =1 +4 =5
4.699999999999 + .00005 = 4.700049999999
```

truncates correctly to 4.7.

PRECISION 1 and FLOAT.PRECISION 4 (remember, round defaults to 3)

```
rounding point =1 +3 =4
4.699999999999 + .0005 = 4.700499999999
```

also truncates correctly to 4.7.

We recommend that you not specify a large number for round. In general, the operating system floating point calculations can handle a maximum of 14 significant digits, depending on your hardware and operating system. When you exceed this maximum, the rightmost digits in the results of any arithmetic calculations on the number are likely to be incorrect. The actual number of digits used by the operating system to truncate a number depends on the following:

d = I + MAX(F,(P+T))

- d – The number of digits used to truncate.
- I – The number of integer digits in the number.
- F – The number of fractional digits in the number.
- P – PRECISION.
- T – *round*.

Tip: If d exceeds the maximum number of significant digits supported by your operating system, truncation may be wrong. So, when I or PRECISION is large, keep round small.

Examples

If you execute FLOAT.PRECISION with no option, UniData returns the current settings, as shown in the following example:

```
:FLOAT.PRECISION 4,6
:FLOAT.PRECISION
FLOAT.PRECISION mode 4 , 6
```

The following UniBasic program requests the user to input a setting for PRECISION. Then the program performs some calculations and executes the UniBasic INT function.

```
PRINT ""
PRINT "Enter PRECISION: ";INPUT prec.var
PRECISION prec.var
PRINT "4/3*2 = ":4/3*2
PRINT "8/3*2 = ":8/3*2
PRINT "INT(2.999999999) = ":INT(2.999999999)
PRINT "INT(2.999995999) = ":INT(2.999995999)
IF 2.999995999=3 THEN PRINT "2.999995999 = 3"
ELSE PRINT "2.999995999 # 3"
IF 2.999999999=3 THEN PRINT "2.999999999 = 3"
ELSE PRINT "2.999999999 # 3"
END
```

The following sample executions of the preceding program demonstrate how different FLOAT.PRECISION and PRECISION settings affect results produced by arithmetic calculations and the UniBasic INT function.

```
:FLOAT.PRECISION 0
:RUN BP precision.test
Enter PRECISION:
?5
4/3*2 = 2.66667
8/3*2 = 5.33333
INT(2.999999999) = 2
INT(2.999995999) = 2
2.999995999 # 3
2.999999999 # 3
:FLOAT.PRECISION 1
:RUN BP precision.test
Enter PRECISION:
?5
4/3*2 = 2.66666
8/3*2 = 5.33334
INT(2.999999999) = 3
INT(2.999995999) = 3
2.999995999 # 3
2.999999999 # 3
:FLOAT.PRECISION 2
:RUN BP precision.test
Enter PRECISION:
?5
4/3*2 = 2.66667
8/3*2 = 5.33333
INT(2.999999999) = 3
INT(2.999995999) = 3
2.999995999 = 3
2.999999999 = 3
```

In this next execution, the result of applying the UniBasic INT function to 2.999995999 is 2 because UniData adds 1 to the eighth digit to the right of the decimal point, causing the number to be rounded to 2.999996. Then, UniData truncates all digits to the right of the decimal point in order to make the number an integer. However, the result of the same procedure against 2.99999999 is 3 because the addition of 1 to the eighth digit results in 3, which is an integer.

```
:FLOAT.PRECISION 3
:RUN BP precision.test
Enter PRECISION:
?5
4/3*2 = 2.66667
8/3*2 = 5.33333
INT(2.999999999) = 3
INT(2.999995999) = 2
2.999995999 # 3
2.999999999 = 3
```

The next two executions demonstrate use of FLOAT.PRECISION option 4: Compare the results of the first two operations in these executions to see that results of arithmetic operations are truncated at the level of precision set by the UniBasic PRECISION command.

Also, because PRECISION is applied before numbers are printed, option 4 causes 2.999995999 and 2.999999999 to be truncated to 2.99 in the last two operations, so the program selects the # (not equal to) symbol: 2.999995999 # 3 and 2.999999999 # 3.

```
:FLOAT.PRECISION 4
:RUN BP precision.test
Enter PRECISION:
?2
4/3*2 = 2.66
8/3*2 = 5.32
INT(2.999999999) = 2
INT(2.999995999) = 2
2.999995999 # 3
2.999999999 # 3
:RUN BP precision.test
Enter PRECISION:
?1
4/3*2 = 2.6
8/3*2 = 5.2
INT(2.999999999) = 2
INT(2.999995999) = 2
2.999995999 # 3
2.999999999 # 3
```

Related Commands

UniBasic

INT, PRECISION – For information, see the *UniBasic Commands Reference*.

forcecp

Syntax

forcecp

Description

The system-level forcecp command forces a Recoverable File System (RFS) checkpoint. A checkpoint flushes the system buffer and conducts other RFS-related activities. For more information about the recoverable file system, see *Administering the Recoverable File System*.

Execute this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL prompt.

Example

The following example illustrates the forcecp command from the ECL prompt:

```
:!forcecp
CheckPoint time before ForceCP: Wed Jun 30 15:11:20 1999
.CheckPoint time after ForceCP: Wed Jun 30 18:00:21 1999
.CP has been forced successfully.
CP has been forced successfully
```

GETUSER

Syntax

GETUSER

Description

The ECL **GETUSER** command displays the user number, name, and ID for the current UniData session:

- USER NUMBER – The UNIX or Windows NT process ID (pid). All UniData processes that are invoked in a single session use this pid.
- USER NAME – The login name for this process.
- USER ID – The ID for your login name assigned by UNIX or Windows NT.

Example

In the following example, UniData displays a user number, name, and ID:

```
:GETUSER
USER NUMBER=2000
USER NAME  =carolw
USER ID    =1283
```

Related Command

LISTUSER

GROUP.STAT

Syntax

GROUP.STAT [DICT] *filename* [LPTR]

Synonyms

GROUP-STAT, ISTAT

Description

The ECL **GROUP.STAT** command displays file and group statistics, including size and number of records.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Analyzes the dictionary portion of the file.
filename	The name of a UniData file to be analyzed.
LPTR	Sends output to the printer instead of the terminal screen.

GROUP.STAT Parameters

Examples

The following example displays group statistics for the INVENTORY demo file. During command execution a greater than sign (>) displays to represent each record.

```
:GROUP.STAT INVENTORY
File = INVENTORY modulo=19 hash type=0 blocksize=1024
Split/Merge type = KEYONLY
Grp# Bytes Records
  0   764     9>>>>>>>>>
  1   628     8>>>>>>>>
  2   736     9>>>>>>>>>
  3   542     7>>>>>>>
  4   558     7>>>>>>>
  5   672     9>>>>>>>>>
  6   662     9>>>>>>>>>
  7   722    10>>>>>>>>>>
  8   736    10>>>>>>>>>>
  9   840    11>>>>>>>>>>>
 10   868    11>>>>>>>>>>>
 11   987    12>>>>>>>>>>>>
 12   757    11>>>>>>>>>>>
 13   642     8>>>>>>>>
 14   600     9>>>>>>>>>
 15   740     9>>>>>>>>>
 16   759    10>>>>>>>>>>
 17   697     9>>>>>>>>>
 18   595     7>>>>>>>
======= =====
    13505   175   Totals
      542     7   Minimum in a group
      987    12   Maximum in a group
    710.8   9.2   Averages per group
    110.66  1.44  Standard deviation from average
    0.16  0.16  Percent std dev from average
File has 1 over files, 1 prime files
:GROUP.STAT DICT INVENTORY
File = DICT INVENTORY modulo=1 hash type=0 blocksize=1024
Grp# Bytes  Records
  0   575   16>>>>>>>>>>>>>>>>
======= =====
     575   16  Totals
     575   16  Minimum in a group
     575   16  Maximum in a group
   575.0  16.0  Averages per group
   0.00  0.00 Standard deviation from average
   0.00  0.00 Percent std dev from average
The actual file size in bytes       = 2048.
:
```

The next example shows the sort of distribution that contributes to inefficient file access. To generate the next example, memresize converted a copy of the INVENTORY demo database file to the KEYDATA split/merge type (inappropriate because of the wide variation in record sizes) and REBUILD.FILE rehashed the keys:

```
:GROUP.STAT INV_COPY
File = INV_COPY modulo=69 hash type=0 blocksize=1024
Split/Merge type = KEYDATA
Grp# Bytes  Records
  0   295     4>>>>
  1     0     0
  2   291     4>>>>
  3     0     0
  4   282     3>>>
  5    72     1>
  6   186     3>>>
  7    77     1>
  8   296     4>>>>
  9    93     1>
.
.
.
 67   153     2>>
 68   613     7>>>>>>>
======= =====
    13505   175   Totals
        0     0   Minimum in a group
      687     8   Maximum in a group
    195.7   2.5   Averages per group
    169.26  2.10  Standard deviation from average
    0.86  0.83  Percent std dev from average
File has 1 over files, 1 prime files
:
```

gstt

Syntax

gstt

Description

The system-level **gstt** command displays the status and usage of global pages of shared memory. See the *Administering UniData* manual for more information on shared memory.

Use this command at the system prompt, or use the ECL (bang) command to execute this command from the ECL prompt.

Example

The following example illustrates a gstt command display:

```
% gstt
--------------------- GCTs Statistics -------------------

Total GCTs (GSMs allowed): 40
Pages/GSM.................: 32 (4096K bytes)
Bytes/Page................: 128K bytes

GCTs used (GSMs created).: 1 (3% of 40)

    Active GSMs....: 1 (32 pages in total, 4096K bytes)

        Pages Used...........: 2 (6%, 256K bytes)
        Pages Freed..........: 30 (94%, 3840K bytes)

    Inactive GSMs..: 0

        Pages Freed..........: 0 (0K bytes)

    Total Pages Used......: 2 (6%, 256K bytes)
    Total Pages Freed.....: 30 (94%, 3840K bytes)
    Total memory allocated: 4096K bytes
 ----------------- End of GCTs Statistics ----------------
```

guide

Syntax

guide *filename* [*filename...*] [-b [*b_filename*] | -nb] [-d {1 | 2 | 3 } [{-l | -s} *count*]] [[-o [*o_filename*] [-p *page_length*] | -np] [-na] [-ne] [-ns] | [-a [*a_filename*] | -na] [-e [*e_filename*]] [-s [*s_filename*]] [-f [*f_filename*]] [-h {a | 0 | 1 } [-m *new_modulo*]] [-i [*i_filename*]] [-r [*r_filename*]] [-Z *num_child_processes*] [-U###]

Description

The system-level **guide** command analyzes hashed files, generates statistics, and provides suggestions for optimizing file sizes and ensuring data integrity. **UniData must be running when you execute guide.**

Default reports include:

- Management advice (option -a)
- File errors (option -e)
- Detailed statistics (option -s [*s_filename*])
- Damaged groups (option -f)

For detailed information about using guide to assess file damage and to manage file integrity, refer to the *Administering UniData* manual.

You must have read and write permissions on files analyzed.

guide no longer requires exclusive access to a file, and utilizes parallel processing.

Although guide analyzes recoverable files, the output of guide is not recoverable. Therefore, if a system or media failure occurs while you are running guide, you need to rerun guide after recovery. For more information about the guide utility and recoverable files, see the *Administering the Recoverable File System* manual.

Because new files are created by each execution, you should review and delete unneeded ones or you may accumulate a large number of them.

Tip: Once you have identified damaged groups with guide, use the UniData system-level fixfile command to repair them.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename [*filename*...]	Specifies the file or files to analyze. Separate multiple file names with a space. You must have read and write access to these files.
-b [*b_filename*]	Summarizes file analysis in *b_filename*. Default file name is GUIDE_BRIEF.LIS.
-nb	Default. No summary report is generated.
-d {1 | 2 | 3}	Reports on file size: 1 — Summarizes file size info. 2 — Default; reports file size info. 3 — Adds information about distribution of data sizes. **Note**: Cannot be used with the -ns option.
{-l | -*s*} *count*	Adds to information displayed by -d. Displays, in quotation marks, keys of smallest records. Key ends with * if truncated. *count* specifies number to list. Default is 3. -l — lists keys only -s — sorts and lists keys **Note**: Must be combined with the -d option.
-o[*o_filename*]	Combines output in *filename*, rather than placing it in separate files. If *filename* is not specified, sends combined output to the standard output device. The default output device is the display terminal. **Tip**: Specify output device at the operating system level (for example, stty in UNIX).

guide Parameters

Parameter	Description
-p *page_length*	When output from option -o is directed to the terminal, specifies display page length. Default is 24 lines. At end of page display, UniData prompts: `Press RETURN to continue...` You must respond with one of the following: ENTER — Displays the next page. N — Scrolls the remainder of the output with no pagination. Q — Quits display.
-np	Default. Scrolls output on terminal with no pagination.
-na	No management advice is reported. This is the opposite of the -a parameter.
-ne	No detailed error reporting. This is the opposite of the -e parameter.
-ns	Default. No detailed statistical reporting. This is the opposite of the -s parameter.
-a [*a_filename*]	Default. Reports file management advice in *a_filename*. Default file name is GUIDE_ADVICE.LIS.
-e [*e_filename*]	Default. Reports statistical errors in *e_filename*. Default file name is GUIDE_ERRORS.LIS.
-s [*s_filename*]	Default. Reports detailed statistical information in *s_filename*. Default file name is GUIDE_STATS.LIS.
-f [*f_filename*]	Default. Reports damaged groups in *f_filename*. Default *f_filename* is GUIDE_FIXUP.DAT. *f_filename* can be used as input for ECL commands fixfile, dumpgroup, and fixgroup.

guide Parameters (continued)

Parameter	Description
-h {a | 0 | 1}	Evaluates hash algorithms of type: ■ a — evaluates both types ■ 0 ■ 1 **Note**: This option produces no output for dynamic files.
-m *new_modulo*	Analyzes the effects a different modulo would have on *filename*. Must be used with the -h parameter.
-i [*i_filename*]	Analyzes all files listed in *i_filename*. Default file name is GUIDE_INPUT.DAT. In *i_filename*, list one file name per line. Blank lines and lines beginning with ! are ignored.
-r [*r_filename*]	Directs output to UniData database *r_filename*. *r_filename* must be the system-level file name. Copy the dictionary for *r_filename* from *udthome*/sys/D_UDT_GUIDE on UNIX or *udthome*\sys\D_UDTGUIDE on Windows Platforms. Later, you can execute UniQuery commands against *r_filename*.
-Z *num_child_processes*	Defines the number of concurrent processes to use when analyzing the file. The default is 4. If the file guide is analyzing has less than 100 groups, guide only uses one process.
-U###	Searches files for the existence of the ASCII character you specify in the records and keys in the file.

guide Parameters (continued)

Output Reports

Depending on the parameter you include, guide may create any or all of the following reports. If any of these output files exist when you execute guide, UniData changes all output file names by appending a six-digit time stamp to each file name. This way, only the most current output files have no time stamp; and if a particular output file is not created during this execution, no file of that name exists.

Report	Default File Name	Parameter	Description
File management advice	GUIDE_ADVICE.LIS	-a	Provides advice for improving file sizing or cleanup.
File errors	GUIDE_ERRORS.LIS	-e	Lists structural errors.
Detail	GUIDE_STATS.LIS	-s	Details statistics on *filename*.
Summary	GUIDE_BRIEF.LIS	-b	Summarizes record counts, total size, used size, and modulo.
Damaged groups	GUIDE_FIXUP.DAT	-f	Lists damaged groups. This file can be used as input for ECL commands fixfile, dumpgroup, and fixgroup.

guide Output Files

Using the U### Option

If you use the U### option, guide searches files for the existence of the ASCII character you specify in the records and keys in the file. For example, guide -U0 searches files for CHAR(0).

If guide encounters the character you specify, it returns a message similar to the following example:

```
TEST
File Integrity:
   Group 0, block 1, record number 0 = “AAA” has char (0) in key
   Group 0, block 1, record number 0 = “AAA” record has char (0)
   in data
   Group 0, block 0, long record number 1 = “BBB” record has
   char (0) in data.
   Group 2, block 5, long record number 0 = “AAA” record has
   char (0) in data.

Files Processed:    1
Errors encountered: 4
```

Note: Using the -U### option may degrade the performance of guide.

Examples

The following report is generated by the -s [*s_filename*] parameter. By default, it is stored in GUIDE_STATS.LIS:

```
INVENTORY
  Basic statistics:
     File type.............................. Recoverable Dynamic
Hashing
    File size
     [dat001]............................... 20480
     [over001].............................. 9216
   File modulo.............................. 19
   File minimum modulo...................... 19
   File split factor........................ 60
   File merge factor........................ 40
   File hash type........................... 1
   File block size.......................... 1024
  Group count:
   Number of level 1 overflow groups....... 8
   Primary groups in level 1 overflow...... 8
  Record count:
   Total number of records.................. 175
   Average number of records per group..... 9.21
   Standard deviation from average......... 3.58
  Record length:
   Average record length.................... 71.20
     Standard deviation from average......... 18.30
```

This output was generated on a damaged version of the INVENTORY file:

```
:!guide INVENTORY -o

INVENTORY
  Basic statistics:
    File type.............................. Recoverable Dynamic
Hashing
    File size
      [dat001]............................. 20480
      [over001]............................ 3072
    File modulo............................ 19
    File minimum modulo.................... 19
    File split factor...................... 60
    File merge factor...................... 40
    File hash type......................... 0
    File block size........................ 1024
  File Integrity:
    Group 2, block 3 has incorrect group number 1633746946
  Management advice:
         This file's integrity has been compromised,
    please repair it.

Files processed:   1
Errors encountered: 1
```

The following file listing shows a set of files produced over a four-day period. Notice the following:

- Only GUIDE_FIXUP.DAT has no time stamp, indicating that this is the only file created during the last execution of guide. This was the execution in the preceding example.

- GUIDE_STATS.LIS_032798_A is the latest version of this file, indicating that this file was not created during the last two executions of guide.

```
:ls -lt GUI*
-rw-r--r--   1 carolw   staff         15 Mar 27 15:36
GUIDE_FIXUP.DAT
-rw-r--r--   1 carolw   staff        154 Mar 27 15:34
GUIDE_ADVICE.LIS_032798_B
-rw-r--r--   1 carolw   staff       1787 Mar 27 15:34
GUIDE_ERRORS.LIS_032798_B
-rw-r--r--   1 carolw   staff         15 Mar 27 15:34
GUIDE_FIXUP.DAT_032798
-rw-r--r--   1 carolw   staff        555 Mar 27 15:34
GUIDE_STATS.LIS_032798_B
-rw-r--r--   1 carolw   staff         46 Mar 27 15:20
GUIDE_ADVICE.LIS_032798_A
-rw-r--r--   1 carolw   staff         46 Mar 27 15:20
GUIDE_ERRORS.LIS_032798_A
-rw-r--r--   1 carolw   staff         46 Mar 27 15:20
GUIDE_STATS.LIS_032798_A
-rw-r--r--   1 carolw   staff         46 Mar 27 15:16
GUIDE_ADVICE.LIS_032798
-rw-r--r--   1 carolw   staff         46 Mar 27 15:16
GUIDE_ERRORS.LIS_032798
-rw-r--r--   1 carolw   staff         46 Mar 27 15:16
GUIDE_STATS.LIS_032798
-rw-r--r--   1 carolw   staff         14 Mar 26 15:49
GUIDE_FIXUP.DAT_032698
-rw-r--r--   1 carolw   staff         46 Mar 24 11:20
GUIDE_ADVICE.LIS_032498
-rw-r--r--   1 carolw   staff         46 Mar 24 11:20
GUIDE_ERRORS.LIS_032498_B
-rw-r--r--   1 carolw   staff       1497 Mar 24 11:20
GUIDE_STATS.LIS_032498_B
-rw-r--r--   1 carolw   staff         46 Mar 24 11:19
GUIDE_ERRORS.LIS_032498_A
-rw-r--r--   1 carolw   staff       1848 Mar 24 11:19
GUIDE_STATS.LIS_032498_A
-rw-r--r--   1 carolw   staff         46 Mar 24 11:18
GUIDE_ERRORS.LIS_032498
-rw-r--r--   1 carolw   staff       1497 Mar 24 11:18
GUIDE_STATS.LIS_032498
```

Related Commands

dumpgroup, fixfile, fixgroup, verify2

guide_ndx

Syntax

guide_ndx{-x |-X}{1|2 |3},*{index_names, ... |* ALL} [-t *template* |-T *template*] *filename*

Description

As with other UniData file types, an index file could become corrupt due to hardware failures, the interruption of a write to the index file, or an incomplete write. The guide_ndx utility checks for physical and logical corruption of an index file.

If an index file is corrupt, UniData displays a run time error when a UniData process tries to access the index. If the index file is associated with a recoverable file, a message is written to the sm.log.

The guide_ndx command creates two files, the GUIDE_XERROR.LIS and the GUIDE_STATS.LIS. GUIDE_ERROR.LIS lists any corruption found in the index file, and GUIDE_STATS.LIS list statistics about the index. If you have a corrupt index, you must rebuild it using the CREATE.INDEX and BUILD.INDEX commands. For more information and creating and building indexes, see *Using UniData*.

Note: We recommend deleting the index with the DELETE.INDEX ALL command. Using the ALL option deletes all alternate key indexes and the index file itself.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-x{1 | 2 | 3}	Determines the type of checking guide_ndx performs. ■ 1 – Performs physical checking ■ 2 – Performs logical checking ■ 3 – Performs physical and logical checking
index_names	The index names you want guide_ndx to check. Separate each index name with a comma, or enter ALL to check all indexes for the file.
-t *template*	The template to use for output files. The default is GUIDE.
filename	The name of the data file containing the index.

guide_ndx Parameters

Example

The following example illustrates the contents of the GUIDE_XERROR.LIS file when guide_ndx detects corruption:

```
%pg GUIDE_XERROR.LIS
INVENTORY
Checking index 'INV_DATE' physically...
Invalid key length (30569, key item 65) in node 24576.
Bytes left not matched (recorded 3157, calulated 4933) in node
24576.
Checking index 'FEATURES' physically...
Checking index 'COLOR' physically...
```

The next example illustrates the GUIDE_XSTATS.LIS file:

```
%pg GUIDE_XSTATS.LIS
INVENTORY
Large index.......... INVENTORY/idx001
Alternate key length. 60
Node/Block size...... 6K
OV blocks............ 1
# of indices......... 3
Index auto update.... Enabled, No updates pending
Index Name F-type V-type K-type Nulls Dups F-No/VF-pos (Root)
INV_DATE D S N Yes Yes 1 (24576 [1-4])
FEATURES D S T Yes Yes 4 (30720 [1-5])
COLOR D M T Yes Yes 5 (36864 [1-6])
```

The following table describes the column heading that display in output for the X_STATS.LIS file.

Column Heading	Description
Index name	Name of the index.
F-type	Type of attribute indexed: D for data attribute, V for a virtual attribute.
V-type	Value code for the attribute. S for singlevalued, M for multi-valued or multi-subvalued.
K-type	Type of index: Txt for text, Num for numeric.
Nulls	“Yes” indicates that empty strings are indexed. “No” indicates that empty strings are not indexed.
Dups	“Yes” indicates that duplicate keys are allowed in the alternate key index. “No” indicates that duplicate keys are not allowed.
F-No/VF-expr	The attribute location for alternate key indexes built on data attributes (D-type) or the virtual attribute definition for alternate key indexes built on virtual attributes (V-type).

X_STATS.LIS Display

HASH.TEST

Syntax

HASH.TEST *filename* [(B | (H | (N | (P]

Synonym

HASH-TEST

Description

The ECL **HASH.TEST** command manipulates certain characteristics of a UniData data file in a test environment without changing the actual parameters of the file. When you use this command, UniData prompts for values for modulo number, hash type, and block size multiplier.

Note: For a block size of 512 bytes, UniData accepts either -1 or 512 at the block size multiplier prompt. Otherwise, UniData uses the block size multiplier. For example, 1=1024, 2=2048, and so on.

UniData calculates statistics based upon these user-supplied values and the contents of the file, and then displays the following data:

- Average number of items per group.
- Average number of bytes per group.
- Number of empty groups.
- Standard deviation.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of a UniData data file.
(B	Suppresses the initial linefeed.
(H	Generates a histogram and detailed information for every group.
(N	Suppresses automatic paging.
(P	Sends output to the printer.

HASH.TEST Parameters

Example

In the following example, UniData prompts for test values and then calculates theoretical statistics for the CLIENTS demo file. The actual parameters for the data file have not changed. The user has entered a block size multiplier of 2, indicating a block size of 2048. Also, the (H option produces detailed information on each group including number of bytes and items, as well as a histogram indicating relative size.

```
:HASH.TEST CLIENTS (H
TEST MODULO: 23
HASH TYPE: 1
BLOCK SIZE(K, -1 for 512): 2

FILE: CLIENTS  MOD: 23  HASH TYPE: 1              16:11:54 Jun 09
1999
      BYTES ITEMS
   0    779     7 *>>>>>>>
   1    422     4 *>>>>
   2    661     6 *>>>>>>
   3    803     7 *>>>>>>>
   4    741     7 *>>>>>>>
   5    922     8 *>>>>>>>>
.
.
.
ITEM COUNT=       134, BYTE COUNT   14586, AVG. BYTES/ITEM=  109
AVG. ITEMS/GROUP=5.8, STD. DEVIATION=1.8, AVG. BYTES/GROUP=634.2
EMPTY GROUPS=       0
    :
```

HELP

Syntax

HELP [*topic*] [*command*] [-k *keyword*]]

Description

The ECL **HELP** command displays online help for UniData commands, including the following topics:

- UniData ECL commands and keywords, including commands you enter at the system prompt. You can enter synonyms for commands from legacy applications.
- UniBasic commands, functions, and operators.
- UniQuery commands and keywords.
- UniData SQL commands and keywords.

If you use this command without any options, UniData displays command syntax and indicates valid topics.

Tip: You can access the UniData help system from within AE by using XEQ (execute ECL command). For example, from within AE enter "XEQ HELP OPEN" to display help on the UniBasic OPEN command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
command	Any UniBasic, UniData, UniQuery, or UniData SQL command. If the command contains multiple words separated by a space, such as **CREATE TABLE** in UniData SQL and **INPUT @** in UniBasic, you must enclose the command in quotation marks.
topic	A subject. These are product names (for example, UNIDATA, UNIBASIC, UNIQUERY, or SQL). If you enter a topic without a command, HELP lists all the available commands for that topic. You can also enter a command with a topic to specify which command to display if there is more than one topic with the same command. For example, there are three SELECT commands (UniQuery, UniData SQL, and UniBasic).
-k *keyword*	Indicates a word to search for in the help system. This feature is not case-sensitive.

HELP Parameters

HUSH

Syntax

HUSH [ON | OFF]

Description

The ECL **HUSH** command turns on or off system output display on the terminal.

Warning: Do not use HUSH ON before you execute a command, paragraph, or sentence that requests user input. The process will appear to hang.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	Toggles between ON and OFF.
ON	UniData does not display the colon prompt nor any output to the terminal.
OFF	Default. UniData displays the colon prompt and output to the terminal.

HUSH Parameters

Examples

In the following example, the HUSH command prevents UniData from displaying the colon prompt, command lines, and the output that follows until the HUSH OFF command is entered. For this example, a UniQuery statement and HUSH OFF follow HUSH ON.

```
:HUSH ON
:
```

To verify that UniData recognized the command input after the HUSH ON command was entered, display the command stack. In the following example, notice item number 2. This is the command that was entered while HUSH ON was active.

```
:.L
...
 3 HUSH ON
 2 LIST CLIENTS WITH LNAME LIKE "P..."
 1 HUSH OFF
:
```

HUSHBASIC

Syntax

HUSHBASIC [ON | OFF]

Description

The ECL **HUSHBASIC** command determines whether brief or detailed UniBasic error messages are displayed.

For more information about UniBasic, see the *Developing UniBasic Applications* manual.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	Toggles between ON and OFF.
ON	Displays brief UniBasic error messages.
OFF	Displays detailed UniBasic error messages.

HUSHBASIC Parameters

Example

The following example compares the brief versus detailed error message displayed when HUSHBASIC is ON and OFF using first the keywords ON and OFF, then executing HUSHBASIC with no keyword, toggling between the two settings.

```
:HUSHBASIC OFF
:RUN BP TESTPROG
In at line 1 can not find object/catalog file: 'BP/_TESTPROG'.
:HUSHBASIC ON
:RUN BP TESTPROG
can not find object/catalog file: 'BP/_TESTPROG'.
:HUSHBASIC
:RUN BP TESTPROG
In at line 1 can not find object/catalog file: 'BP/_TESTPROG'.
:HUSHBASIC
:RUN BP TESTPROG
can not find object/catalog file: 'BP/_TESTPROG'.
```

ipcstat

Syntax

ipcstat [-q] [-m] [-s]

Description

The system-level ipcstat command displays the status of interprocess communication (IPC) facilities. In addition, UniData provides the names of the UniData processes associated with each resource.

For detailed information about this utility, see the section on managing IPC facilities in the *Administering UniData* manual.

Note: Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the colon prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	Displays the status of all message queue, shared memory, and semaphores.
-q	Displays status of message queues.
-m	Displays status of shared memory.
-s	Displays status of semaphores.

ipcstat Parameters

Example

The following example shows an ipcstat display. Where unknown appears in the output, resources were created by non-UniData processes.

```
%ipcstat
```

ISTAT

ISTAT is a synonym for the GROUP.STAT command. For more information, see GROUP.STAT.

Synonyms

GROUP.STAT, GROUP-STAT

kp

Syntax

kp

Description

The system-level **kp** command reports on current UNIX kernel parameters related to shared memory, semaphores, and message queues. This command is supported on UniData for UNIX only. The report is routed to the display terminal. See your UNIX system documentation for explanations of these kernel parameters.

Note: If you are not logged on as root, some items in the report may display as -1. This indicates that the values for that item are not available to you.

Use this command at the system prompt, or use the ECL (bang) command to execute this command from the ECL prompt.

Example

The following is a sample kp report:

```
# kp
shmmni = 200
shmseg = 120
shmmax = 67108864
shmmin = 1

msgmni = 100
msgtql = 40
msgmnb = 16384
msgmax = 8192

semmni = 64
semmnu = 100
```

LIMIT

Syntax

LIMIT

Description

The ECL LIMIT command displays maximum size limits for elements of UniData. These limits are not configurable.

See *Using UniQuery* for more information on limits to UniQuery parameters.

Example

The following example shows UniData limits:

```
:LIMIT
U_MAXFNAME: Unix file name limit = 46.
U_NAMESZ: Record id(key) size = 126.
U_SELEMAX: Number of select list = 10.
U_MAXDATA: Number of DATA statement = 500.
U_HEADSZ: HEADER/FOOTER length = 2120.
U_MAXHASHTYPES: Number of hash functions = 3.
U_MAXSORT: Number of sort fields(BY...) in LIST = 20.
U_MAXWITH: WITH stack size = 120.
U_MAXWHEN: WHEN stack size = 60.
U_MAXCAL: Number of SUM+AVG+PCT+CAL in LIST = 54.
U_MAXBREAK: Number of BREAK.ON+BREAK.SUP in LIST = 15.
U_MAXLIST: Number of attribute names in LIST = 999.
U_LINESZ: Page width in printing = 272.
U_PARASIZE: Paragraph name and its parameter size = 256.
U_LPCMD: System spooler name = lp -c .
U_MAXPROMPT: Number of prompts allowed in paragraph = 60.
U_FSIZE: Dictionary field name size = 31.
U_MAXVALUE: Number of values WHEN can handle = 10240.
U_MAXBYEXPVAL: Number of values BY.EXP can handle = 10240.
U_SENTLEN: Maximum sentence length = 9247.
U_PROCBUFSZ: Proc buffer size = 4095.
U_NIDES: Maximum number of virtual fields in query= 256.
:
```

LINE.ATT

Syntax

LINE.ATT *line* [*DELAY*]

Synonym

LINE-ATT

Description

The ECL LINE.ATT command attaches a communication line to the current process. The attaching process then has exclusive use of that line until it is detached with the LINE.DET command. A single process can attach up to five resources per UniData session.

Warning: On some platforms, you must specify DELAY in LINE.ATT to avoid problems with subsequent UniBasic SEND commands overlaying data.

Before you can use this command, you must execute the SETLINE command to initialize the communications line.

Tip: Tape devices, printers, and other devices must be defined within UniData before they can be accessed. Refer to your host operating system documentation for information about setting up peripherals on your system. For information on defining devices within UniData, see Administering UniData.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
line	A number assigned to the (line) device you are attaching. The line number is defined by the SETLINE command.
DELAY	Your process waits for a “received” message before allowing further activity by the process. This option does not time out, but waits indefinitely.

LINE.ATT Parameters

Example

In the following example, UniData attaches line 0 to the current process:

```
:LINE.ATT 0
LINE 0 ATTACHED
```

Related Commands

UniData

LINE.DET, LINE.STATUS, PROTOCOL, SETLINE, UNSETLINE

UniBasic

GET, SEND For information, see the *UniBasic Commands Reference*.

LINE.DET

Syntax

LINE.DET *line*

Synonym

LINE-DET

Description

The ECL LINE.DET command releases a communication line so it is no longer reserved for the exclusive use by the current user process.

Note: You can concurrently attach up to five lines per UniData session. Use SETLINE to define the lines and LINE.ATT to attach them.

Tip: Tape devices, printers, and other devices must be defined within UniData before they can be accessed. Refer to your host operating system documentation for information about setting up peripherals on your system. for information on defining devices within UniData, see Administering UniData.

Examples

In the following example, the LINE.DET command detaches line 0 from the current environment:

```
:LINE.DET 0
LINE 0 DETACHED
```

Related Commands

UniData

LINE.ATT, LINE.STATUS, PROTOCOL, SETLINE, UNSETLINE

UniBasic

GET, SEND For information, see the *UniBasic Commands Reference*.

LINE.STATUS

Syntax

LINE.STATUS

Synonym

LINE-STATUS

Description

The ECL LINE.STATUS command displays the current status of all communication lines.

Tip: Tape devices, printers, and other devices must be defined within UniData before they can be accessed. Refer to your host operating system documentation for information about setting up peripherals on your system. For information on defining devices within UniData, see Administering UniData.

Example (UniData for UNIX)

In the following example, UniData displays all communication lines:

```
:SETLINE 0 /dev/pty/ttyv6
:LINE.STATUS
LINE# STATUS UDT# USER-NAME DEVICE-NAME
0 Available N/A N/A /dev/pty/ttyv6
Line number(s) are attached by the current udt process:

None
:
```

Example (UniData for Windows Platforms)

In the following example, UniData displays all the lines in the system set by SETLINE:

```
:SETLINE 0 COM1
:LINE.STATUS
LINE# STATUS UDT# USER-NAME DEVICE-NAME
0 Available N/A N/A COM1
Line number(s) are attached by the current udt process:
None
:
```

Related Commands

UniData

LINE.ATT, LINE.DET, PROTOCOL, SETLINE, UNSETLINE

UniBasic

GET, SEND For information, see the *UniBasic Commands Reference*.

LIST.CONNECT

Syntax

LIST.CONNECT

Synonym

LIST-CONNECT

Description

The LIST.CONNECT command displays NFA (Network File Access) parameters for all connections. When you enter LIST.CONNECT, UniData displays the following information about server connections:

- UniData process number.
- USRNBR (System-level process ID assigned to a UniData session).
- UID (system-level user ID).
- User name.
- Type of user, for example client (udt/clnt) or server (udt/svr).
- Family.
- Domain.

For more information on NFA, see *Developing OFS/NFA Applications.*

LIST.CONNECT Display

The following table describes the column headings that display in the output for the LIST.CONNECT command.

Column Heading	Description
UDTNO	The UniData user number.
USRNBR	System-level process ID assigned to a UniData session.
UID	The system-level user ID number.
USRNAME	The user name.
USRTYPE	The type of process. For NFA, this is always “udt.”
FAMILY	The OFS Family (for NFA, this is always UDT) described in the VOC entry for the file being accessed.
DOMAIN	Information on the domain described in the VOC entry for the file being accessed. It is in the following syntax: *machine:voc:port* where *machine* is the name of the server machine, *voc* is the path of the VOC file, and *port* is the port number being used.

LIST.CONNECT Display

Example

In the following example, UniData displays the current NFA users:

```
:LIST.CONNECT
UDTNO USRNBR UID USRNAME USRTYPE FAMILY DOMAIN
3 18910 1104 ubj01 udt UDT hp1:/users/ubj01:1155
8 19156 1083 peggys udt UDT hp1:/users/ubj01:1155
```

LIST.INDEX

Syntax

LIST.INDEX *filename* [*attribute* [*attributeM...attributeN*] | ALL] [STATISTICS | STATS |DETAIL] [NO.PAGE] [LPTR *n*]

Synonym

LIST-INDEX

Description

The ECL **LIST.INDEX** command displays information about alternate key indexes for a particular data file.

If LIST.INDEX completes successfully, UniData sets @SYSTEM.RETURN.CODE to the number of indexes listed. If LIST.INDEX does not complete successfully, UniData sets @SYSTEM.RETURN.CODE to -1.

For detailed information about indexes, see *Using UniData*.

Using Indexes Created in an Earlier Release

Keep the following in mind when upgrading or using an index that was created with and earlier release of UniData:

- On UniData for UNIX, when upgrading from a release earlier than 3.3, you need to rebuild indexes. UniData added a time stamp feature at Release 3.3.
- Indexes created at Release 4.1 of UniData for UNIX or Release 3.6 of UniData for Windows NT, are not backwardly compatible. Beginning with these releases, indexes were no longer compressed.

Tip: Use the UniBasic INDICES function to find out when an index was created.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of the UniData file.
attribute | *ALL*	Indicates one or more alternate key indexes to be examined. If you do not stipulate *attribute*, UniData displays all alternate key indexes for the file.
STATISTICS | STATS	Lists detailed statistical information about alternate key indexes on *filename*. If you do not indicate the alternate key index name (*attribute*), UniData provides statistics for all alternate key indexes. **Note**: Using this keyword on large files may adversely affect system performance.
DETAIL	Displays index entries.
NO.PAGE	Prevents the report from pausing at the end of each display page.
LPTR *n*	Directs the report to logical printer *n*.

LIST.INDEX Parameters

Examples

For the following example, we first created three alternate key indexes on the ORDERS file. UniData displays information about these indexes:

```
:LIST.INDEX ORDERS
Alternate Key Index Details for File ORDERS Page 1
File.................. ORDERS
Alternate key length.. 20
Node/Block size....... 4K
OV blocks............. 1 (1 in use, 0 overflowed)
Indices............... 3 (1 D-type)
```

LIST.INDEX Display

The following table describes the column heading that display in output for the LIST.INDEX command.

Column Heading	Description
Index name	Name of the index.
F-type	Type of attribute indexed: D for data attribute, V for a virtual attribute.
K-type	Type of index: Txt for text, Num for numeric.
Built	"No" indicates that the index has not been built using the BUILD.INDEX command; "Yes" indicates that the index has been built.
Empties	"Yes" indicates that empty strings are indexed. "No" indicates that empty strings are not indexes.
Dups	"Yes" indicates that duplicate keys are allowed in the alternate key index. "No" indicates that duplicate keys are not allowed.
In-DICT	"Yes" indicates that the dictionary contains an attribute with the same name as the index.
S/M	"S" indicates that the indexed attribute is singlevalued. "M" indicates that the indexed attribute is multivalued.
F-No/VF-expr	The attribute location for alternate key indexes built on data attributes (D-type) or the virtual attribute definition for alternate key indexes built on virtual attributes (V_type).

LIST.INDEX Display

STATISTICS Display

The following table describes the column headings that display in the output for the LIST.INDEX command when you include the STATISTICS keyword.

Column Heading	Description
Index name	The index for which statistics are provided.
# of Keys	The total number of alternate key values in the index.
# of OV Keys	The total number of overflowed key values in the index.
Records per Alternate key	The average, minimum, and maximum number of records associated with each of the alternate key values.

STATISTICS Display

The following example shows the STATISTICS display for a group of alternate key indexes that we created for the ORDERS demo file. Page 2 contains the statistics.

```
:LIST.INDEX ORDERS STATISTICS
Alternate Key Index Details for File ORDERS Page 1
File.................. ORDERS
Alternate key length.. 20
Node/Block size....... 4K
OV blocks............. 1 (0 in use, 0 overflowed)
Indices............... 4 (1 D-type)
Index updates......... Enabled, No updates pending
Index-Name...... F-type K-type Built Empties Dups In-DICT S/M F-no/VF-
expr....
NAME V Txt Yes Yes Yes Yes S TRANS('CLIENTS',
CLIENT_NO,'FNAME','X'): " ": TRANS('CLIENTS',CLIENT_NO,'LNAME','
X')
GRAND_TOTAL V Num Yes Yes Yes Yes S PRICE*QTY; SUM(S

OV blocks............. 1 (0 in use, 0 overflowed)
Indices............... 4 (1 D-type)
Index updates......... Enabled, No updates pending
Index-Name...... F-type K-type Built Empties Dups In-DICT S/M F-no/VF-
expr....
NAME V Txt Yes Yes Yes Yes S TRANS('CLIENTS',
CLIENT_NO,'FNAME','X'): " ": TRANS('CLIENTS',CLIENT_NO,'LNAME','X')
GRAND_TOTAL V Num Yes Yes Yes Yes S PRICE*QTY; SUM(SUM(@1))
COUNTRY V Txt Yes Yes Yes Yes S TRANS('CLIENTS',CLIENT_NO,'COUNTRY','X')
PRODUCT_NO D Num Yes Yes Yes Yes M 4
Details for Index NAME in File ORDERS Page 2
Alternate Key Value # of Records for Key Overflowed
1 No
Adam Monterey 4 No
Al Elliott 1 No
Alicia Rodriguez 4 No
Andre Halligan 1 No
...
Statistics:
Records per Alternate Key
Index name # of Keys # of OV Keys Average Minimum Maximum
NAME 69 0 2.8 1 7
Details for Index GRAND_TOTAL in File ORDERS Page 6
Alternate Key Value # of Records for Key Overflowed
$0.00 1 No
$17.99 1 No
$39.95 1 No
$42.89 1 No
...
Statistics:
Records per Alternate Key
Index name # of Keys # of OV Keys Average Minimum Maximum

GRAND_TOTAL 189 0 1.0 1 2
...
```

Related Commands

BUILD.INDEX, CREATE.INDEX, DELETE.INDEX, DISABLE.INDEX, ENABLE.INDEX,UPDATE.INDEX

LIST.LANGGRP

Syntax

LIST.LANGGRP

Synonym

LIST-LANGGRP

Description

The ECL **LIST.LANGGRP** command displays the current language group ID. For more information about using UniData in languages other than English, see *UniData International*.

Language Group ID

The following table shows the UniData language names (udtlang) and the language group identifiers.

Group #	udtlang Name	Language Group ID
Group 1	English (US, UK)	255 / 192 / 129
Group 2	Japanese (EUC) French (ISO8859-1) English_G2 (English)	159 / 130 / 129

UniData Language Groups

Example

The following example shows a LIST.LANGGRP display:

```
:LIST.LANGGRP
Current language group ID: 255/192/129
```

LIST.LOCKS

Syntax

LIST.LOCKS

Synonym

LIST-LOCKS

Description

The ECL **LIST.LOCKS** command displays all locks currently set on system resources.

For more information on creating and clearing locks on system resources, see the CLEAR.LOCKS and LOCK commands.

Any of the following UniData commands can issue locks that LIST.LOCKS displays.

Command	How Lock Is Released
acctrestore	UniData releases the lock when the account is restored (UniData finishes reading the tape).
LINE.ATT	ECL command. LINE.DET releases the lock.
LOCK	UniBasic statement. UNLOCK releases the lock. For more information, see the *UniBasic Commands Reference*.
LOCK *num*	ECL command. BYE or a UniBasic UNLOCK statement releases the lock.
PHL *num*	PQN command.
T.ATT	ECL command. T.DET releases the lock.

Commands That Issue UniData Locks

Example (UniData for UNIX)

In the following example, UniData displays the status of all system resources that are locked:

```
:LIST.LOCKS
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
1 2253 1283carolw ts/1 semaphor -1 0 1 X 10:44:29 Jul 31
6 2365 1283carolw ts/6 semaphor -1 0 2 X 10:44:29 Jul 31
```

LIST.LOCKS Display

The following table describes the column headings that display in the output for the LIST.LOCKS command.

Column Heading	Description
UNO	Sequential number UniData assigns to the UniData session.
UNBR	Process Group ID (pid) of the user setting the lock.
UID	User ID of the user setting the lock.
UNAME	Login name of the user setting the lock.
TTY	Terminal device of the user setting the lock.
FILENAME	File name in which the record is locked.
INBR	I-node of the locked file.
DNBR	Used in conjunction with INBR to define the file at the operating system level.
RECORD ID	Record ID of the locked record.
M	Record lock mode.
TIME	The time at which the lock was set.
DATE	The date on which the lock was set.

LIST.LOCKS Display

Example (UniData for Windows Platforms)

In the following example, UniData displays the status of all system resources that are locked:

```
:LIST.LOCKS
UNO UNBR UID UNAME FILE NAME RECORD ID M TIME DATE
002 122 1000 claireg semaphore 64 X 10:44:29 Jul 31
:
```

LIST.LOCKS Display

The following table describes the column headings of the LIST.LOCKS display.

Column Heading	Description
UNO	The sequential number UniData assigns to the UniData session.
UNBR	Process group ID of the user setting the lock.
UID	User ID of the user setting the lock.
UNAME	Login name of the user setting the lock.
FILE NAME	The name of the file in which the record is locked. For resource locks, the word “semaphore” displays.
RECORD ID	Record ID of the locked record. For resource locks, the resource number displays.
M	Record lock mode.
TIME	The time at which the lock was set.
DATE	The date on which the lock was set.

LIST.LOCKS Display

LIST.PAUSED

Syntax

LIST.PAUSED

Synonym

LIST-PAUSED

Description

The ECL **LIST.PAUSED** command lists all processes that have been paused with the ECL PAUSE or UniBasic PAUSE command.

Example

The following example shows a typical LIST.PAUSED display. In the display, a hyphen (-) indicates that no timeout period has been specified for the pause:

```
:LIST.PAUSED
Number of Paused Users
~~~~~~~~~~~~~~~~~~~~~
5
UDTNO USRNBR UID USRNAME USRTYPE TTY LEFTTIME TOT_TIME
1 13656 1016 user1 udt pts/39 100 200
2 14430 1237 user2 udt pts/17 50 150
3 7484 1196 user3 udt pts/38 - -
```

LIST.PAUSED Display

The following table describes the column headings that display in the output for the LIST.PAUSED command.

Column Headings	Description
UNO	Sequential number UniData assigns to the UniData session.
UNBR	Process group ID of the paused session.
UID	User ID of the user whose session is paused.
USRNAME	Login name of the user whose session is paused.
USRTYPE	Type of session that is paused.
TTY	Terminal device of the user whose session is paused.
LEFTTIME	Number of seconds left until the process resumes.
TOT_TIME	Total number of seconds the process is paused.

LIST.PAUSED Display

Related Commands

UniData

LIST.PAUSED, PAUSE, WAKE

UniBasic Command

PAUSE, WAKE – For information, see the *UniBasic Commands Reference*.

LIST.QUEUE

Syntax

LIST.QUEUE [USERNAME *user_name* | FILENAME *filename* | *user_number*][DETAIL]

Synonym

LIST-QUEUE

Description

The ECL **LIST.QUEUE** command lists processes that currently waiting for locks. If a process is waiting for a lock, LIST.QUEUE displays information about the holder of the lock and processes waiting for the lock. Locks are set by each udt process through the general lock manager (GLM) module.

UniBasic commands that check for locks, such as READU and READVU, cause processes to wait for locks to be released before proceeding.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
USERNAME *user_name*	Lists all locks the user is waiting for. *user_name* is the operating system login name.

LIST.QUEUE Parameters

Parameter	Description
FILENAME *filename*	Lists all users waiting for locks for the file name you specify.
user_number	Lists all locks the *user_number* is waiting for. The user number can be found in the UNBR column of the LIST.READU and LIST.QUEUE output.
DETAIL	Displays a detailed listing.

LIST.QUEUE Parameters (continued)

Examples

The following example illustrates the output from the LIST.QUEUE command when you do not specify any parameters.

```
:LIST.QUEUE
FILENAME RECORD_ID M OWNER UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6031 2 pts/2 11:05:44 Aug 04
------------------------------------------------------------------
--------
FILENAME RECORD_ID M WAITING UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6130 4 ttyp1 11:05:54 Aug 04
INVENTORY 11060 X clair 6188 1 ttyp3 11:06:04 Aug 04
```

The next example illustrates the LIST.QUEUE output when you specify a user name:

```
:LIST.QUEUE USERNAME root
FILENAME RECORD_ID M OWNER UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6031 2 pts/2 11:35:46 Aug 04
------------------------------------------------------------------
--------
FILENAME RECORD_ID M WAITING UNBR UNO TTY TIME DATE
INVENTORY 11060 X root 6259 5 ttyp2 11:35:56 Aug 04
:
```

The next example illustrates the LIST.QUEUE command output when you specify a file name:

```
:LIST.QUEUE FILENAME INVENTORY
FILENAME RECORD_ID M OWNER UNBR UNO TTY TIME DATE
INVENTORY 11060 X root 6259 5 ttyp2 11:38:16 Aug 04
------------------------------------------------------------------
--------
FILENAME RECORD_ID M WAITING UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6188 1 ttyp3 11:38:36 Aug 04

INVENTORY 11060 X clair 6031 2 pts/2 11:38:46 Aug 04
:
```

The final example shows the output from the LIST.QUEUE command when you specify a user number:

```
:LIST.QUEUE 6763
FILENAME RECORD_ID M OWNER UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6758 5 pts/3 14:16:26 Aug 04
------------------------------------------------------------------
--------
FILENAME RECORD_ID M WAITING UNBR UNO TTY TIME DATE
INVENTORY 11060 X clair 6763 6 ttyp1 14:16:46 Aug 04
:
```

LIST.QUEUE Display

The LIST.QUEUE display in the previous examples use the default display. Information about the owner of the lock is listed above the line. Information about processes waiting for the lock is listed below the line, sorted by the date and time the process requested the lock.

The following table describes the column headings that display in the output for the LIST.QUEUE command for the owner of the lock.

Column Heading	Description
FILENAME	The name of the file holding the lock.
RECORD_ID	The record ID holding the lock.
M	The type of lock held. X is an exclusive lock, S is a shared lock.
OWNER	The user name of the owner of the lock.

LIST.QUEUE Owner Display

Column Heading	Description
UNBR	The process group ID (pid) of the user who set the lock.
UNO	The sequential number UniData assigns to the udt process for the owner of the lock.
TTY	The Terminal device of the user owning the lock.
TIME	The time the lock was set.
DATE	The date the lock was set.

LIST.QUEUE Owner Display (continued)

The next table describes the LIST.QUEUE column headings for the processes waiting for locks.

Column Heading	Description
FILENAME	The name of the file for which a lock is requested.
RECORD_ID	The record ID of the record for which a lock is requested.
M	The type of lock requested. X is an exclusive lock, S is a shared lock.
WAITING	The user name of the process waiting for a lock.
UNBR	The process ID (pid) of the user waiting for a lock.
UNO	The sequential number UniData assigns to the udt process waiting for a lock.
TTY	The terminal device of the user waiting for a lock.
TIME	The time the lock was requested.
DATE	The date the lock was requested.

LIST.QUEUE Waiting Display

The following example illustrates the LIST.QUEUE display when you specify the DETAIL option:

```
:LIST.QUEUE DETAIL
FILENAME RECORD_ID M INBR DNBR OWNER UNBR UNO TTY TIME DATE
INVENTORY 10060 X 241938 1073807361 clair 13798 3 pts/0 14:48:47
Nov 19
------------------------------------------------------------------
--------
FILENAME RECORD_ID M INBR DNBR WAITING UNBR UNO TTY TIME DATE
INVENTORY 10060 X 241938 1073807361 root 13763 1 ttyp2 14:48:57
Nov 19
```

The following table describes the column headings that display in the output for the LIST.QUEUE command when you specify the DETAIL option.

Column Heading	Description
FILENAME	The name of the file for which a lock is held.
RECORD_ID	The record ID of the record for which a lock is held.
M	The type of lock held. X is an exclusive lock, S is a shared lock.
INBR	The i-node of the file holding the lock.
DNBR	Used in conjunction with the INBR to define the file holding the lock at the operating system level.
OWNER	The user name of the process holding the lock.
UNBR	The process ID (pid) of the user holding a lock.
UNO	The sequential number UniData assigns to the udt process holding a lock.
TTY	The terminal device of the user holding a lock.
TIME	The time the lock was set.
DATE	The date the lock was set.

LIST.QUEUE Detail Display

The next table describes the column headings that display in the output for the LIST.QUEUE command when you specify the DETAIL option for processes waiting for locks.

Column Heading	Description
FILENAME	The name of the file for which a lock is requested.
RECORD_ID	The record ID of the record for which a lock is requested.
M	The type of lock held. X is an exclusive lock, S is a shared lock.
INBR	The i-node of the file for which a lock is requested.
DNBR	Used in conjunction with the INBR to define the file for which a lock is requested at the operating system level.
WAITING	The user name of the process requesting a lock.
UNBR	The process ID (pid) of the user requesting a lock.
UNO	The sequential number UniData assigns to the udt process requesting a lock.
TTY	The terminal device of the user requesting a lock.
TIME	The time at which the lock was requested.
DATE	The date on which the lock was requested.

LIST.QUEUE Detail Display

LIST.READU

Syntax

LIST.READU [*user_number* | ALL | FILENAME *filename* | USERNAME *user_name*] [DETAIL]

Synonym

LIST-READU

Description

The ECL LIST.READU command displays a list of file and record locks. You can display information about file and record locks by user number, user name, or file name, or you can display all READU locks.

Note: Use the GETUSER command to retrieve your user number. Execute LISTUSER to find out the user numbers for other users.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
user_number	Displays all locks held by the user number you specify.
ALL	Displays all currently active locks.
FILENAME *filename*	Displays all active locks associated with the file name you specify. If the file name does not reside in the current account, nothing is displayed.

LIST.READU Parameters

Parameter	Description
USERNAME *user_name*	Displays all active locks associated with the user name you specify.
DETAIL	Displays detailed information.
-N	Scrolls display of the list without pausing at the bottom of each page.

LIST.READU Parameters (continued)

Examples

The following example illustrates the output from the LIST.READU command when you do not specify any options:

```
:LIST.READU
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
4 6739 0 root ttyp5 INVENTOR 24193 10738 11000 X 16:22:13 Aug 04
5 6758 1172 clair pts/3 INVENTOR 24193 10738 10060 X 16:21:53 Aug 04
:
```

The next example illustrates the output from the LIST.READU command when you specify a user number. The user number can be found in the output from the LIST.QUEUE and LIST.READU commands under the UNBR column.

```
:LIST.READU 6739
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE

4 6739 0 root ttyp5 INVENTOR 24193 10738 11000 X 16:25:44 Aug 04
:
```

The next example illustrates output from the LIST.READU command when you specify a user name:

```
:LIST.READU USERNAME claireg
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
5 6758 1172 clair pts/3 INVENTOR 24193 10738 11060 X 16:28:14 Aug 04
:
```

The final example illustrates output from the LIST.READU command when you specify a file name:

```
:LIST.READU FILENAME INVENTORY
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
4 6739 0 root ttyp5 INVENTOR 24193 10738 11000 X 16:28:24 Aug 04
5 6758 1172 clair pts/3 INVENTOR 24193 10738 11060 X 16:28:14 Aug 04
:
```

LIST.READU Column Headings

The following table describes the column headings of the LIST.READU display.

Column Heading	Description
UNO	The sequential number UniData assigns to the udt process that set the lock.
UNBR	The process ID of the user who set the lock.
UID	The user ID of the user who set the lock.
UNAME	The login name of the user who set the lock.
TTY	The terminal device of the user who set the lock.
FILENAME	The file name in which the record is locked.
INBR	The i-node of the locked file.
DNBR	Used in conjunction with INBR to define the file at the operating system level.
RECORD_ID	The record ID of the locked record.
M	The type of lock. X indicates an exclusive lock. S indicates a shared lock.
TIME	The time at which the lock was set.
DATE	The date on which the lock was set.

LIST.READU Column Headings

LIST.TRIGGER

Syntax

LIST.TRIGGER [DATA | DICT] *filename*

Synonym

LIST-TRIGGER

Description

The ECL **LIST.TRIGGER** command displays a list of triggers.

For more information about triggers, see *Developing UniBasic Applications*.

Note: UniData triggers monitor the update or deletion of records in UniData files. When a trigger is present and a user attempts to update or delete records in the file, the trigger executes a user-defined, globally cataloged, UniBasic subroutine.

Parameters

The following tables describes each parameter of the syntax.

Parameter	Description
filename	A UniData file name.
DATA	Lists triggers associated with the data file. This is the default behavior.
DICT	Lists triggers associated with the dictionary file.

LIST.TRIGGER Parameters

Example

The following example shows how UniData displays trigger information with the LIST.TRIGGER command:

```
:LIST.TRIGGER ORDERS
BEFORE UPDATE TRIGGER: DEMO_RTN
BEFORE DELETE TRIGGER: not defined
:
```

Related Commands

CREATE.TRIGGER, DELETE.TRIGGER

LIST.USERSTATS

Syntax

LIST.USERSTATS

Description

The **LIST.USERSTATS** command displays statistics of UniData activities. If you have issued the ENABLE.USERSTATS command, UniData displays statistics for your process only. If you have not issued the ENABLE.USERSTATS command, UniData displays statistics collected for all UniData processes since UniData was started.

Example

The following example illustrates the output from the LIST.USERSTATS command:

```
:LIST.USERSTATS
File I/O Statistics
Physical File Opens........ 0
File Closes................ 0
Temp File Closes........... 0
Dynamic File Split......... 0
Dynamic File Merge......... 0
Record Reads............... 12
Record Writes.............. 0
Record Deletes............. 0
Level 1 Overflow........... 0
Level 2 Overflow........... 0
Program Control Statistics
Private Code Calls......... 0
Shared Code Calls.......... 0
Shared Code Failures....... 0
CALLC Calls................ 0
Chain Calls................ 0
Gosub Calls................ 0
Goto Calls................. 0
Execute Calls.............. 0
Pcperform Calls............ 0
Dynamic Array Statistics
DELETE..................... 0
FIND....................... 0
INSERT..................... 0
LOCATE..................... 0
MATPARSE................... 0
MATCHFIELD................. 0
COUNT...................... 0
EXTRACT.................... 0
FIELD...................... 0
REMOVE..................... 0
REPLACE.................... 0
INDEX...................... 0
Lock Statistics
Record Locks............... 0
Record Unlocks............. 0
Semaphore Locks............ 0
Semaphore Unlocks.......... 0
Shared Group Locks......... 24
Exclusive Group Locks...... 0
Shared Index Locks......... 0
Exclusive Index Locks...... 0
Lock Failures.............. 0
Index Statistics
Index Reads................ 0
Index Writes............... 0
```

```
Log Reads.................. 0
Log Writes................. 0
Node Merges................ 0
Node Split................. 0
Node Reuse................. 0
Overflow Reads............. 0
Overflow Writes............ 0
:
```

LISTPEQS

Syntax

LISTPEQS

Synonym

SP-LISTQ

Description

The ECL **LISTPEQS** command lists the status of all requests made to the system printer by the requesting process. This command operates like the UNIX lpstat command. If the print queue for the process is empty, UniData returns to the ECL prompt.

For more information about lpstat, see your UNIX system documentation.

Note: LISTPEQS is supported on UniData for UNIX only.

LISTPTR

Syntax

LISTPTR

Description

The ECL **LISTPTR** command displays the printers defined for your system.

Examples

The following example displays printers defined for a UNIX system:

```
:LISTPTR
device for hpzone4: /dev/null
device for hpzone3: /dev/null
device for parallel: /dev/c1t0d0_l
:
```

The next example displays printers defined for a Windows platform:

```
:LISTPTR
Unit.. Printer..................
Port......................Status..
0 \\DENVER4\hpzone3 hpzone3 Running
1 LEGAL \\DENVER4\hpzone4 Running
:
```

LISTUSER

Syntax

LISTUSER

listuser

Description

The ECL **LISTUSER** command and the system-level listuser command display the number of users licensed for your installation and a list of the UniData processes currently running.

In the event a UniData user session aborts through a power failure or other abnormal circumstance, UniData registers the aborted process as an active user, and it appears as such in the LISTUSER display. Eventually, the cleanupd daemon will detect these processes and remove the aborted process from the user list.

Note: Phantom processes do not count against the number of UniData licenses.

Tip: To remove aborted processes that register as active users, use the system-level deleteuser command. For more information about deleteuser, see Administering UniData.

Example (UniData for UNIX)

The following example displays users for which UniData is licensed and number currently active:

```
:LISTUSER
Max Number of Users UDT SQL TOTAL

~~~~~~~~~~~~~~~~~~ ~~~ ~~~ ~~~~~
32 3 0 3
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
1 27398 1210 amyc udt 13:32:18 Jul 23 1999
4 27286 1172 claireg udt pts/1 09:45:04 Jul 23 1999
5 27319 1283 carolw udt pts/2 10:12:10 Jul 23 1999
:
```

LISTUSER Display

The following table describes the column headings in the LISTUSER display.

Parameter	Description
UDTNO	Sequential number UniData assigns to each user.
USRNBR	System-level process ID (pid) assigned to a UniData session.
UID	System-level ID assigned to a user.
USRNAME	Login name of the user.
USRTYPE	Type of process the user is running.
TTY	Device ID.
TIME	Time the user process started.
DATE	Date the user process started.

LISTUSER Display

Example (UniData for Windows Platforms)

LISTUSER output on UniData for Windows Platforms is shown in the following example:

```
:LISTUSER
Max Number of Users UDT SQL TOTAL
~~~~~~~~~~~~~~~~~~~ ~~~ ~~~ ~~~~~
16 4 0 4
UDTNO USRNBR UID USRNAME USRTYPE TTY IP-ADDRESS TIME DATE
1 131 1404 claireg udt pts/1 Console 14:34:02 Jul 22 1999
2 122 500 Administ udt pts/2 192.245.122.28 14:41:37 Jul 22 1999
3 98 1001 USER01 udt pts/3 192.245.122.28 15:24:17 Jul 22 1999
4 156 1404 claireg udt pts/4 Console 15:18:11 Jul 22 1999
5 154 500 Administ phantom pts/5 Console 15:30:43 Jul 22 1999
:
```

LISTUSER Display Attributes

The following table lists the LISTUSER command display attributes.

Parameter	Description
UDTNO	Sequential number UniData assigns to each user.
USRNBR	Process ID of the UniData session.
UID	Windows ID of the user.
USRNAME	Login name of the user.
USRTYPE	Type of process the user is running.
TTY	Session identifier, formed by concatenating the string “pts/” and the UDTNO.
IP-ADDRESS	Location where the session is logged in; either “Console” or a valid IP address.
TIME	The time at which the user process started.
DATE	The date on which the user process started.

LISTUSER Display Attributes

Related Command

GETUSER

LO

LO is a synonym for the BYE command. For more information, see BYE.

Synonyms

BYE, QUIT

LOCK

Syntax

LOCK *resource* [NO.WAIT]

Description

The ECL **LOCK** command reserves a resource for exclusive use by your process.

If you do not use the NO.WAIT keyword, your process waits until the resource has been released.

Note: A UniData resource lock behaves like a system-level semaphore lock.

To release a lock set by your process, execute CLEAR.LOCKS or SUPERCLEAR.LOCKS. Resource locks are automatically released when the user session ends.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
resource	A number, from 0 to 63, inclusive, that identifies the resource to be reserved. UniData can identify 64 resources.
NO.WAIT	Your process returns to ECL if the resource is locked, without waiting for the resource to become available.

LOCK Parameters

Example (UniData for UNIX)

In the following example, the LOCK command reserves resource 2. Then, LIST.LOCKS lists the current system resource locks:

```
:LOCK 2
:LIST.LOCKS
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
5 27319 1283carolw ts/2 semaphor -1 0 2 X 13:54:49 Jul 2:
```

Example (UniData for Windows Platforms)

In the following example, the LOCK command reserves resource 2. Then, LIST.LOCKS lists the current system resource locks:

```
:LOCK 2
:LIST.LOCKS
UNO UNBR UID UNAME TTY FILENAME RECORD_ID M TIME DATE
1 251 1049668 claireg Console semaphore 1 X 19:14:44 Nov 03
```

Related Commands

CLEAR.LOCKS, LIST.LOCKS, SUPERCLEAR.LOCKS

log_install

Syntax

log_install [-l | -a | -h]

Description

The system-level **log_install** command initializes the Recoverable File Systems log files and archive files using information from the log configuration table and the archive configuration table. When you use this command, the UniData daemons must not be running. For more information about this command and recoverable files, see *Administering the Recoverable File System*.

To use this command, you must log in as root.

Tip: We recommend that you run cntl_install, which invokes log_install, rather than executing log_install directly.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-l	Default. Initializes log files only.
-a	Initializes both archive files and log files. If you include this option when the archiving system is not enabled, only the log configuration table gets installed. **Tip**: To enable archiving, set the ARCH_FLAG parameter in the UniData configuration file to any positive integer.
-h	Displays online help for log_install.

log_install Parameters

Example

The following example illustrates the log_install command with the -a option:

```
# log_install -a
WARNING: log_install will replace your log files, if they exist,
without
making a backup copy. Do not run log_install unless you are
certain you
no longer need your earlier log files for recovery.
Do you want to continue? (Y/N) [n]
y
.........
#
```

Related Command

cntl_install

LOGTO

Syntax

LOGTO *account*

Description

The ECL **LOGTO** command changes the current process to another account. account must exist in the directory udthome on the home file system, or you must provide the full path to account.The LOGOUT paragraph is not executed when you log to another account.

Note: Ordinarily, whenever you change to an account, UniData executes the login paragraph for that account unless you are logged in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms. Set UDT.OPTIONS 20 to on to remove this exception. (With UDT.OPTIONS 20 on, UniData executes the login paragraph when a root or Administrator user switches accounts.)

Tip: On UniData for UNIX, execute UNIX ln -s in udthome to create a symbolic link. This enables you to distribute accounts over multiple file systems while still using LOGTO.

Examples

In the following example, the user executes the LOGTO command to switch to the UniData demo database account. The ECL WHERE command that precedes and follows the example displays the current account. These examples are taken from UniData for UNIX. On UniData for Windows Platforms, the path contains the backslash.

```
:WHERE
/home/carolw/demo
:LOGTO demo
:WHERE
/users/ud60/demo
:
```

You can return to the original account with the LOGTO command, as shown in the following example:

```
:WHERE
/disk1/ud60/demo
:LOGTO /home/carolw/demo
:WHERE
/home/carolw/demo
:
```

LS

Syntax

LS [*path*]

Description

The ECL **LS** command displays the files that reside in the current account or in path. path may be a DIR-type file or a file pointer (F-type).

Examples

The following example shows an LS command display for the current account:

```
:LS
BP D_CLIENTS D_STATES INVENTORY _HOLD_
BP_SOURCE D_COURSES D_STUDENT MENUFILE _PH_
CATEGORIES D_CTLG D_TAPES ORDERS _REPORT_
CLIENTS D_CUSTOMER D_VOC PARAGRAPHS _SCREEN_
COURSES D_INVENTORY D__HOLD_ SAVEDLISTS __V__VIEW
CTLG D_MENUFILE D__PH_ STAFF savedlists
CUSTOMER D_ORDERS D__REPORT_ STATES vocupgrade
D_BP D_PARAGRAPHS D__SCREEN_ STUDENT
D_BP_SOURCE D_SAVEDLISTS D___V__VIEW TAPES
D_CATEGORIES D_STAFF D_savedlists VO
:
```

The next example shows output when LS is executed against a dynamic hashed file in the UniData demo database. This file is in an overflow state, and at least one index exists for this file:

```
:LS ORDERS
dat001

idx001
over001
```

Related Command

LSL

LSL

Syntax

LSL

Description

The ECL **LSL** command displays a long listing of all of the files in a UniData account.

On UniData for UNIX, the first line of this report is the total number of files in the account. Subsequent lines list the files and subdirectories on the first level of the account. On UniData for Windows Platforms, LSL executes the MS-DOS dir command. LSL does not list files in subdirectories.

Example

The following example shows an LSL display on UniData for UNIX:

```
:LSL
total 570
drwxrwxrwx 2 root sys 24 Jul 11 16:17 BP
drwxrwxrwx 2 root sys 1024 Jul 17 10:06 BP_SOURCE
-rw-rw-rw- 1 root sys 4096 Jul 11 16:17 CATEGORIES
-rw-rw-rw- 1 root sys 21504 Jul 11 16:17 CLIENTS
-rw-rw-rw- 1 root sys 4096 Jul 11 16:17 COURSES
drwxrwxrwx 2 root sys 24 Jul 11 16:17 CTLG
-rw-rw-rw- 1 root sys 4096 Jul 11 16:17 CUSTOMER
-rw-rw-rw- 1 root sys 2048 Jul 11 16:17 D_BP
-rw-rw-rw- 1 root sys 2048 Jul 11 16:17 D_BP_SOURCE
-rw-rw-rw- 1 root sys 2048 Jul 11 16:17 D_CATEGORIES
-rw-rw-rw- 1 root sys 2048 Jul 11 16:17 D_CLIENTS
-rw-rw-rw- 1 root sys 2048 Jul 11 16:17 D_COURSES
...
```

Related Command

LS

lstt

Syntax

lstt [-l *n* | -L*pid*]

Description

The system-level **lstt** command displays details about local control tables (LCTs) in shared memory. See *Administering UniData* for more information about shared memory and LCTs.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-l *n*	Displays additional information about a designated local control table identified by *n*, a local control table.
-L *pid*	Displays additional information about a local control table identified by a *pid*, (a system-level process identification number of a group leader).

lstt Parameters

Example

The following example shows general statistical information about all LCTs on a system:

```
% lstt
---------------------- LCTs Statistics ----------------------
Total LCTs (Process Groups allowed): 40

LCTs Used (Active Process Groups): 5 (12% of 40) Total Ps: 10
Total Global Pages Used: 12 (1536K bytes)
Total Self-created.....: 0 (0K bytes)
Total memory used......: 1536K bytes
-------------------- End of LCTs Statistics -------------------
:
```

Related Commands

gstt, sms

MAG_RESTORE

Syntax

MAG_RESTORE [-D] [-E] [-G | GB] [-GC]
[-H[DYNAMIC0 | DYNAMIC1]]
[-O] [-S] [-U [0-9]] [-M [0-3]]
[-X *char_list*][-Kn][-A *outputfile*][-C *filename*][-B *outputfile*][-T *directory*]
[-R ALL | *filelist*] [-L [0-9]]
[*acct_name*]

Description

The system-level **MAG_RESTORE** command restores a PRIME® account that was saved to tape with the PRIME MAGSAV command with REV19, NO_ACL, on the same level as the User File Directory (UFD). For each MAGSAV, only one logical volume may be included. MAG_RESTORE restores accounts, with their original names, to the current directory. If UniData cannot read a name from the tape, it uses acct_name.Ifacct_name is the name of an account that does not exist in the current directory, UniData executes the newacct command to create a new one. When multiple accounts exist on a single save, UniData prompts for owner and group for each account.

PRIME ® dynamic files are restored as UniData dynamic files. Hash type 0 is assigned if -HDYNAMIC is not specified.

MAGSAV saves in variable-length blocks. UniData reads the tape as a single block, or reads the first six blocks to determine block size.

Tip: If you have saved very large data files (larger than 1 gigabyte) from PRIME ® , we recommend that you create the target UniData files as dynamic before you begin the restore. Assign a modulo to accommodate a file about 40 percent larger than the original PRIME file. (When converting PRIME ® files larger than 1.5 gigabytes, the UniData dynamic files created are approximately 40 percent larger.)

Note: Execute this command at the operating system prompt.

Parameters

The following table lists the MAG_RESTORE parameters.

Parameter	Description
-D	Overwrites hashed files in an existing account with files from tape, but does not create new files. Does not restore dictionary files.
-E	Clears each file on disk.
-GB	MAGSAV writes data in variable-length blocks. However, when a tape is copied with the UNIX dd command, data is written onto the new tape in fixed-length blocks. -GB reads a backup tape created in this way.
-GC	Reads PRIME 2350 (60-mb cartridge) tapes. -GC is valid for UniData releases after 2.2.2.
-HDYNAMIC0	Converts all restored files to dynamic with hash type 0.
-HDYNAMIC1	Converts all restored files to dynamic with hash type 1.
-O	Overwrites all data in the account, including that in dictionary and DIR-type files, from tape. The files must already exist in the current directory. **Note**: Execute MAG_RESTORE -C to create the files on disk before executing MAG_RESTORE -O to populate them.
-S	Truncates file names to 12 characters in length. This parameter is not necessary if you run MAG_RESTORE on an operating system that automatically shortens file and program names.
-U [0-9]	Indicates a tape unit from which to read. The tape unit must be described in the tapeinfo file in *udthome*/sys. Default is 0. UniData reserves unit 9 for disk image. **Tip**: Use the SETTAPE command first to set the tape unit.
-M [0-3]	Converts data based on one of the following options: ■ 0 – Default. No conversion. Data is assumed to be ASCII. ■ 1 – EBCDIC conversion. ■ 2 – Invert high bit. ■ 3 – Swap bytes.

MAG_RESTORE Parameters

Parameter	Description
-X *char_list*	*char_list* indicates characters to be considered invalid for: ■ file names ■ account names ■ record IDs in DIR-type files ■ While restoring, UniData converts these characters to underscore (_). If the resulting name conflicts with an existing account name, UniData adds a character to the end of the name to make it unique. For example: A&B becomes A_B. If A_B is used by another file, the name become A_Ba. ■ Default invalid characters are the following: space * ? / & ‘. ■ You cannot specify nonprinting characters as invalid. Do not separate characters in *char_list* with spaces or commas.
-K *n*	Defines the size of the internal memory buffer (in kilobytes). Default size is 8000 K. System restoration performs best when buffer size is large. Change the size to match the capacity of your operating system.
-A *filename*	Creates *filename*, an ASCII text file, in the current directory, containing statistics about each file on the tape. -A does not restore files. See “Preparing for Restoration” following this table.
-B *outputfile*	Adjusts the modulo or block size for *outputfile*. The list should contain a line entry for each file. To adjust these elements, format the entries as in this example: *file1*, 1, 203 *file2*, 4, 101 *file3*, 3, -1 *file4*, -1, 11 **Note**: “-1” tells MAG_RESTORE to keep the original modulo or block size multiplier.
-C *filename*	Reads the file created by a previous execution of MAG_RESTORE with the -A *filename* option. Creates, in the current directory, the files listed in *filename*, but does not restore data.

MAG_RESTORE Parameters (continued)

Parameter	Description
-T	Separates the working directory and the target directory. Optionally places the working directory on RAM-DISK to improve system performance. RAM-DISK has a faster I/O speed but less disk space. Optionally places the target directory on another system through the Network File System (NFS) to overcome disk shortage.
-R *filelist* | ALL	Restores both data and dictionary portions of files listed in *filelist*. You create filelist, an ASCII file containing a single-line entry for each file to be ignored. The syntax for each line is as follows: *PRIME_filename* Use the ALL keyword to load all of the files that are on the tape but are not currently in the account.
-L [0-9]	Adjusts the file pointer position. -L can restore the account to any directory level. Each directory occupies 48 bytes. Use the following numeric indicators to set the file pointer to the correct directory in the path: ■ 0 – MAGSAV executed in the account's own directory. ■ 1 – Default. MAGSAV executed at a directory level higher than the account. ■ 2 - 9 – Supports nested accounts. **Tip**: Before you use MAGSAV on PRIME® accounts that you intend to restore with MAG_RESTORE, be certain the PRIME® accounts are on the same directory level with the User File Directory (UFD).
acct_name	New name fro the restored account to be used if UniData cannot obtain a name from the account on tape.

MAG_RESTORE Parameters (continued)

Preparing for Restoration

We recommend that you the follow this procedure to make the restoration more efficient. Use the -A parameter in conjunction with -C and -O to determine file status before files are loaded. This decreases load time, because UniData then does not have to resize files during restoration.

1. 1. Execute MAG_RESTORE -A *filename* to generate a file containing statistics about the files on tape. Use these statistics to evaluate the suitability of the projected modulo, file type, and file separation.

   filename is stored in the current directory. For each file, UniData lists the following on a single line separated by commas:

   - The position of the file on the tape.
   - The type of UniData file.
   - The name of the UniData file.
   - The file separation.
   - The original modulo of the file on tape Informix recommends a modulo based on the number of records and the size of the file. This recommended modulo is never smaller than the original modulo.
   - The proposed key length.
   - The total record length for the file.
   - The number of records in the UniData file.

2. 2. Use an ASCII text editor to modify the file generated in Step 1 as desired. For example, you might eliminate files from the list that you do not want UniData to restore.
3. 3. Execute MAG_RESTORE -C filename to create new UniData files in the destination directory. Remember, filename must be the name of the file created in Step 1. Add other parameters as desired.
4. 4. Execute MAG_RESTORE -O filename to load the data and dictionary records into the files created in Step 3. Add other parameters as desired.

UniData may display any of the following messages during the restore.

Message	Description
Create file modulo separator [---newfile]	UniData is loading the file using the modulo and block size multiplier found on the tape. If the file name contains invalid characters or is too long, UniData changes its name to “newfile.”
DUMP_MD	UniData is reading an MD file.
DICT	UniData is reading a dictionary file.

MAG_RESTORE Messages

Message	Description
DATA	UniData is reading a single-level hashed data file.
DIR	UniData is reading a single-level sequential file.
LF	UniData is reading a multi-level hashed data file.
LD	UniData is reading a multi-level sequential file.
Loading (filename) ...	UniData is loading the data into existing files rather than creating files. This is the default when you run MAG_RESTORE with the -D or -O option.
Replace to multi-level success	A single-level file changed to a multi-level file.
Replace to multi-level failure	UniData failed to change a single-level file into a multi-level file.
Resize (filename) to new modulo --- (modulo)	The file called *filename* has an inadequate modulo; UniData resized the file to a more efficient modulo (*modulo*).
Create file failure	UniData failed to create the file.
Open file failure	UniData failed to open the file.

MAG_RESTORE Messages (continued)

Files Created by MAG_RESTORE

MAG_RESTORE creates the following output files during the restore.

File Name	Description
DUMP_VOC	Hashed file. VOC in PRIME® systems and Pick® systems.
pgm_map	Hashed file. Lists long file names changed to short file names.

MAG_RESTORE Output Files

File Name	Description
dispmsg	Text file. Saves screen display messages including error and dump messages displayed at end-of-reel. UniData saves the first 70 characters displayed.
resize_list	Text file. Lists the names of files that need to be resized.
idx_list	Text file. Saves index information on the account.

MAG_RESTORE Output Files (continued)

MAKE.MAP.FILE

Syntax

MAKE.MAP.FILE

Synonym

MAKE-MAP-FILE

Description

The ECL **MAKE.MAP.FILE** command rebuilds the _MAP_ file, which contains information on globally cataloged UniBasic programs. _MAP_ is located in *udthome*/sys on UniData for UNIX or *udthome*\sys on UniData for Windows Platforms.

This command does the following:

- Clears _MAP_
- Executes SELECT CTLGTB (global catalog space) and, for each key in the select list, verifies that the file still exists in *udthome*/sys/CTLG/x on UniData for UNIX or *udthome*\sys\CTLG\x on UniData for Windows Platforms. If it does, UniData writes a record for it in the _MAP_ file.

Tip: Use the UniQuery LIST or ECL MAP command to view the contents of the _MAP_ file.

Related Command

MAP

makeudapi

Syntax

makeudapi

Description

The system-level **makeudapi** command builds a new UniData executable (udapi_slave) with links to C programs so that they are accessible through InterCall, UniObjects, or UniObjects for Java.

Note: This command is supported on UniData for UNIX only.

The command reads the following files:

- **base.mk** – This is a version of the make file, and is located in *udthome*/work. UniData uses base.mk as a template for creating new.mk, then executes new.mk to create the new udapi_slave executable.
- **cfuncdef** – This function definition file is also located in *udthome*/work. It contains definitions for C functions that UniData has incorporated into the current release of UniData. Do not modify this file.
- **cfuncdef_user** – This file contains definitions for site-specific C functions that you want to link into InterCall, UniObjects, or UniObjects for Java.
- UniData Libraries – When you install UniData, you are prompted for the path where you want to locate these.

Note: It is best to log in as root to execute makeudapi. UniData may be up and running, and users may be logged in. However, if users are logged in, the makeudapi command may not allow you to overwrite the production udapi_slave, depending on your operating system. Some operating systems display an error message and exit, while others prompt you to decide whether you want to overwrite the production udapi_slave. If the production version is not overlaid, you must manually copy it.

Related Command

makeudt

makeudt

Syntax

makeudt [-n nfa]

Description

The system-level **makeudt** command builds a new UniData executable (udt).

Note: This command is supported on UniData for UNIX only.

The command reads the following files:

- **base.mk** – This is a version of the make file, and is located in *udthome*/work. UniData uses base.mk as a template for creating new.mk, then executes new.mk to create the new udt executable.
- **cfuncdef** – This function definition file is also located in *udthome*/work. It contains definitions for C functions that UniData has incorporated into the current release of UniData. Do not modify this file.
- **cfuncdef_user** – This file contains definitions for site-specific C functions that you want to link into UniData.
- **UniData Libraries** – When you install UniData, you are prompted for the path where you want to locate these.

For detailed information about building a UniData executable, see *Administering UniData* or *Developing UniBasic Applications*.

Note: It is best to log in as root to execute makeudt. UniData may be up and running, and users may be logged in. However, if users are logged in, the makeudt command may not allow you to overwrite the production udt, depending on your operating system. Some operating systems display an error message and exit, while others prompt you to decide whether you want to overwrite the production udt. If the production version is not overlaid, you must manually copy it.

Parameters

The following table describes the parameters of the syntax.

Parameter	Description
-n nfa	Use this option only if you are **not** using UniData OFS / NFA. This option uses "dummy" libraries rather than network libraries required by NFA. Software development environments may or may not include the network libraries; if yours does not include these, and you do not use the -n nfa option, makeudt fails.

makeudt Parameter

MAP

Syntax

MAP

Description

The ECL **MAP** command rebuilds the _MAP_ file and displays its contents on the terminal screen. The _MAP_ file, located in *udthome*/sys on UniData for UNIX or in udthome\sys on UniData for Windows Platforms, contains information about globally cataloged UniBasic programs.

Tip: You can also use the UniQuery LIST command to view the contents of _MAP_, for example, LIST _MAP_ ALL.

For more information on UniBasic programs, see *Developing UniBasic Applications*.For more information on catalog space, see *Administering UniData.*

Example

In the following example, UniData rebuilds and displays the contents of the _MAP_ file to the terminal screen:

```
:MAP
MAP 09:15:33 Jun 23 1999 1
NAME..... TYPE ARG ORIGINATOR..... WHO.... OBJ... DATE.... LAST
REF
508E S 41 @UDTHOME/SYS_BP 508E root 184 05/15/99 05/15/99
COUNT.MSG S 31 @UDTHOME/DENAT_BP CO root 582 05/15/99 05/15/99
UNT.MSG
SORT_AE S 11 @UDTHOME/AE_BP SORT_ root 1650 05/15/99 05/15/99

AE
7201 S 41 @UDTHOME/SYS_BP 7201 root 180 05/15/99 05/15/99
NFA.EXECSEL.U S 31 @UDTHOME/SYS_BP NFA. root 154 05/15/99 05/15/99
EXECSEL.U
S_VALID_FILE_CHE S 61 @UDTHOME/SYS_BP S_VA root 1712 05/15/99 05/15/99
CK LID_FILE_CHECK
.
.
.
```

MAP Display

The following table describes the column headings that display in the output for the MAP command.

Column Heading	Description
NAME	Name of the cataloged program.
TYPE	Type of the cataloged program: ■ M – Main program ■ S – Subroutine
ARG	Number of parameters in the call.
ORIGINATOR	Full path to the file where the program was cataloged.
WHO	Login name of the user who cataloged the program.
OBJ	Size of the object code in bytes.
DATE	Date the program was cataloged.
LAST REF	Date the program was last accessed.

MAP Display

Related Command

MAKE.MAP.FILE

MAX.USER

Syntax

MAX.USER *number*

Synonym

MAX-USER

Description

The ECL **MAX.USER** command determines the maximum number of users who can log into UniData. If MAX.USER is less than the number of users currently logged in, UniData does not force current users to log out.

After stopping and starting UniData (stopud and startud), the number of users is reestablished to the number licensed. To reset to this number without stopping and restarting UniData, use MAX.USER with the correct number, or -1.

If you set MAX.USER to 0 (zero) and exit UniData, you will have to restart the daemons to start UniData again.

Note: To execute MAX.USER, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Tip: Use MAX.USER for limiting the number of users on the system during system maintenance.

mediarec

Syntax

mediarec [-s [*MM:DD:YY:*]*HH:MM*[:*SS*]] [-e [*MM:DD:YY:*] *HH:MM* [:*SS*]] [-f *path/filename*][-T *start_LSN*[,*end_LSN*]]

Description

The **mediarec** command restores changes to your recoverable files by applying archives since the last backup.

Parameters

The following table describes the parameters for the syntax.

Parameter	Description
[-s]	Specifies the recovery start time. If you do not use the -s parameter, the whole archive set (from the last backup to current) is recovered.
[-e]	Specifies the recovery end time. If you do not use the -e parameter, the whole archive set (from the last backup to current) is recovered.
[-f]	Specifies a file that contains a list of files (one path and file name per line) to recover. If you do not use the -f parameter, mediarec recovers all file.s
[-T]	Specifies the starting LSN and the ending LSN for media recovery. If you only specify the starting LSN, mediarec will prompt for the next sequential LSN.

mediarec Parameters

Example

In the following example, the mediarec command restores a database:

```
Screen Example
#mediarec
Using UDTBIN=/usr/ud60/bin
For media recovery, you would be required to have space for two
temporary files, one to hold the largest archive file and another
to hold the largest CP size. Please note the following info,
read documentation about media recovery procedure and re-start
media recovery.
Max CP Size (in bytes): 54272
Max Arch File Size (in bytes): 4218880
Also, if youre planning to use the tape(s) created by archive
process, please setup restore script
/usr/ud51/include/arch_restore properly
(tape device) and load the first archive tape.
Do you want to continue?(y/n)[n]
All output and error logs have been saved
to /usr/ud52/bin/saved_logs directory.
SMM is started.
Starting media recovery... Please wait.
For media recovery, youll be asked to upload
archive files one by one by sequence number into
the /usr/ARCH file.
de_arch: reading archive file on disk
The file TEST may have been deleted at OS level
If you choose to not re-create this file now,
the Media Recovery will be aborted to keep
the system transaction consistent.
Would you like it re-created? (y/n) [y]y
Deleting file D_TEST.
Deleting file TEST.
Create file D_TEST, modulo/1,blocksize/1024
Hash type = 0

Create dynamic file TEST, modulo/5,blocksize/1024
Hash type = 1
Added “@ID”, the default record for UniData to DICT TEST.
....
Please check /usr/ud60/FileInfo for un-recovered file level
operations.
*****!!! Media Recovery Finished!!!*****
SM stopped successfully.
SMM stopped successfully.
Media Recovery finished.
Please use /usr/ud60/bin/startud to start the system
```

memresize

Syntax

memresize [DICT] *filename* [*modulo* [,*block.size.multiplier*]] [TMPPATH *pathname*] [TYPE {0 | 1}] [MEMORY *buf_size*] [RESTORE] [STATIC | [DYNAMIC] [KEYONLY | KEYDATA][PARTTBL *part_tbl*]] [NOPROMPT]

Description

The system-level **memresize** command resizes a hashed file in size, modulo, block size, or hashing algorithm. memresize also converts between static and dynamic hashed files and changes the split / merge type and the part table for dynamic files. memresize operates in an internal memory buffer and writes to disk only when the buffer becomes full or when the memresize operation completes.

Parameters

The following table describes the parameters of the syntax.

Parameter	Description
DICT	Resizes the dictionary portion of *filename*.
filename	The name of the file to be resized.
modulo	The new modulo number to be assigned to the file.
block.size.multiplier	An integer between 0 and 16 that UniData uses to determine file size. See "Estimating the File Size" in the CREATE.FILE command for more information about sizing files.

memresize Parameters

Parameter	Description
TMPPATH *pathname*	The path where UniData locates a working copy of the file during resizing. The default is /tmp on UniData for UNIX or \TEMP on UniData for Windows Platforms. This parameter has no effect if the resulting file is a dynamic file.
TYPE {0 | 1}	Hash type for the resized file.
MEMORY *buf_size*	Size in kilobytes of memory buffer used for the operation. memresize may perform faster with a larger memory allocation. The minimum size is 256K. The default on most systems is 8000K (8 MB). You can assign as much memory as is available on your system. For example, 12000000 assigns 12 MB of memory to the memresize command.
RESTORE	Skip over file corruption that cannot be fixed, but continue resizing the file. Use this parameter when a file must be restored regardless of corruption.
STATIC	After resizing, the file is a static hashed file.
DYNAMIC	After resizing, the file is a dynamic hashed file.
KEYDATA	After resizing, the file is dynamic and the split/merge type is KEYDATA.
KEYONLY	After resizing the file is dynamic and the split/merge type is KEYONLY (the default).
PARTTBL,*part_tbl*	After resizing, file is a dynamic file. *part_tbl* is the path and file name of a previously established part table. memresize copies *part_tbl* into the dynamic file directory. The copy of *part_tbl* in the dynamic file directory serves as the "per-file" part table for the dynamic file. **Note**: This option is supported on UniData for UNIX only.
NOPROMPT	If you specify this parameter, memresize does not prompt you to free disk space if it encounters a file system full. memresize removes the temporary file that was under construction and quits, leaving the original, live file untouched. UniData displays messages to the screen.

memresize Parameters (continued)

Additional Information

Notice the following points about memresize options:

- Specifying DYNAMIC, KEYONLY, KEYDATA, or PARTTBL on the command line causes the resized file to be dynamic.
- The DICT option is invalid if combined with any of the DYNAMIC options.
- You cannot convert UniData system files (for instance, a VOC file or the ERRMSG file) into a dynamic file. memresize reports an error and fails.
- The TMPPATH option is invalid if any DYNAMIC options are specified (or if the starting file is dynamic and no file type options are specified).
- If the starting file is recoverable, the resized file is recoverable. If the starting file is nonrecoverable, the resized file is nonrecoverable.
- If the starting file has an index, memresize uses the following logic to handle index related files:
  - If both the starting file and the resulting file are STATIC, leave the index file and index log file unchanged.
  - If the starting file is STATIC and the resulting file is DYNAMIC, copy the index file to idx001 and the index log file (if it exists) to xlog001 in the dynamic file directory.
  - If the starting file is DYNAMIC, and the resulting file is STATIC, and the starting file has only one index part file (idx001) and no more than one index log file (xlog001), copy idx001 to X_*filename* and xlog001 (if it exists) to x_*filename* on UniData for UNIX or L_*filename* on UniData for Windows Platforms in the account directory.
  - If the starting file is DYNAMIC and the resulting file is STATIC, and the starting file has more than one index part file, do not process the index or index log files. Display a message directing the user to re-create and rebuild the indexes.
  - If both the starting file and the resulting file are DYNAMIC, simply copy the index file or files and the index log file (if there is one) to the new dynamic file resident directory.

Default Rules

The following table lists the default rules for memresize. Refer to this table to determine settings for any memresize options that are not explicitly set on the command line.

Parameter	Default
block.size.multiplier	Same as starting file.
DICT	Specifies the dictionary portion of the file. If not specified, memresize the data portion of *filename*.
modulo	Same as the current modulo of the starting file.
PARTTBL	Not applicable if starting file or resulting file is STATIC. If starting file and resulting file are dynamic, use the starting file's per-file part table if there is one. If the starting file does not have a per-file part table, use the system default part table (current setting of PART_TBL configuration parameter or environment variable). **Note**: This option is supported on UniData for UNIX only.
STATIC | DYNAMIC	Same as the starting file.
KEYONLY | KEYDATA	Same as the starting file.
TMPPATH	On UniData for UNIX, /tmp by default. On UniData for Windows Platforms, \TEMP by default. You can specify another path for memresize to use as work space.
TYPE {0 | 1}	Same as the starting file.

memresize Parameters:Default Rules

Examples

The following examples were generated on UniData for UNIX in the order in which they appear by using a copy of the INVENTORY file from the UniData demo database. In the following example, FILE.STAT displays information before resizing:

```
:FILE.STAT INVENTORY
File name (Recoverable Dynamic File) = INVENTORY
Number of groups in file (modulo) = 19
Dynamic hashing, hash type = 0
Split/Merge type = KEYONLY
Block size = 1024
File has 2 groups in level one overflow.
Number of records = 175
Total number of bytes = 13505
...
```

In the next example, memresize converts the file to a static file, with a new modulo.

```
:!memresize INVENTORY 23 STATIC
Resize INVENTORY mod(,sep) = 23(,-1) type = -1 memory = 8000 (k)
static
175 record(s) in file.
INVENTORY RESIZED from 19 to 23
Total time used =2 (sec)
:FILE.STAT INVENTORY
File name (Recoverable Static File) = INVENTORY
Number of groups in file (modulo) = 23
Static hashing, hash type = 0
Block size = 1024
File has 1 groups in level one overflow.
Number of records = 175
Total number of bytes = 13505
...
```

Notice that parameters that were not specified (for instance, block.size.multiplier,MEMORY,and TYPE) were not changed. Some of these parameters appear as -1 in the memresize output, indicating they are not changed.

In the next example, memresize converts the file to a KEYDATA dynamic file with a per-file part table on UniData for UNIX.

:!memresize INVENTORY MEMORY 12000 KEYDATA PARTTBL
/home/terric/parttbl

Resize INVENTORY mod(,sep) = 0(,-1) type = -1 memory = 12000 (k) dynamic

```
KEYDATA PARTTBL=/home/terric/parttbl
RESIZE file INVENTORY to 23.
175 record(s) in file.
INVENTORY RESIZED from 23 to 23
Total time used =1 (sec)
:FILE.STAT INVENTORY
File name (Recoverable Dynamic File) = INVENTORY
Number of groups in file (modulo) = 23
Dynamic hashing, hash type = 0
Split/Merge type = KEYDATA
Block size = 1024
File has 2 groups in level one overflow.
Number of records = 175
Total number of bytes = 13505
...
:!ls -l INVENTORY
total 6
lrwxrwxrwx 1 terric unisrc 41 Jun 16 15:06 dat001 -> /usr/uni-data/
partfiles/ABINVENTORY/dat001
lrwxrwxrwx 1 terric unisrc 42 Jun 16 15:06 over001 -> /usr/uni-data/
partfiles/ABINVENTORY/over001
-rw-rw-rw- 1 terric unisrc 72 Jun 16 15:06 parttbl
```

Notice that after memresize is executed, INVENTORY is a dynamic file even though the DYNAMIC keyword was not specified. Because KEYDATA and PARTTBL are applicable only to dynamic files, using these keywords produces a dynamic file. The dynamic file directory contains links to dat001 and over001 and the per-file part table (parttbl).

Note: The per-file part table is a valid option on UniData for UNIX only.

Related Command

RESIZE

MENUS

Syntax

MENUS

Description

The ECL **MENUS** command invokes the MENUS utility, through which you can modify, display, and print VOC records.

For more information about using the UniData MENUS utility, see *Using UniData.*

Example

When you execute the MENUS command, UniData displays the main menu:

```
:MENUS
MENU Maintenance 15:10:53 Jul 31 1999
1= Enter/Modify a MENU
2= Enter/Modify a formatted MENU
3= Display a summary of all MENUs on a MENU file
4= Display the contents of a MENU
5= Enter/Modify a VOC MENU selector
6= Enter/Modify a VOC stored sentence item
7= Display all MENU selector item on the VOC file
8= Display all stored sentence items on the VOC file
9= Display the dictionary of a file
10= Print a summary of all MENUs on a MENU file
11= Print the contentes of a MENU
12= Print the dictionary of a file
13= Enter/Modify a VOC stored paragraph item
which would you like? (1 - 13)=
```

MESSAGE

Syntax

UniData for UNIX

MESSAGE [!*port*][*user*][*]*string*

UniData for Windows Platforms

MESSAGE [*user*][!*tty*][*]*string*

Description (UniData for UNIX)

The ECL **MESSAGE** command sends text to one or more user terminals.

You must have write permission on the target terminal to send a message to that device.

You can use the UNIX mesg command to set permissions that control access to your terminal. Add this command to your .login or .profile file to set this for each work session. See your operating system documentation for more instructions on the mesg command and setting permissions.

Note: Use the WHO command to determine user login names and port numbers.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
!*port*	The terminal assigned to a user on the same computer. Tip: Execute any of the following to get !*port* (login name): ■ MYSELF ■ !who am i ■ !tty ■ LISTUSER
user	The ID (login name) of the user to receive the message.
*	Directs the message to all user terminals.
string	The message to be sent.

MESSAGE Parameters

Examples

In the following example, sends a message to all users:

```
:MESSAGE * The system will shut down in three minutes.
```

The preceding message displays as follows on all user terminals:

```
From carolw /dev/pts/6 : The system will shut down in three
minutes.
```

Description (UniData for Windows Platforms)

The ECL MESSAGE command directs UniData to send text to a designated user, to a designated session, or to all users.

Note: On UniData for Windows Platforms, UDT.OPTIONS 90 (U_MESSAGE_RAW) enables users to suppress the display of sender information in MESSAGE output.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
user	The login name of the user who is to receive the message.
!*tty*	Sends a message to the terminal whose “tty” you specify. **Note**: Displays the tty with the ECL LISTUSER command.
*	Sends a message to all users.
string	The message to be sent to users.

Message Parameters

Examples

The following example shows MESSAGE output with and without UDT.OPTIONS 90 turned on. For the example, sender and receiver are the same process:

```
:UDT.OPTIONS 90 OFF
:MESSAGE USER01 “Accounts Payable update is complete.”
From USER01 127.0.0.1: “Accounts Payable update is complete.”
:UDT.OPTIONS 90 ON
:MESSAGE USER01 “Accounts Payable update is complete.”
“Accounts Payable update is complete.”
:
```

Notice that only the message string itself displays if UDT.OPTIONS 90 is on.

The next two examples illustrate the !tty option. The following example records a session where two messages were sent, one with and one without UDT.OPTIONS 90:

```
:LISTUSER
Max Number of Users UDT SQL TOTAL
~~~~~~~~~~~~~~~~~~~ ~~~ ~~~ ~~~~~
16 3 0 3
UDTNO USRNBR UID USRNAME USRTYPE TTY IP-ADDRESS TIME DATE
1 68 1404 claire udt pts/1 Console 15:35:41 Jul 21 1999
2 140 1001 USER01 udt pts/2 192.245.122.28 17:21:19 Jul 21 1999
3 132 500 Administ udt pts/3 192.245.122.28 17:22:05 Jul 21 1999
:myself
Administrator pts/3 17:22:05 Jul 21 1998 (192.245.122.28)
:MESSAGE !pts/2 “The General Ledger update is complete.”
:UDT.OPTIONS 90 ON
:MESSAGE !pts/2 “The meeting was canceled.”
:
```

The following message records a session at the terminal where the two messages were received:

```
:MYSELF
USER01 pts/2 17:21:19 Jul 21 1999 (192.245.122.28)
:From Administrator 192.245.122.28: “The General Ledger update is
complete.”
“The meeting was canceled.”
:
```

MIN.MEMORY

Syntax

MIN.MEMORY TEMP *n*

Synonym

MIN-MEMORY TEMP

Description

The **ECL MIN.MEMORY TEMP** command overrides the UniData configuration parameter MIN_MEMORY_TEMP, which defines the number of local pages reserved in memory for a UniData session. The default configuration parameter setting is 64.

Example

The following example sets MIN_MEMORY_TEMP to 128:

```
:MIN.MEMORY TEMP 128
```

mvpart

Syntax

mvpart *filename/part_name destination*

Description

The system-level mvpart command moves one or more part files of a dynamic file. mvpart sets or resets symbolic links if needed and creates or updates a prefix table (.fil_prefix_tbl) at the destination location if needed. Using mvpart ensures that the links, file locations, and prefix tables remain synchronized.

Note: mvpart is supported on UniData for UNIX only.

mvpart is an offline tool. If you execute mvpart while the UniData daemons are running, an error message displays and the command fails.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	UNIX path and file name of the dynamic file directory. Cannot be a synonym or SETFILE pointer.
part_name	Name of the part file you wish to move (for instance, dat00x, over00x, idx00x).
destination	Location to place the part file being moved. Must be either "." or a fully qualified UNIX path. Must be an entry in the part table for *filename*. Use "." to move a part file back to its original dynamic file directory.

mvpart Parameters

Examples

The following examples were generated from a copy of the INVENTORY file from the UniData demo account. The first example shows how the file was created and populated:

```
:CREATE.FILE PRODUCTS 19 DYNAMIC PARTTBL /home/terric/parttbl
Create file D_PRODUCTS, modulo/1,blocksize/1024
Hash type = 0
Create dynamic file PRODUCTS, modulo/19,blocksize/1024
Hash type = 0
Split/Merge type = KEYONLY
Added "@ID", the default record for UniData to DICT PRODUCTS.
:COPY FROM DICT INVENTORY TO DICT PRODUCTS ALL
@ID exists in PRODUCTS, cannot overwrite
15 records copied
:COPY FROM INVENTORY TO PRODUCTS ALL
175 records copied
:!ls -l PRODUCTS
total 6
lrwxrwxrwx 1 terric unisrc 32 Jun 3 09:35 dat001 -> /tmp/part-files/
ACPRODUCTS/dat001
lrwxrwxrwx 1 terric unisrc 33 Jun 3 09:35 over001 -> /tmp/part-files/
ACPRODUCTS/over001
-rw-rw-rw- 1 terric unisrc 35 Jun 3 09:35 parttbl
```

Notice that the per-file part table (parttbl) is in the dynamic file directory.The dat001 and over001 are physically located on a different file system. The location of dat001 and over001 is determined by the part table, shown in the next example:

```
:!more ./PRODUCTS/parttbl
. 10000000
/tmp/partfiles 10000
```

The following example shows how to move the dat001 back to the dynamic file directory. Notice that it is not necessary to set your current working directory to the UniData account:

```
# pwd
/usr/ud60
# $UDTBIN/mvpart $UDTHOME/demo/PRODUCTS/dat001 .
# ls -l $UDTHOME/demo/PRODUCTS
total 44
-rw-rw-rw- 1 root sys 20480 Jun 3 09:46 dat001
lrwxrwxrwx 1 claireg unisrc 33 Jun 3 09:35 over001 -> /tmp/part-
files/
ACPRODUCTS/over001
-rw-rw-rw- 1 claireg unisrc 35 Jun 3 09:35 parttbl
```

Notice these points about the preceding example:

- You must be logged in as root to execute mvpart.
- You can execute mvpart from any directory as long as you specify the full path and file name for the dynamic file directory. If it is located in your current directory, you can specify its relative path.
- When you specify . in the command line, the part file is moved to its original dynamic file directory, not to your current directory.

The following example shows what happens if a user executes the mvpart command while the UniData daemons are running:

```
:!mvpart
mvpart has detected that the UniData daemons are running.
The system administrator must stop the daemons (with stopud)
before mvpart can execute.
```

Warning: If you want to relocate part files, shut down UniData and use mvpart. Do not use the UNIX cp or mv command, or your file may be damaged and UniData may crash. Also, the cp and mv commands do not update symbolic links or the .fil_prefix_tbl.

MYSELF

Syntax

MYSELF

Description

The ECL **MYSELF** command displays the following session information for the user logged in to the terminal where the command is executed:

- The login name.
- On UniData for UNIX, the terminal identification number (tty).
- On UniData for Windows Platforms, the tty number is a session identifier constructed by appending the udtno (displayed in the output from LISTUSER) to the string pts/.
- The date and time the user logged on to UniData.
- On UniData for Windows Platforms, the terminal identification (Console or IP address).

Example

The following example shows a MYSELF command display on UniData for UNIX:

```
:MYSELF
carolw pts/6 Jul 31 10:41
:
```

newacct

Syntax

newacct [*account.owner*][*group*]

Description

The system-level **newacct** command creates a UniData account in the current directory.

If you do not specify an account owner or group, newacct lists the available owners and groups and prompts for them. A maximum of 4096 login names are displayed. You can limit the login names or groups by specifying account owner and group in the command line.

For more information about creating new UniData accounts, see *Administering UniData.*

Note: Unless you log in as root on UniData for UNIX or Administrator on UniData for Windows Platforms, UniData uses your current login and group ID and ignores your responses to the prompt.

Example

The following example creates a new UniData account:

```
# $UDTBIN/newacct
The UDTHOME for this account is /disk1/ud52/.
Do you want to continue(y/n)? y
Current directory is '/home/carolw'
......................... List of Users .........................
abuls croweb jeffa linq pamm spooler ukm01
adm daemon jeffreyk lisac pasche srcman uks01
...

carolw
...
# Please enter account owners login name: carolw
......................... List of Groups .........................
acctg consulting lp nuucp root tty
adm daemon lp other sbusers ukusers
adm daemon mail other sys unisrc
bin dw mail remusers sys users
bin guests nogroup root techserv users
Please enter the account group name: users
Initializing the account ...
#
```

newhome

Syntax

newhome *path*

Description

The system-level **newhome** command creates an alternate global catalog space for globally cataloged UniBasic programs.

newhome creates or overlays the directory indicated by path, and then copies all files from *udtbin*/sys to path/sys on UniData for UNIX or to *udtbin*\sys to path\sys on UniData for Windows Platforms. After setting up the new home account, you must reset the environment variable udthome to point to the new home account. Also, you must recatalog UniBasic programs or copy their object code to the new catalog space to make them available to the new account.

newhome does not create the entire directory structure that exists in the default udthome, and it does not copy UniBasic executables developed at your site.

Note: To execute the newhome command, you must be root on UniData for UNIX or Administrator on UniData for Windows Platforms.

See *Administering UniData* for more information on creating an alternate global space and for managing cataloged UniBasic programs.

Files and Directories Created by newhome

UniData creates or overlays the directory indicated by path. This directory will contain only the subdirectory sys, which contains the following files and directories:

```
# ls
@README CTLGTB D_HELP.FILE LANGGRP

@README-IMPORTANT DENAT_BP D_JAPANESE.MSG MULTIBYTE.CONFI
@VERSIONS DICT.DICT D_MSG.DEFAULTS SAVEDLISTS
AE_BP D_AE_BP D_SAVEDLISTS SYS_BP
AE_COMS D_AE_COMS D_SYS_BP UDTSPOOL.CONFIG
AE_COMS_DEMO D_AE_DOC D_UDT_GUIDE VOC
AE_DOC D_BP D_VOC X_HELP.FILE
AE_SECURITY D_COLLATIONS D__MAP_ _MAP_
AE_SYSTOOLS D_CTLG D__PH_ _PH_
AE_UPCHARS D_CTLGTB ENGLISH.MSG makefile
AE_XCOMS D_DENAT_BP ENGLISH_G2.MSG set_sys.sh
BP D_ENGLISH.MSG FRENCH.MSG uniapi.msg
COLLATIONS D_ENGLISH_G2.MSG HELP.FILE vocupgrade
CTLG D_FRENCH.MSG JAPANESE.MSG
```

The following files and directories make up the program catalog spaces:

- D_CTLGTB
- CTLGTB
- D_CTLG
- CTLG, including subdirectories a through z and X for storing globally cataloged programs.

Creating an Alternate Catalog Space on UniData for Windows Platforms

Complete the following steps to create an alternate global catalog space on UniData for Windows Platforms:

1. **Log on to Your System**

   Log in to your system as an Administrator.

2. **Create the Folder**

   Use the MS-DOS mkdir command to create the folder (or create it through NT Explorer or My Computer). Then use the Security tab on the folders Properties sheet to give Administrators Full Control permissions to Administrators.

3. **Set UDTHOME Environment Variable**

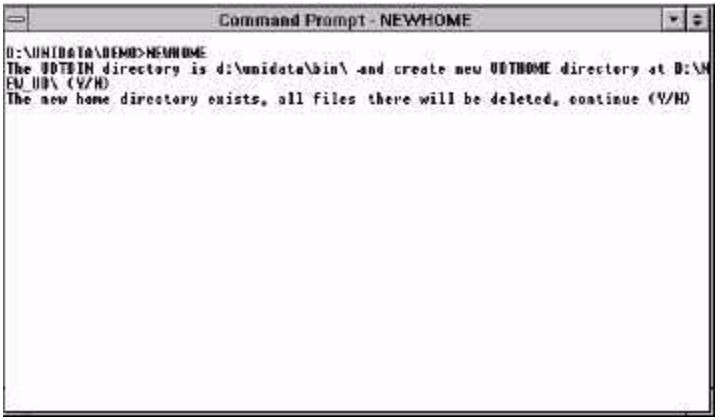
   To execute the newhome command, you must set the environment variable UDTHOME to point to the directory you just created. The following example shows how to create the directory and set UDTHOME from the MS-DOS command prompts.

Note: Do not change the value of UDTHOME for any other users until you have completed all the steps for the new alternate global catalog space.

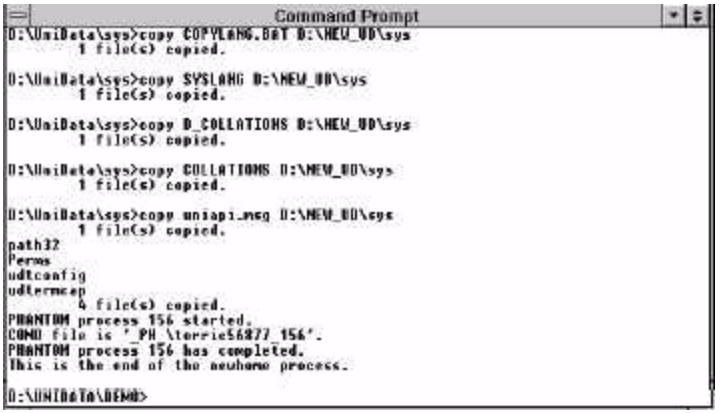
4. **Execute newhome**

The system-level newhome command copies relevant files from the default udthome into the directory you specified with the UDTHOME environment variable.

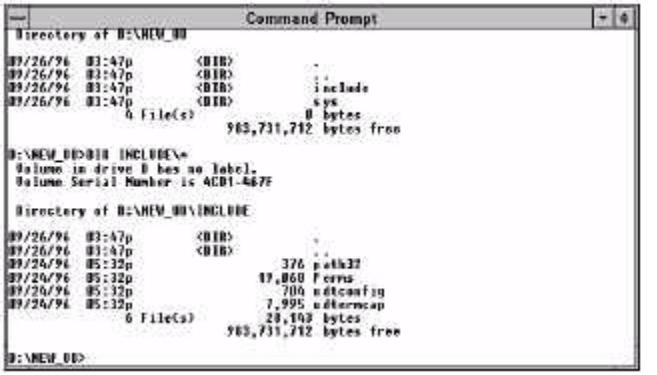
The following screen illustrates typical output from the newhome command:



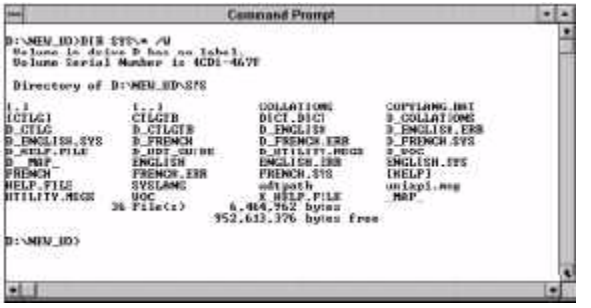
Answering Y at the prompt causes the command to complete, as shown in the following example:



The next screens show the results of the newhome command. The first screen shows the udthome directory. Notice that the command has created and populated the sys and include directories.

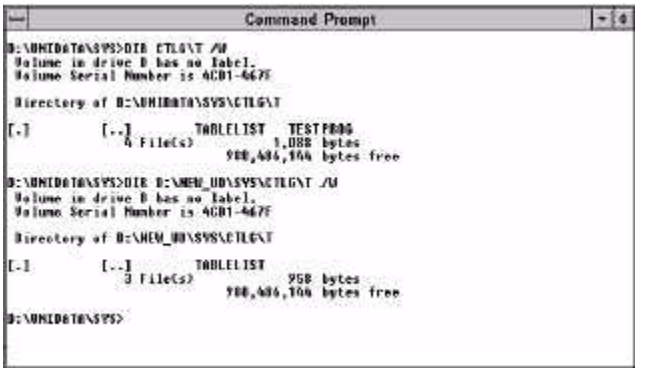


The next screen shows the contents of the new sys directory:



Notice that the newhome command created and populated two subdirectories: sys and include. newhome does **not** create the entire directory structure that exists in the default *udthome*.

The newhome command also copies all globally cataloged programs released with UniData into the alternate global catalog. newhome does not copy UniBasic programs that you developed at your site. The following example compares some contents of a newly created alternate global catalog and the default global catalog for your system:



Notice that the alternate global catalog does not include the TESTPROG program.

5. Activate the Alternate Global Catalog

Complete the following steps to begin using the alternate global NScatalog space. Remember that the value for the UDTHOME NSenvironment variable determines which global catalog space a user accesses when cataloging a program or executing a globally cataloged program. The VOC pointer for CTLGTB determines which global catalog table the user accesses.

- Modify VOC Pointer – Decide which UniData accounts should access the new global catalog space. For each such account, modify the VOC entry for the global catalog table to point to the new location. Users can still compile and catalog if this VOC pointer and the UDTHOME environment variable are not consistent, but they may encounter puzzling results, since CTLGTB and CTLG will not necessarily match.

You can make the VOC entry a soft pointer, so that the current setting for the UDTHOME environment variable determines the location of both the global catalog and the global catalog table. The following screen shows an example of a soft VOC pointer:

- Modify UDTHOME Environment Variable for Users – You need to reset the UDTHOME environment variable for each user who should access the alternate global catalog space. The value of UDTHOME this is defined during a particular UniData session determines which global catalog space a user accesses. Users with access to the Control Panel or the MS-DOS prompt can reset UDTHOME.
- Move Application Programs Into the New Space – Enter UniData in an account where your application programs reside,

and globally catalog all the programs that should be accessed from the new space. Since you have reset UDTHOME, cataloging the programs globally locates them in the new catalog space.

Creating an Alternate Catalog Space on UniData for UNIX

Follow the steps below to create an alternate global catalog space:

1. **Make New Directory**

   At the system prompt, create the directory for the new global catalog space, then change to the new directory, as shown in the following example:

   % **mkdir claireg**
   % **cd claireg**
   % **pwd**
   /disk1/claireg

2. **Execute newhome**

   Execute the newhome command, indicating the path to the location for the new account. In this case, a new UNIX directory, testenv, will be created under /disk1/claireg:

   % **newhome testenv**
   Creating new UniData home /disk1/claireg/testenv ...
   UniData has created the new home account. This account contains only the sys directory with UniData's cataloged programs. To access your new home account, you must reset the UDTHOME environment variable.

3. **Set UDTHOME Environment Variable**

   To access the new home account, reset the UDTHOME environment variable.

   From the Bourne or Korn shell:

   UDTHOME=/disk1/claireg/testenv;export UDTHOME

   From the C shell:

   % **setenv UDTHOME /disk1/claireg/testenv**

4. **Make UniBasic Programs Available**

   Make available to the new account any globally cataloged UniBasic programs. You can do this by setting a VOC pointer to the old catalog space, or by copying the cataloged programs into the new account:

   - VOC pointer – You can associate CTLGTB with *udthome* by setting up a VOC pointer in each account. The pointer looks like this:

     F
     @UDTHOME/sys/CTLGTB
     @UDTHOME/sys/D_CTLGTB/

   - **Copy object code records** – To copy all globally cataloged programs, enter the following series of UNIX commands, replacing *original_udthome* and *new_udthome* with the paths to your program files:

     %**cd original_udthome/sys/CTLG**
     find * -type f -print | cpio -pm new_udthome/sys/CTLG

NEWPCODE

Syntax

NEWPCODE *path*

Description

The ECL **NEWPCODE** (new pseudo-code) command activates the latest version of a program. path is the full path to the new object code for the program. The NEWPCODE command is effective only in the udt session from which it is executed.

If a UniBasic program CALLs or EXECUTEs another program or subroutine, UniData executes the version that was cataloged when the calling program began executing unless you do one of the following:

- Stop and restart the executing program.
- Execute NEWPCODE to activate another version

You do not need to execute NEWPCODE if you globally catalog a program because global cataloging notifies the shared memory server that a new version is available. However, if you catalog the program locally or directly, you do need to execute NEWPCODE to remove the object code from local memory.

Tip: Use NEWPCODE in a UniBasic program to modify, recompile, recatalog, and retest it without exiting to ECL. An example is provided in the following section.

For more information about writing programs in UniBasic, see *Developing UniBasic Applications*.

Example

In the following UniBasic program, notice that, until NEWPCODE is executed, UniData executes the version of the program in shared memory. The line that contains the NEWPCODE command is shown in bold.

```
EXECUTE "DELETE.CATALOG test"; * START CLEAN
OPEN 'BP' TO BP ELSE STOP
* create simple BASIC program to print HELLO
REC = 'PRINT "HELLO"'
WRITE REC ON BP, "test"
*compile, catalog, and run the program
EXECUTE "BASIC BP test"
EXECUTE "CATALOG BP test"
EXECUTE "test"
*Change TEST program to print HELLO THERE, recompile and run
again.
BPREC = 'PRINT "HELLO THERE"'
WRITE BPREC ON BP, "test"
EXECUTE "BASIC BP test"
PCPERFORM "cp BP/_testc /disk1/ud60/sys/CTLG/t/testc"
* instead of using
*EXECUTE "CATALOG BP test FORCE"
EXECUTE "testc"
* HELLO is still printed on the screen.
* Note: /usr/ud is the path to the UniData home directory.
EXECUTE "NEWPCODE /disk1/ud60/sys/CTLG/t/testc"
EXECUTE "testc"
* HELLO THERE is printed on the screen
END
```

The preceding program displays the following output:

```
:BASIC BP TEST_NEWP
Compiling Unibasic: BP/testc in mode 'u'.
compilation finished
HELLO
Compiling Unibasic: BP/testc in mode 'u'.
compilation finished

HELLO
HELLO THERE
```

Related Command

newversion

newversion

Syntax

newversion *path_program user*[,*userM*...,*userN*]

Description

The system-level **newversion** command replaces the UniBasic executable in shared memory with a newly cataloged version. Programs and subroutines are replaced only when the calling and called programs are in use.

newversion differs from NEWPCODE in that newversion requires that you specify a user or users to obtain the new version, and all other users obtain the previous version. NEWPCODE, on the other hand, changes the version of a program in shared memory for all users.

Use this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

You can define the users who have permission to execute the newversion command by modifying the udtconfig file. To define the users, create an entry in udtconfig for NEWVERSION_USERS, followed by the user numbers allowed to execute newversion. Separate each user number with a comma. If you want all users to be able to execute newversion, set the user number to ALL, as shown in the following example:

```
# cd /usr/ud60/include
# vi udtconfig
“udtconfig” 140 lines, 2486 characters
# Unidata Configuration Parameters
#
# Section 1 Neutral parameters
# These parameters are required by all Unidata installations.
# 1.1 System dependent parameters, they should not be changed.
LOCKFIFO=1
SYS_PV=3

# 1.2 Changable parameters
NFILES=60
NUSERS=20
WRITE_TO_CONSOLE=0
TMP=/tmp/
NVLMARK=
FCNTL_ON=0
TOGGLE_NAP_TIME=161
NULL_FLAG=0
N_FILESYS=200
N_GLM_GLOBAL_BUCKET=101
N_GLM_SELF_BUCKET=23
GLM_MEM_ALLOC=10
GLM_MEM_SEGSZ=4194304
NEWVERSION_USERS=ALL
.
.
.
```

If you do not modify the udtconfig file, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms to execute the newversion command.

For more information about cataloging UniBasic programs, see the CATALOG command or *Administering UniData.*

Tip: Use the LISTUSER command to obtain a list of process IDs (USRNBR).

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
path_program	The full path to the new version of a compiled program.
user	Process ID the administrator assigns to a UniData session. If you specify more than one *user*, separate user numbers with spaces.

newversion Parameters

Related Commands

CATALOG, NEWPCODE

NFAUSER

Syntax

NFAUSER(“*username*”, “*password*”)

Description

Beginning at UniData 5.0, a Network File Access (NFA) connection from an NFA client requires a valid user name and password. If the client connection is made via udtelnet, this information is available and passed to the NFA server for connecting. If the session is a console session, the system prompts for the user name and password when a connection is requested, such as when you OPEN the first NFA file on a database.

UniBasic now provides the NFAUSER function which enables you to set the user name and password in a UniBasic program.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
username	A valid user name on the NFA server to which you are connecting.
password	The password corresponding to username.

NFAUSER Parameters

After running this function and providing a valid user name and password combination, an NFA connection no longer prompts for this information, regardless if it is from the console or via udtelnet. NFAUSER does not validate the user name/password combination, but registers them in an internal variable for later use.

NODIRCONVERT

Syntax

NODIRCONVERT [ON | OFF]

Description

The **NODIRCONVERT** command provides the ability to read and write items in a DIR-type file without translating any characters.

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
ON	Newlines are not converted to field marks when read from a DIR-type file.
OFF	READ statements convert newlines to field marks. The WRITE statement converts them back to newlines. This is the default setting.

NODIRECONVERT Parameters

ON.ABORT

Syntax

ON.ABORT *command*

Synonym

ON-ABORT

Description

The ECL ON.ABORT command identifies a command that UniData executes when a UniBasic program aborts. command may be an ECL command, a paragraph, or a directly or globally cataloged UniBasic program. This setting remains in effect until you clear it with the CLEAR.ONABORT command.

Note: UDT.OPTIONS 105, U_EXECUTE_ONABORT, determines whether to allow ON.ABORT to take effect from a PERFORM or EXECUTE statement in UniBasic. For more information about this option, see the UDT.OPTIONS Commands Reference.

Examples

The following is a VOC entry for a paragraph called APOLOGY. This paragraph displays “This program has terminated. We are sorry for the inconvenience.”

```
VOC RECORD ID==>APOLOGY
0 @ID=APOLOGY
1 F1=PA
2 F2=DISPLAYThis program has terminated. We are sorry for the
inconvenience.
```

Here is a UniBasic program that always aborts because it contains the ABORT command:

```
DISPLAY “This example shows what happens when a program aborts if
you set ON.ABORT in UniData.”
DISPLAY “For more information about the ON.ABORT command, refer to
the following material:”
ABORT
DISPLAY “UniData Commands Reference”
```

This example sets ON.ABORT to the paragraph APOLOGY, then runs TEST_PROG, which aborts when it reaches the ABORT command. Then APOLOGY executes, displaying its message.

Finally, the cursor returns to the UniData colon prompt.

```
:ON.ABORT APOLOGY
:RUN BP TEST_PROG
This example shows what happens when a program aborts if you set
ON.ABORT in UniData.
For more information about the ON.ABORT command, refer to the
following material:
This program has terminated. We are sorry for the inconvenience.
:
```

Related Command

CLEAR.ONABORT

ON.BREAK

Syntax

ON.BREAK command

Synonym

ON-BREAK

Description

The **ON.BREAK** command executes command, a VOC paragraph, or a sentence when the user presses the interrupt key during execution of UniQuery statement in the current UniData session. By default, the cursor returns to the environment from which the ON.BREAK command was executed.

Tip: Use ON.BREAK to allow users to break out of report display, but then offer a menu rather than allowing them access to the ECL prompt.

For more information on creating VOC sentences and paragraphs, see *Using UniData.*

The interrupt key must first be enabled by setting PTERM -BREAK ON. See your operating system documentation for instructions on setting the interrupt key.

After the user presses the break key, UniData displays the default break message:

```
BREAK: Enter Q<return> to Quit. Any other character to continue
```

ON.BREAK does not change or remove this prompt. *command* executes after the user enters Q and presses ENTER.

Examples

The following example displays the VOC sentence BREAK.KEY:

```
001: S
002: DISPLAY You have pressed the BREAK key.
```

The following example demonstrates the effect of setting ON.BREAK to execute the preceding sentence. First ON.BREAK is set to execute BREAK.KEY. Then the user executes LIST CLIENTS LNAME.

```
:ON.BREAK BREAK.KEY
:LIST CLIENTS LNAME
LIST CLIENTS LNAME 11:27:54 Jun 06 1999 1
CLIENTS... Last Name......
9999 Castiglione
10034 Anderson
9980 Ostrovich
10015 di Grigorio
...
Enter <New line> to continue...
```

At this point, the user presses the BREAK key. The default prompt displays, to which the user responds by entering Q and pressing ENTER. Notice that the header for the report displays again.

```
BREAK: Enter Q<return> to Quit. Any other character to continueQ
LIST CLIENTS LNAME 11:30:45 Jun 06 1999 2
CLIENTS... Last Name......
```

Finally, the sentence BREAK.KEY executes, and the cursor returns to the ECL prompt:

```
You have pressed the BREAK key.
:
```

Related Commands

CLEAR.ONBREAK, PTERM

PAGE

Syntax

PAGE *filename record*

Description

The ECL **PAGE** command displays the contents of a record to the screen.The display pauses at the bottom of each page and continues after the user presses ENTER.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	A UniData file name.
record	A record in *filename*. You can list only one record ID on the command line.

PAGE Parameters

Example

The following example displays a record from the INVENTORY demo file. Notice that a UniData delimiter is displayed as ‘ y.’ Your system may display a different character.

```
:PAGE CLIENTS 9999
Paul
Castiglione
Chez Paul
45, reu de Rivoli
Paris
75008
France
3342425544y3342664857
WorkyFax
(EOF)Enter h for help, <CR> for next page
```

PATHSUB

Syntax

PATHSUB

Description

The ECL **PATHSUB** command changes paths and subpaths globally in all catalog entries and file pointers in the VOC. You do not have to provide the full path, just the part that you want to change. PATHSUB first selects all local catalog entries and for each item, replaces the old path with the new path. Then PATHSUB selects all DIR and F-type pointers and substitutes the new path for the old.

Tip: Use PATHSUB to change the VOC pointers after moving an account.

Check your entry carefully, because PATHSUB replaces the Original sub-path with the path you enter with no verification that the path is valid.

Example

The following example shows output from PATHSUB on UniData for UNIX. In this example, the user is changing an account directory subpath name from /disk1 to /usr (the full paths are /disk1/ud60 and /usr/ud60) on UniData for UNIX. Notice that UniData prompts for the old and new paths, then echoes them back for confirmation before continuing. The subsequent messages follow processing as UniData looks for the old path in VOC records that point to locally cataloged programs (finding none), then in VOC records that contain file pointers (finding 9).

```
:PATHSUB
This program allows you to globally substitute paths or sub-paths
in the voc. For example, if you move your accounts from /usr/ud to
/usr2 you could update all voc entries to reflect this path with
this program.

Original sub-path : /disk1
New sub-path : /usr
Old path: /disk1
New path: /usr
Is this acceptable? (y/n) : y
Updating local catalog entries in voc...

4 records selected to list 0.

Updated 0 local catalog entries in voc.

Updating file pointers in voc...

48 records selected to list 0.

Updated 9 file pointers in voc.

Voc has been updated.
```

PAUSE

Syntax

PAUSE [*wait_time*]

Description

The ECL **PAUSE** command suspends the UniData process that issues the command for the amount of time specified by *wait_time*. Notice the following points when executing PAUSE:

- PAUSE has no effect if *wait_time* is a negative number, or if another UniData process has previously issued a command for this process.
- To pause a process indefinitely, omit *wait_time*, or specify a *wait_time* of 2.
- PAUSE must be executed by the process to be paused.

Examples

The following series of screen displays demonstrate execution of the ECL PAUSE command. First, a UniData session is paused. Following this, a separate screen display shows the paused session listed as output of the LIST.PAUSED command, which was executed from a different UniData session. The final screen display demonstrates waking the paused session with the WAKE command.

```
:PAUSE
:LIST.PAUSED
Number of Paused Users
~~~~~~~~~~~~~~~~~~~~~

1
UDTNO USRNBR UID USRNAME USRTYPE TTY LEFTTIME TOT_TIME
1 1052 1283 carolw udt pts/0 - -:
Screen Example
:WAKE 1052
:
```

Related Commands

UniData

LIST.PAUSED, WAKE

UniBasic Commands

PAUSE, WAKE — For information, see the *UniBasic Commands Reference*.

PHANTOM

Syntax

PHANTOM *process*

Description

The ECL **PHANTOM** command executes process in the background. process can be an ECL command, a paragraph, or a globally cataloged program.

UniData stores the output from the background process in the _PH_ file under a record name made up of the users login name concatenated with the internal system time and the process ID.

Since the task is running in the background, any processes that require input should have an associated DATA statement, or have data in the DATA queue. If a request for input that would normally be directed to the display terminal is made to a background process, the process aborts.

If a login paragraph exists in the VOC file of the account from which you issue the PHANTOM command, UniData executes the login paragraph before executing the background process. You may want to test @USER.TYPE in your login paragraph and not execute any processing that should be executed only in an interactive UniData session.

Warning: Background processes you create are independent of your process. They will survive as phantom processes even if you terminate your process (by logging out of the system for instance). Since UniData stores the output from phantom processes in _PH_, this can create a large _PH_ file.

@USER.TYPE returns the type of process currently running. There are three types of processes:

- Normal terminal processes (@USER.TYPE = 0).
- Background (PHANTOM) processes (@USER.TYPE = 1).
- Redirected standard input (@USER.TYPE = 2).

Starting PHANTOM Processes from the Operating System

You can invoke UniData from the operating system, including a PHANTOM command on the same command line using the following syntax:

```
udt PHANTOM process
```

On UniData for UNIX, the shell functions pipe (|) and I/O redirection (>) also work with udt:

```
% echo “LIST CLIENTS ALL” | udt > out &
```

Tip: Such udt processes do not work within all C shell environments, but function properly under the UNIX Bourne shell.

PHANTOM Command Exit Codes

When phantom processes are running, you may see an error message like the following, where code is an exit code number:

```
Phantom run basic error exit code
```

The following table lists the exit codes generated by phantom processes.

Code	Description
1	Runtime error.
3	User abort statement.
4	Phantom process requested input data.
5	Phantom process was interrupted.
6	Message queue error.

PHANTOM Exit Codes

Examples

The following example executes the paragraph CUST.PROCESS as a phantom. Note that this is a simple representation. It is not unusual for other things to occur before the completion message appears.

```
:PHANTOM CUST.PROCESS
:PHANTOM process 5370 started.
COMO file is _PH_/ud60151599_5370/
PHANTOM process 5370 has completed.
```

In the next example, UniData processes a UniQuery statement in the background and stores the output in the _PH_ file:

```
:PHANTOM LIST CLIENTS
:PHANTOM process 13495 started.
COMO file is '_PH_/peggys61432_13495'.
```

The LIST command confirms the existence of the output file peggys61432_13495:

```
:LIST _PH_
LIST _PH_ 17:04:48 Jun 06 1999 1
_PH_......
O_TEST_SES
SION
peggys6143
2_13495
2 records listed
```

The SPOOL command in the next example displays the output of the above process to the terminal:

```
:SPOOL _PH_ peggys61432_13495 -T
_PH_:
peggys61432_13495
LIST CLIENTS NAME COMPANY ADDRESS CITY STATE ZIP COUNTRY PHONE
PHONE_TYPE
17:03:53 Jun 06 1999 1
CLIENTS 9999
Name Paul Castiglione
Company Name Chez Paul
Address 45, reu de Rivoli
City Paris
State/Territory
Postal Code 75008
Country France
Phone Number (33) (4) 24-25-54-4
(33) (4) 26-64-85-7
Phone Category Work
Fax
CLIENTS 10034
Name Fredrick Anderson
Company Name Otis Concrete
Address 854, reu de Rivoli

City Paris
Enter <New line> to continue...
```

PORT.STATUS

Syntax

PORT.STATUS [USER *username* | PID*pid* | PORT*device* | LPTR | FILEMAP | CALL.STACK]

Description

A new ECL command, **PORT.STATUS**, has been added at this release. PORT.STATUS displays information about resource usage for a udt process that is running.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
USER *username*	Lists information only for the *username* you specify.
PID *pid*	Lists information only for the *pid* you specify.
PORT *device*	Lists information only for the device you specify.
LPTR	Sends output to the printer.
FILEMAP	Lists the files open in UniBasic for the *pid* you specify. You must use this option with this PID *pid* option.
CALL.STACK	Lists the current ECL stack for the pid you specify. If the process is running a UniBasic program, UniData also displays the UniBasic call stack.

PORT.STATUS Parameters

Examples

The following example illustrates the output from the PORT.STATUS command when you use the USER option:

```
:PORT.STATUS USER claireg

Licensed/Effective # of Users Udt Sql Total
   32 /32              1        0        1
Udtno Pid      User Port   Last command processed
1     26345claireg pts/t1 PORT.STATUS USER claireg

:PORT.STATUS USER claireg
Licensed/Effective # of Users Udt Sql Total
   32 /32                        2  0     2
Udtno Pid   User     Port   Last command processed
1     26345 claireg pts/t1 PORT.STATUS USER claireg
2     26433 claireg pts/0  AE
```

The next example shows the output from PORT.STATUS when you use the PID option:

```
:PORT.STATUS PID 26433
Licensed/Effective # of Users Udt Sql Total
   32 /32                        2  0     2
Udtno Pid    User    Port  Last command processed
2     26433 claireg pts/0 AE
```

The next example illustrates the FILEMAP option of the PORT.STATUS command:

```
:PORT.STATUS PID 26433 FILEMAP
Licensed/Effective # of Users Udt Sql Total
   32 /32                        2  0     2
Udtno Pid    User    Port  Last command processed
2     26433 claireg pts/0 AE

S File names
O /home/claireg/VOC
O /home/claireg/AE_COMS
O /liz1/ud52/sys/AE_DOC
O /home/claireg/VOC
```

The final example shows the output from the PORT.STATUS command with the CALL.STACK option:

```
:PORT.STATUS PID 26433 CALL.STACK
Licensed/Effective # of Users Udt Sql Total
   32 /32                        2  0     2
Udtno Pid   User     Port  Last command processed
2     26433 claireg pts/0 SELECT CUSTOMER WITH STATE = “CO”

Session is not in BASIC.
ECL session stack
AE
LIST CUSTOMER
SELECT CUSTOMER WITH STATE = “CO”
SELECT CUSTOMER WITH STATE = “CO”
```

PRIMENUMBER

Syntax

PRIMENUMBER *number*

Description

The ECL **PRIMENUMBER** command displays the first prime number that is equal to or greater than *number*.

Example

In the following example, UniData returns the prime number 449, which is the first prime number greater than or equal to 444.

```
:PRIMENUMBER 444
PRIME number is 449
```

PRINT.ORDER

Syntax

PRINT.ORDER [0 | 1]

Synonym

PRINT-ORDER

Description

The ECL **PRINT.ORDER** command determines the order in which UniData completes print jobs and sends them to the printer. This setting is meaningful only when more than one print job at a time is active in a UniBasic program. Printer units do not close in any specific order by default.

For more information on directing printing in UniData, see *Administering UniData*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	UniData displays the current order.
0	Default. No specific order is used.
1	UniData closes the most recently used printer first.

PRINT.ORDER Parameters

Related Commands

UniData

SP.ASSIGN

UniBasic

PRINT ON – For information, see the *UniBasic Commands Reference*.

PROTOCOL

Syntax

PROTOCOL *line* [“*options*”]

Description

The ECL **PROTOCOL** command sets data line transmission characteristics and protocols for a *line*. The *line* must already be attached.

Tip: Use the SETLINE command to define a tty device. Use the LINE.ATT command to attach a communication line to that device to your process.

Parameters (UniData for UNIX)

The following table describes each parameter of the syntax:

Parameter	Description
line	A tty device defined by the SETLINE command.
options	A group of tty attributes. The options for this command are the same as the UNIX stty and termio commands. *options* must be enclosed in double quotation marks. If you do not indicate any options, UniData displays the current settings.

PROTOCOL Parameters

For more information on the stty command, see your host operating system documentation.

Example (UniData for UNIX)

The following example sets line 0 on a UNIX operating system with

- Baud rate of 9600.
- No echo on input.

- Canonical process turned off (for input).

```
:LINE.ATT 0
:PROTOCOL 0 “9600 -echo -icanon”
```

Parameters (UniData for Windows Platforms)

The following table describes each parameter of the syntax.

Parameter	Description
line	The line number assigned to the device with the SETLINE command.
BAUD = *b*	The baud rate for the communication device. May be a baud rate or a baud rate index.
DATA = *d*	The number of bits in the bytes transmitted and received. Can be from 4 to 8.
STOP = *s*	The number of stop bits; may be 1, 1.5, or 2.
Parity = *p*	The method of marking boundaries of characters. May be none, even, odd, mark, or space.
to = on | off	Controls behavior of transmission if input buffer approaches full. If to = off, transmission stops; if to = on (recommended) transmission does not stop.
xon = on | off	Select / clear Xon / Xoff flow control. If xon = on, Xon / Xoff is selected.
odsr = on | off	DSR handshaking.
octs = on | off	CTS handshaking.
dtr = on | off | hs	DTR circuit.
rts = on | off | hs |tg	RTS circuit.
idsr = on | off	DSR sensitivity.

PROTOCOL Parameters

Include the options, separated by spaces, in a string enclosed with quotation marks, as follows:

```
PROTOCOL 0 “Baud = 9600 xon = on”
```

Example (UniData for Windows Platforms)

In the following example, PROTOCOL displays the current settings for a COM port:

```
:LINE.STATUS
LINE# STATUS PID USER-NAME DEVICE-NAME
0 Available N/A N/A COM1

Line number(s) are attached by the current udt process:
      None

:LINE.ATT 0
LINE 0 ATTACHED
:PROTOCOL 0
Settings for line 0:
Baud Rate = 1200Parity = EvenData Bits = 7Stop Bits = 1.
```

Related Commands

UniData

LINE.ATT,LINE.DET, LINE.STATUS, SETLINE, UNSETLINE

UniBasic

GET, SEND — For information, see the *UniBasic Commands Reference*.

PTERM

Syntax

PTERM [-BREAK {OFF | ON}] [-DISPLAY] [-ERASE "*char*"] [-FULL] [-HALF {LF | NOLF}] [-KILL "*char*"] [-NOXOFF] [-XOFF]

Description

The ECL **PTERM** command establishes terminal settings. These settings remain in effect until the UniData session ends or until the process executes another PTERM command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-BREAK {OFF | ON}	Toggles the interrupt character off and on. ■ OFF – Disables the interrupt key. ■ ON – Default. Enables the interrupt key.
-DISPLAY	Displays the current tty setting for your terminal.
-ERASE "*char*"	Establishes the value (*char*) of the erase character. You cannot set *char* to its current value. *char* is a single ASCII character.
-FULL	Establishes full-duplex mode for your terminal. Under full-duplex, all characters typed in echo to the screen. Full-duplex is the default value.

PTERM Parameters

Parameter	Description
-HALF {LF | NOLF}	Establishes half-duplex mode for your terminal. Under half-duplex, characters you enter at your terminal do not echo to the screen. ■ LF – UniData does not echo a carriage return with a line feed. This is the default. ■ UniData echoes a carriage return with a line feed.
-KILL "*char*"	Establishes *char* as the kill character. *char* is a single ASCII character.
-NOXOFF	Disables support for XON / XOFF. The default value is XON / XOFF enabled.
-XOFF	Establishes XON / XOFF support for your terminal. When you enable the XON / XOFF feature, CTRL-S stops output to the screen. CTRL-Q resumes output.

PTERM Parameters (continued)

Examples

In the following example, UniData changes some terminal settings:

- Disables the interrupt key.
- Changes the erase character to ^B (UniData does not display the control character on the command line when you enter it).
- Disables the XON / XOFF feature.

```
:PTERM -BREAK OFF -ERASE "" -NOXOFF
:
```

In the next example, UniData displays the current values of PTERM:

```
:PTERM -DISPLAY
Erase =^B = 02 octal
Kill = ^U = 025 octal
FULL duplex.
XOFF disabled(have to physically turn off XON/XOFF on smart
terminals).
BREAK OFF
:
```

PTRDISABLE

Syntax

PTRDISABLE *printer* [-c | -w]

Synonym

STOPPTR

Description

The ECL **PTRDISABLE** command prevents UniData from printing jobs that are associated with a queue named *printer*.

On UniData for Windows Platforms, only users with Full Control permissions on a printer can control the printer with PTRDISABLE and PTRENABLE. Check **Permissions** on the **Security** tab of the printers **Properties** sheet to determine who has permissions.

Tip: To resume printing, use the PTRENABLE command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
printer	Name assigned to a print queue with the SETPTR command and the DEST option.
-c	Cancels the current print job before disabling the print queue. **Note**: This option works with UNIX System V releases only.
-w	Allows the current print job to complete before disabling the print queue. **Note**: This option works only with UNIX System V releases.

PTRDISABLE Parameters

Examples

In the following example, taken from UniData for UNIX, UniData disables a print queue called hpzone3:

```
:PTRDISABLE hpzone3
printer "hpzone3" now disabled
:
```

The next example, taken from UniData for Windows NT, disables a local printer called LETTER:

PTRDISABLE LETTER

```
:LISTPTR
Unit.. Printer.................
Port.....................Status..
0 LETTER \\DENVER4\hpzone3 Paused
1 \\DENVER4\hpzone3 hpzone3 Running
2 LEGAL \\DENVER4\hpzone3 Running
3 \\DENVER4\hpzone2 hpzone2 Running
```

Tip: You can use this command in conjunction with the SETPTR options FORM and DEST to turn off print queues associated with different forms and to load new forms into the printer and clear paper jams.

Related Commands

PTRENABLE, SETPTR

PTRENABLE

Syntax

PTRENABLE *printer*

Synonym

STARTPTR

Description

The ECL PTRENABLE command resumes printing jobs that are associated with printer. *printer* is the name of a print queue that was disabled by a PTRDISABLE command.

On UniData for Windows Platforms, PTRENABLE resumes printing after you pause the printer through Start > Setting > Printers. Only users with Full Control permissions on a printer can control the printer with PTRDISABLE and PTRENABLE. Check **Permissions** on the **Security** tab of the printers **Properties** sheet to determine who has permissions.

Tip: Use the SETPTR command to assign a name to printer.

Use the PTRENABLE command to load new forms into the printer and clear paper jams.

Examples

In the following example, taken from UniData for UNIX, UniData enables printer queue hpzone3, which allows all print jobs destined for this queue to print:

```
:PTRENABLE hpzone3
printer "hpzone3" now enabled
:
```

In the next example, taken from UniData for Windows NT, UniData enables a local printer called LETTER, which allows all print jobs sent to this printer to print:

```
:PTRENABLE LETTER
:LISTPTR
Unit.. Printer.................
Port.....................Status..
0 LETTER \\DENVER4\hpzone3 Running
1 \\DENVER4\hpzone3 hpzone3 Running
2 LEGAL \\DENVER4\hpzone3 Running
3 \\DENVER4\hpzone2 hpzone2 Running
:
```

Related Commands

PTRDISABLE, SETPTR

QUIT

QUIT is a synonym for the BYE command. For more information, see BYE.

Synonyms

BYE, LO

READDICT.DICT

Syntax

READDICT.DICT

Synonym

READDICT-DICT

Description

The ECL **READDICT.DICT** command reloads DICT.DICT into virtual memory. Execute READDICT.DICT to apply changes to DICT.DICT made during the current UniData session.

Note: UniData loads DICT.DICT into memory once at the beginning of each UniData session to improve performance by eliminating the need to repeatedly open and read this frequently used file.

READDICT.DICT first looks for a pointer in the VOC to a local DICT.DICT file. If it exists, UniData reloads that version. If not, UniData reloads the global version, located in *udthome*/sys/DICT.DICT on UniData for UNIX or *udthome*\sys\DICT.DICT on UniData for Windows Platforms.

READDICT.DICT displays no messages — it just returns you to the ECL prompt after completion.

For more information about the DICT.DICT dictionary, see *Using UniData*.

REBUILD.FILE

Syntax

REBUILD.FILE *filename*

Synonym

REBUILD-FILE

Description

The ECL **REBUILD.FILE** command rebuilds a dynamic hashed file, splitting or merging groups as needed, based on the split and merge thresholds. REBUILD.FILE checks every group in the file for a split load and then for a merge.

This command is useful when many processes access the same dynamic file and some restriction prevents splitting or merging. The command is also useful after executing CONFIGURE.FILE to redistribute the keys and data in accordance with a new modulo, split load, merge load, or split/merge type. REBUILD.FILE works only on dynamic hashed and dynamic hashed multilevel subfiles.

For more information on dynamic files, see *Using UniData.*

Warning: Do not rebuild files when users are accessing them. File corruption could result.

Examples

For the following example memresize changed the modulo of a copy of the INVENTORY demo database file from 19 to 3. The guide utility suggests rebuilding the file, and REBUILD.FILE rebuilds the file:

```
:!guide INVENTORY -o
INVENTORY
Basic statistics:
File type.............................. Recoverable Dynamic
Hashing
File size
[dat001]............................... 4096
[over001].............................. 14336
File modulo............................ 3
File minimum modulo.................... 3
.
.
.
Group count:
Number of level 1 overflow groups....... 13
Primary groups in level 1 overflow...... 3
Primary groups over split factor........ 3
.
.
.
Management advice:
Running REBUILD.FILE may improve performance
for access to the file. This conclusion was reached
for the following reasons:
- File has 3 groups over split load.

:REBUILD.FILE INVENTORY

:!guide INVENTORY -O
INVENTORY
Basic statistics:
File type.............................. Recoverable Dynamic
Hashing
File size
[dat001]............................... 12288
[over001].............................. 18432
File modulo............................ 11
File minimum modulo.................... 3
File split factor...................... 60
File merge factor...................... 40
...
Group count:
Number of level 1 overflow groups....... 12

Primary groups in level 1 overflow...... 6
...
Predicted optimal size:
```

```
Records per block..................... 10
Percentage of near term growth........ 10
Scalar applied to calculation......... 0.00
Block size............................ 1024
Modulo................................ 11
...
```

Notice that executing REBUILD.FILE changed the current modulo from 3 to 11 and guide no longer recommends rebuilding the file.

Related Command

CONFIGURE.FILE

RECORD

Syntax

RECORD *filename record*

Description

The ECL **RECORD** command displays the group to which a particular record is hashed. If record does not exist, UniData displays the group to which the record would hash if it was added.

UniData indicates whether the record exists, and, if more than one record is in the group, displays the ID and length for each record.

Note: UniData hashes records to groups based on the file modulo. UniData group numbering starts with 0 (zero), rather than 1.

Tip: Use RECORD to locate and correct record IDs that appear to be duplicates because one contains nonprinting characters. First, LIST @IDs for a file (without sorting). Review the list, locating duplicate keys, then execute RECORD on adjacent records. Depending on the modulo of the file, you may find the real key and the duplicate in different groups. Then write a UniBasic program to open the file and delete the offending record (ECL DELETE will not let you specify a key containing a nonprinting character).

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	A UniData hashed file.
record	A record ID in filename.

RECORD Parameters

Examples

The following example checks record ID 10086 against the CLIENTS demo database file and finds that it is hashed to group 14. UniData also displays all record IDs and the length of each record in the group.

```
:RECORD CLIENTS 10086
10086 hashed to group 14 and was found
# length @ID
0 100 9994
1 105 10029
2 97 10010
3 101 9975
4 104 10067
5 117 10048
6 112 10086
:
```

In the following example, as indicated, record 80 does not exist in the CLIENTS file.

```
:RECORD CLIENTS 80
80 hashed to group 0 and was not found

# length @ID
0 96 9999
1 102 10034
2 110 9980
3 115 10015
4 102 10072
5 110 10053
6 108 10091
:
```

Here we add record 80 and execute RECORD again.

```
:COPY FROM CLIENTS 9999, 80
1 records copied
:RECORD CLIENTS 80
80 hashed to group 0 and was found
# length @ID
0 96 9999
1 102 10034
2 110 9980
3 115 10015
4 102 10072
5 110 10053
6 108 10091
7 9680
:
```

RELEASE

Syntax

RELEASE *filename* [*record*]

Description

The ECL **RELEASE** command clears locks placed on a file or record by UniData or UniBasic commands that set locks. For more information on UniData locks, see *Developing UniBasic Applications* and *Administering UniData.*

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	UniData file name.
record	A locked record in *filename.*

RELEASE Parameters

RELEASE.ITEMS

Syntax

RELEASE.ITEMS

Synonym

RELEASE-ITEMS

Description

The ECL **RELEASE.ITEMS** command clears all record locks set by your process.

For more information on UniBasic and UniData locks, see *Developing UniBasic Applications* and *Administering UniData.*

Note: This command does not release locks set by other processes even when executed by someone logged in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms. Root or Administrator can execute the ECL SUPERRELEASE command to clear other users locks.

List active locks with LIST.READU. GETUSER displays your user number.

Examples

The following UniBasic program locks a record with the RECORDLOCKU command and releases the lock by executing RELEASE.ITEMS:

```
* TESTING LOCKING/RELEASE COMMANDS *
OPEN “ORDERS” TO A ELSE STOP “CANNOT OPEN”
LID = “801”
RECORDLOCKU A, LID ON ERROR STOP
PRINT “RECORD LOCKED WITH RECORDLOCKU”
EXECUTE “LIST.READU”
SLEEP 3
EXECUTE “RELEASE.ITEMS”
PRINT “EXECUTING RELEASE.ITEMS COMMAND”
PRINT “LISTING LOCKS AGAIN”
EXECUTE “LIST.READU”
SLEEP 3
END
```

The next example locks and unlocks the record by running the preceding program:

```
:RUN BP TEST_1
RECORD LOCKED WITH RECORDLOCKU
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
2 7093 1283carolw ts/1 ORDERS 219261 107380 801 X 14:17:27 Jun 08
EXECUTING RELEASE.ITEMS COMMAND
LISTING LOCKS AGAIN
:
```

RESIZE

Syntax

RESIZE [DICT] *filename* [*modulo* [,*block.size.multiplier* | -]] [TYPE [0 | 1]]

Description

The ECL **RESIZE** command changes the size of a static data file. UniData resizes the file based on its original modulo and overflow status, or you can change its modulo.

To resize a file, sufficient virtual memory to hold a copy of the file must be available.

RESIZE applies this formula to calculate modulo for the file being resized:

```
(actual filesize / blocksize)*0.9
```

Each time you execute RESIZE, UniData applies the same calculation, causing the file size to increase by two blocks.

For more information on selecting an optimum file modulo number, see *Using UniData.*

Warning: Resizing a file while another process is updating the file corrupts the file.

Tip: The ECL system-level memresize command executes more quickly and offers more options.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	Resizes the file according to its original modulo.
DICT	Indicates that a static dictionary file is to be resized.
filename	The static file to resize.
modulo	New modulo for the resized static file. Must be a prime number.
block.size.multiplier	An integer between 0 and 16, that determines block size: ■ 0 – 512 bytes ■ 1 – 1,024 bytes ■ 2 – 2,048 bytes ■ 4 – 4,096 bytes ■ 8 – 8,192 bytes ■ 16 or greater – 16,384 bytes
-	Resizes the file according to its actual size, thus correcting overflows.
TYPE[0 | 1]	Hash type for the resized file.

RESIZE Parameters

Examples

In the following example, the CLIENTS demo database static file has a modulo of 19 and size of 21,504 bytes. Notice that the 21 blocks of the file consist of one header block, 19 data blocks (the maximum allowed by the modulo), and one data overflow block. This is confirmed by the message on line 5 that one group is in level-one overflow.

```
:FILE.STAT CLIENTS
File name = CLIENTS
Number of groups in file (modulo) = 19
Static hashing, hash type = 0
Block size = 1024
File has 1 groups in level one overflow.
Number of records = 130
Total number of bytes = 14452
Average number of records per group = 6.8
Standard deviation from average = 0.5
Average number of bytes per group = 760.6
Standard deviation from average = 61.3
Average number of bytes in a record = 111.2
Average number of bytes in record ID = 5.7
Standard deviation from average = 8.7
Minimum number of bytes in a record = 93
Maximum number of bytes in a record = 140
Minimum number of fields in a record = 10
Maximum number of fields in a record = 10
Average number of fields per record = 10.0
Standard deviation from average = 0.0
The actual file size in bytes = 21504.
:
```

The next example, resizes the file using a modulo of 23. Notice the changed statistics and correction of the overflow.

```
:RESIZE CLIENTS 23
CLIENTS RESIZED from 19 to 23
:
:FILE.STAT CLIENTS
File name = CLIENTS
Number of groups in file (modulo) = 23

Static hashing, hash type = 0
Block size = 1024
Number of records = 130
Total number of bytes = 14452
Average number of records per group = 5.7
Standard deviation from average = 0.7
Average number of bytes per group = 628.3
Standard deviation from average = 75.3
Average number of bytes in a record = 111.2
Average number of bytes in record ID = 5.7
Standard deviation from average = 8.7
Minimum number of bytes in a record = 93
Maximum number of bytes in a record = 140
Minimum number of fields in a record = 10
Maximum number of fields in a record = 10
Average number of fields per record = 10.0
Standard deviation from average = 0.0
The actual file size in bytes = 24576.
:
```

In the next example, records are added to a file called EMPLOYEES, which was created for this example. FILE.STAT displays the following statistics for EMPLOYEES:

- Number of groups in file this is the modulo number
- Number of groups in level two overflow and the “Please resize.” message

- Suggested resize modulo on the last line of the display

```
:COPY FROM ORDERS TO EMPLOYEES ALL
192 records copied
:FILE.STAT EMPLOYEES
File name = EMPLOYEES
Number of groups in file (modulo) = 2
Static hashing, hash type = 0
Block size = 1024
File has 4 groups in level two overflow. Please resize.

Number of records = 323
Total number of bytes = 28347
Average number of records per group = 161.5
Standard deviation from average = 0.7
Average number of bytes per group = 14173.5
Standard deviation from average = 46.0
Average number of bytes in a record = 87.8
Average number of bytes in record ID = 4.7
Standard deviation from average = 35.0
Minimum number of bytes in a record = 38
Maximum number of bytes in a record = 271
Minimum number of fields in a record = 7
Maximum number of fields in a record = 10
Average number of fields per record = 8.2
Standard deviation from average = 1.5
The actual file size in bytes = 35840.
Suggested resize modulo = 37.
:
```

The next example resizes EMPLOYEES because of inclusion of the - option. The modulo is changed to 31.

```
:RESIZE EMPLOYEES -RESIZE
file EMPLOYEES to 31.
EMPLOYEES RESIZED from 2 to 31
```

Now, look at the file statistics again to see the other changes made:

```
:FILE.STAT EMPLOYEES
File name = EMPLOYEES
Number of groups in file (modulo) = 31
Static hashing, hash type = 0
Block size = 1024
File has 19 groups in level one overflow.
Number of records = 323
Total number of bytes = 28347
Average number of records per group = 10.4
Standard deviation from average = 0.8
Average number of bytes per group = 914.4
Standard deviation from average = 146.0
Average number of bytes in a record = 87.8
Average number of bytes in record ID = 4.7
Standard deviation from average = 35.0
Minimum number of bytes in a record = 38
Maximum number of bytes in a record = 271
Minimum number of fields in a record = 7
Maximum number of fields in a record = 10
Average number of fields per record = 8.2
Standard deviation from average = 1.5
The actual file size in bytes = 52224.
:
```

Related Command

memresize

REUSE.ROW

Syntax

REUSE.ROW [0 | 1]

Synonym

REUSE-ROW

Description

The ECL **REUSE.ROW** command determines whether a linefeed is executed when the UniBasic PRINT @ function references column only, for instance, PRINT @(10) rather than PRINT @(3,10).

For more information about programming in UniBasic, see *Developing UniBasic Applications*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
0	Default. A line feed is applied before the cursor moves to the specified column.
1	The cursor moves to the specified column on the same row.

REUSE.ROW Options

RUN

Syntax

RUN *directory.file program* [[-N | (N | (N)] | -G | -D | -E | -F]

Description

The ECL **RUN** command executes a compiled UniBasic program.

For more information about programming in UniBasic, see *Developing UniBasic Applications*.

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
directory.file	A UniData DIR file that contains a compiled UniBasic program. You must have a pointer to this file in your VOC.
program	A compiled UniBasic program.
-N | (N | (N)	The screen display scrolls without stopping. Without this option, scrolling stops at the bottom of each page, prompting the user to press return to continue.
-G	Executes a cross-reference report (program profile).
-D	Invokes the UniBasic debugger immediately.
-E	Invokes the UniBasic debugger when a runtime error occurs.
-F	Invokes the UniBasic debugger when a fatal error occurs.

RUN Parameters

Example

The following example shows the output of the RUN command with the -D parameter for a program called PSTLCODE_FMT in the BP_SOURCE file of the demo database. Notice that UniData invokes the UniBasic debugger due to a problem at line 7 of the program.

```
:RUN BP_SOURCE PSTLCODE_FMT -D
***DEBUGGER called at line 7 of program BP_SOURCE/_PSTLCODE_FMT
!
```

Tip: To escape from the UniBasic debugger and return to the ECL colon prompt, enter END.

sbcsprogs

Syntax

sbcsprogs

Description

The system-level **sbcsprogs** command reports the number of users sharing globally cataloged UniBasic programs.

Note: Use this command at the operating system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Example

The following example shows sbcsprogs output. Reference Count indicates the number of users currently using the corresponding program.

```
% sbcsprogs
Program Name                                Reference Count
/disk1/ud60/sys/CTLG/s/SCHEMA_FILE_CHECK        1
/disk1/ud60/sys/CTLG/s/SCHEMA_SQLNAME_ATTRIBUTES      1
/disk1/ud60/sys/CTLG/s/S_FILE_EXIST_PRIVILEGE   1
/disk1/ud60/sys/CTLG/s/S_VALID_SCHEMA_CHECK           1
/disk1/ud60/sys/CTLG/s/SCHEMA_FILE_LIST         1
/disk1/ud60/sys/CTLG/s/SCHEMA_DEPENDENT_VIEWS 1
/disk1/ud60/sys/CTLG/s/SCHEMA_CRT_READ_MAP      1
/disk1/ud60/sys/CTLG/s/SCHEMA_LIST_USERS              1
/disk1/ud60/sys/CTLG/s/S_GET_FILE_OWNER         1
/disk1/ud60/sys/CTLG/s/SCHEMA_FILE_ATTRIBUTES 1
/disk1/ud60/sys/CTLG/s/SCHEMA_UNIQUE_NAME             1

/disk1/ud60/sys/CTLG/s/S_VALID_NAME_CHECK       1
/disk1/ud60/sys/CTLG/s/S_OPEN_SCHEMA_TABLES     1
/disk1/ud60/sys/CTLG/s/SCHEMA_VIEW_TYPE         1
/disk1/ud60/sys/CTLG/s/S_UPD_SCHEMA_TABLES            1
/disk1/ud60/sys/CTLG/s/S_DB_TYPE_CONV           1
/disk1/ud60/sys/CTLG/s/SCHEMA_SUBTABLE_ATTRIBUTES     1
...
```

SET.DEC

Syntax

SET.DEC [*char*]

Synonym

SET-DEC

Description

The ECL SET.DEC command changes the character used to display the decimal point. Any ASCII character is acceptable for char. The default character is period (.). The setting is effective for the current udt session only.

Tip: Use this command to set the decimal representation for displaying money. For more information on localizing UniData for use with your language and monetary system, see UniData International.

Examples

In the following example, the period is displayed for the decimal point:

```
:LIST INVENTORY PRICE
LIST INVENTORY PRICE 17:27:51 Jun 22 1999 1
INVENTORY. Price.....
53050 $369.95
56060 $98.99

57030 $2,995.95
...
```

The next example changes the decimal character to a comma (,):

```
:SET.DEC ,
:
```

The LIST command demonstrates use of the new decimal character:

```
:LIST INVENTORY PRICE
LIST INVENTORY PRICE 17:32:00 Jun 22 1999 1
INVENTORY. Price.....
53050 $369,95
56060 $98,99
57030 $2,995,95
...
```

SET.LANG

Syntax

SET.LANG [*language* | CURRENT | AVAILABLE]

Synonym

SET-LANG

Description

The ECL SET.LANG command changes the language within the current language group. You can specify the language you want by spelling it out in uppercase letters, or by typing the UniData language number. Type AVAILABLE after SET.LANG to display a list of languages to choose from, or type CURRENT to display the current language setting. If you enter SET.LANG without parameters, UniData displays a usage statement.

For more information on localizing UniData for use with your language and monetary system, see *UniData International*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
language	UniData language name. You must enter the language name in uppercase.
CURRENT	Display the settings for the current language.
AVAILABLE	Display the available languages in the current language group.

SET.LANG Parameters

Examples

The following example displays the settings for the current language:

```
:SET.LANG CURRENT
udtlang name: ENGLISH
Date format: 0
Decimal point: .
Thousand delimiter: ,
Money sign: $
:
```

This example displays the languages available within the current group, which is ENGLISH:

```
:SET.LANG AVAILABLE
ENGLISH
ENGLISH_UK
:
```

Next, we change the language to ENGLISH_UK, and execute SET.LANG CURRENT to display the changed language:

```
:SET.LANG ENGLISH_UK
Language 'ENGLISH_UK' assigned!
:SET.LANG CURRENT
udtlang name: ENGLISH_UK
Date format: 0
Decimal point: .
Thousand delimiter: ,
Money sign: $
:
```

Tip: If UniData displays an error message, it could mean the message defaults file for the language does not exist. (Message defaults files reside in udthome/sys on UniData for UNIX or udthome\sys on UniData for Windows Platforms.) See UniData International for information on the language-specific message files.

SET.MONEY

Syntax

SET.MONEY *sign* [POST | PRE]

Synonym

SET-MONEY

Description

The ECL SET.MONEY command changes the UniData delimiter that represents a currency sign. Use this command when you need to change the currency sign from the default set for your language. When SET.MONEY is used without an argument, the command returns a usage message.

The currency sign follows the number in some regions, and UniData honors this convention when the POST option is used. If POST is not set, the currency sign precedes the amount.

For more information about this command and other commands related to using non-English language versions of UniData and the message defaults file, see *UniData International*.

Tip: To insert a space between the currency sign and the amount, use an extra space after the SET.MONEY command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
sign	Character that represents currency.
POST	Currency sign follows the amount.
PRE	Currency sign precedes the amount. This is the default position.

SET.MONEY Parameters

Example

In the following example, the SET.MONEY command sets the currency denomination to the number sign (#).

```
:SET.MONEY #
:
```

Now, when you display data that uses a currency sign, UniData uses the symbol you assigned with the SET.MONEY command:

```
:LIST ORDERS PRICE
LIST ORDERS PRICE 14:24:30 Jun 08 1999 1
ORDERS.... Price.....
813 #99.96
#199.87
#69.94
928 #159.95
859 #200.00
974 #99.95
905 #59.95
790 #159.94
#159.94
...
```

SET.THOUS

Syntax

SET.THOUS *char*

Synonym

SET-THOUS

Description

The ECL SET.THOUS command changes the character that indicates a break for thousands. A comma (,) is the default character.

This command has the following restrictions:

- The decimal point and thousand delimiter cannot be the same character. For instance, when SET.DEC is set as a comma (,), the period symbol (.) loses its functionality as a decimal point except in the constants in UniBasic programs and dictionary items.
- Decimal and thousand delimiters cannot be changed in the middle of the execution of a UniBasic program.

Examples

The following example lists some totals from the ORDERS demo file. Notice that UniData uses a comma for the thousands break character:

```
:LIST ORDERS GRAND_TOTAL
LIST ORDERS GRAND_TOTAL 14:37:42 Jun 08 1999 1
ORDERS.... Grand Total...
912 $779.70
801 $1,799.00
941 $13,999.90

805 $47,555.29
...
```

In the following example, UniData sets the one thousand break character to a period (.).

```
:SET.THOUS .
:
```

The next example shows the display of totals after the thousands character has changed to a period (.):

```
:LIST ORDERS GRAND_TOTAL
912 $779.70
801 $1.799.00
941 $13.999.90
805 $47.555.29
...
```

SET.TIME

Syntax

SET.TIME *hh:mm*[*:ss*]

Synonyms

SET-TIME, SETTIME

Description

The ECL SET.TIME command sets the time for the entire system.You enter the time in a 24-hour format (military time) where *hh* represents hours, *mm* minutes, and *ss* seconds. The seconds portion of the time is optional.

Note: To execute the SET.TIME command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Warning: Do not change the system time while the Recoverable File System (RFS) is running. If you do, you will corrupt the time stamps for RFS.

Example

In the following example, the SET.TIME command sets the system time to 3 minutes and 4 seconds past 1:00 pm:

```
:SET.TIME 13:03:04
```

SET.WIDEZERO

Syntax

SET.WIDEZERO [*float.number*]

Synonym

SET-WIDEZERO

Description

The ECL **SET.WIDEZERO** command sets a range used for comparing very small numbers. When two floating point numbers differ by less than that range, UniData considers them to be equal. The SET.WIDEZERO setting is active for the current UniData session only.

If you do not include float.number, UniData displays the current setting. You must enclose scientific notation, such as 1.0E-10 in quotation marks.

The default value is 0.0 to be backwardly compatible with previous versions of UniBasic.

Examples

This example displays the current wide zero setting:

```
:SET.WIDEZERO
Wide Zero: 0.00E+00
:
```

In the following example, the SET.WIDEZERO command sets the range at 0.001. In UniBasic, if A=5.9915 and B=5.991, then A=B is true because the difference between the two numbers, 0.0005 is less than the wide zero value 0.001.

```
:SET.WIDEZERO 0.001
```

SETDEBUGLINE

Syntax

SETDEBUGLINE *port*

Description

The ECL **SETDEBUGLINE** command makes a terminal port number (port) attachable for dual-terminal debugging with the UniBasic debugger.

For more information on UniBasic and the UniBasic debugger, see *Developing UniBasic Applications.*

Example

In the following example, UniData makes a port attachable:

```
:SETDEBUGLINE ttyv0
:
```

Related Commands

DEBUGLINE.ATT, DEBUGLINE.DET, UNSETDEBUGLINE

SETFILE

Syntax

SETFILE [[*path*][*pointer*] [OVERWRITING]]

Description

The ECL **SETFILE** command creates a file pointer in the VOC for a UniData file. SETFILE does not work on dictionaries, multilevel subfiles, or subdirectories. SETFILE assigns the correct file type to the file pointer.

You can set a pointer in a UniData VOC file to a data file in another UniData account. This feature allows users working in different UniData accounts to share data files. There are two points to remember about setting a VOC pointer:

- A VOC pointer is internal to UniData. On UniData for UNIX, it is not the same thing as a UNIX link. Because of this, even backup utilities that follow symbolic links do not automatically follow VOC pointers.
- Setting a VOC pointer does not alter the physical location of the data file. Although you can access the file from the directory where the pointer resides, the physical location of the file and its indexes remains unchanged.

Note: When UDT.OPTIONS 87 is on and you delete a synonym for a file in another account with DELETE.FILE, UniData deletes both the file pointer in the current directory and the file in the remote account.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
no parameter	UniData prompts for all required information.
path	The full or relative path to the file. If you do not indicate path, UniData prompts for a "treename." You can specify a relative path, but you may not include variables, such as @UDTHOME.
pointer	The name of the VOC entry that will be the file pointer. If you do not indicate a pointer name, UniData prompts for a "filename."
OVERWRITING	Overwrites the VOC entry for an existing file pointer of the same name. **Warning**: UniData does not prompt for confirmation before overwriting the VOC entry.

SETFILE Parameters

Examples

Start from the directory that contains the VOC file where you wish UniData to create the entry for the file pointer. For the following series of examples, taken from UniData on UNIX, that directory is /home/claireg/demo. In the next example, the UNIX pwd command confirms the location:

```
:!pwd
/home/claireg/demo
:
```

Creating a New File Pointer

In the following example, UniData creates a file pointer named ACCOUNTS to the UniData file CLIENTS, which resides in another account (/usr/ud60/demo). Before establishing the pointer, UniData lists the parameters for the pointer and asks for confirmation.

```
:SETFILE /disk1/ud60/demo/INVENTORY stock.file
Establish the file pointer
Tree name /disk1/ud60/demo/INVENTORY
Voc name stock.file
Dictionary name /disk1/ud60/demo/D_INVENTORY
Ok to establish pointer(Y/N) = y
SETFILE completed.
:
```

Use the CT command to display the VOC entry for the new file pointer:

```
:CT VOC stock.file
VOC:
stock.file:
F
/disk1/ud60/demo/INVENTORY
/disk1/ud60/demo/D_INVENTORY
:
```

After creating the VOC entry in your own account, you can execute the ECL LIST command to list the INVENTORY file from that account. Here, the UNIX pwd command confirms the current location, and ECL LIST command lists the stock.file file:

```
:!pwd
/home/claireg/demo
:LIST stock.file PROD_NAME
LIST stock.file PROD_NAME 14:57:02 Jun 08 1999 1
Product
INVENTORY. Name......
53050 Photocopie
r
56060 Trackball
57030 Scanner
31000 CD System
2
10140 Camera
11001 Computer
10150 Camera
...
```

Changing an Existing File Pointer

The OVERWRITING keyword changes the VOC entry pointer. The following example shows the VOC entry for the CLIENTS demo file in /home/claireg/demo:

```
:CT VOC CLIENTS
VOC:
CLIENTS:
F
CLIENTS
D_CLIENTS
:
```

The next example changes the file pointer to the CLIENTS file in another account:

```
:SETFILE /disk1/ud60/demo/CLIENTS CLIENTS OVERWRITING
Establish the file pointer
Tree name /disk1/ud60/demo/CLIENTS

Voc name CLIENTS
Dictionary name /disk1/ud60/demo/D_CLIENTS
SETFILE completed.
```

To compare the new file pointer to the original one, use the CT command. Notice that UniData points to a new location for the CLIENTS file.

Warning: OVERWRITING does not prompt for confirmation before removing the VOC pointer. Also, without the VOC pointer, some users may be unable to access a file in another account.

```
:CT VOC CLIENTS
:CT VOC CLIENTS
VOC:
CLIENTS:
F
/disk1/ud60/demo/CLIENTS
/disk1/ud60/demo/D_CLIENTS
```

Executing SETFILE with No Parameters

In the following example, UniData prompts for required information:

```
:SETFILE
Enter treename = /home/claireg/demo
Enter filename = CLIENTS
Establish the file pointer
Tree name /home/claireg/demo
Voc name CLIENTS
Ok to establish pointer(Y/N) = Y

Pointer CLIENTS already exists, do you want to overwrite(Y/N) = Y
SETFILE completed.
```

Here is the VOC entry for the new file pointer:

```
:CT VOC CLIENTS
VOC:

CLIENTS:
DIR
:
```

Creating File Name Synonyms

You can create a synonym file name by creating a second file pointer to an existing file. You can then use the original or synonym file name to access the file.

Note: Delete the VOC entry that creates a synonym by executing:

DELETE.FILE synonym name

Re-creating a Deleted File Pointer

To demonstrate recreating a deleted file pointer, we first delete the VOC pointer to the CLIENTS demo file. The CT command reveals that the VOC pointer no longer exists, and an attempt to display the records in CLIENTS generates a message that CLIENTS is not a file name.

```
:DELETE VOC CLIENTS
'CLIENTS' deleted.
:CT VOC CLIENTS
VOC:
CLIENTS is not a record in VOC.
:LIST CLIENTS

Not a filename :
CLIENTS
:
```

Next, we reestablish the VOC pointer by using SETFILE and pointing to the demo directory, then confirm with CT that the pointer again exists. Finally, LIST displays the records in the file:

```
:SETFILE /disk1/ud60/demo/CLIENTS
Enter filename = CLIENTS
Establish the file pointer
Tree name /disk1/ud60/demo/CLIENTS
Voc name CLIENTS
Dictionary name /disk1/ud60/demo/D_CLIENTS
Ok to establish pointer(Y/N) = Y
SETFILE completed.
:CT VOC CLIENTS
VOC:
CLIENTS:
F
/disk1/ud60/demo/CLIENTS
/disk1/ud60/demo/D_CLIENTS

:LIST CLIENTS
LIST CLIENTS NAME COMPANY ADDRESS CITY STATE ZIP COUNTRY PHONE
PHONE_TYPE
15:24:05 Jun 08 1999 1
CLIENTS 9999
Name Paul Castiglione
Company Name Chez Paul
Address 45, reu de Rivoli
City Paris
State/Territory
Postal Code 75008
Country France
...
```

SETLINE

Syntax

SETLINE [*line* [*path*]]

Description

The ECL **SETLINE** command initializes a communication line for use during the current UniData session. If you do not specify a parameter, UniData displays the current setting.

SETLINE creates an editable ASCII file. On UniData for UNIX, this file is located in *udthome*/sys/lineinfo. On UniData for Windows Platforms, this file is located in *udthome*\sys\lineinfo.

Note: To initialize a line, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms. However, any user can use the SETLINE command to get line information.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
line	Line unit number from 0 to 499 of a device to be initialized. If you do not indicate a line number, UniData returns all line information. If you indicate the line number without specifying *path* or *devname*, UniData returns information about *line*.
path	On UniData for UNIX, path and name for the physical device, for instance, /dev/tty01. On UniData for Windows Platforms, identifier for serial device, for instance, COM1.

SETLINE Parameters

Example

In the following example, UniData displays the path and name for the device to which line 0 is currently attached:

```
:SETLINE 0
LINE#......: 0
DEVICE-NAME: /dev/pty/ttyv6
:
```

Related Commands

UniData

LINE.ATT, LINE.DET, LINE.STATUS, PROTOCOL, UNSETLINE

UniBasic

GET, SEND For information, see the UniBasic Commands Reference.

SETOSPRINTER

Syntax

SETOSPRINTER ["UNIX_spooler_command [options]"]

Description

The ECL **SETOSPRINTER** command executes a UNIX spooler command. You must enclose the spooler command and options in quotation marks. To reset the printer command to the default, issue SETOSPRINTER with no parameters.

Note: This command is supported in UniData for UNIX only.

The command you set with SETOSPRINTER must be listed in the configuration file UDTSPOOL.CONFIG in udthome/sys. You can edit this file, but write access may be restricted. For more information about editing the UDTSPOOL.CONFIG file, see Administering UniData.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
UNIX_spooler_command	A UNIX spooler command. Must be enclosed in quotation marks. Must be defined in /*udthome*/sys/UDTSPOOL.CONFIG.
options	UNIX spooler command options. Must be enclosed in quotation marks.

SETOSPRINTER Parameters

You can display the setting for the system spooler with the ECL LIMIT command, which lists maximums for all UniData environment variables:

```
:LIMIT
...
U_LPCMD: System spooler name = lp -c.
...
```

In the following example, SETOSPRINTER changes the setting for the UNIX spooler command:

```
:SETOSPRINTER “lp”
:LIMIT
...
U_LPCMD: System spooler name = lp -c .
...
```

SETPTR

Syntax

SETPTR *unit*[,*width*,*length*,*topmargin*,*bottommargin*] [,*mode*]
[“*spooler_options*”[,*options*]]

Description (UniData for UNIX)

The ECL **SETPTR** command directs the print spooler for printer unit for the current UniData session.

The SETPTR option defaults are set internally in UDTSPOOL.CONFIG in *udthome*/sys; you can change them only for the current UniData session.

You can configure as many as 31 printer units in a UniData session, including the default printer (defined as 0). You can configure as many as 255 printer units per UniData installation (units 0 through 254). UniData uses the UNIX print spooler command usually lp or lpr.

Tip: To make work sessions consistent among users, place SETPTR commands in the LOGIN paragraph for each UniData account.

Note: If UDT.OPTIONS 84 is on, and the printer set to a _HOLD_ file, UniData displays the hold entry name each time a new hold file is created. With UDT.OPTIONS 84 OFF, UniData displays the _HOLD_ entry name only when SETPTR or SP.ASSIGN is executed.

Parameters (UniData for UNIX)

The following table describes each parameter of the syntax.

Parameter	Description
unit	Number assigned to a given printer through UNIX: 0–255. (The default is 0). If you do not indicate a printer unit number, UniData displays the current printer settings for Unit 0.
width	Number of characters per line: 0–1,024 characters. If you do not want to change this setting, enter a comma as a placeholder.
length	Number of lines per page: 1 to 32,767 lines. If you do not want to change this setting, enter a comma as a placeholder.
topmargin	Number of lines to leave blank at the top of each page: 0–25. If you do not want to change this setting, enter a comma as a placeholder.
bottommargin	Number of lines to leave blank at the bottom of each page: 0–25. If you do not wish to change this setting, enter a comma as a placeholder.
mode	Several modes work in conjunction with the SETPTR command. See "SETPTR Modes (UniData for UNIX)" in this section. If you do not want to change this setting, enter a comma as a placeholder.
"*spooler_options*"	UNIX **lp** or **lpr** spooler option. Any parameter that you use with your spooler, you can use with SETPTR. Enclose each option in quotation marks. For example: "-o noeject".
options	Report formatting and printer control options. See "SETPTR Options (UniData for UNIX)" in this section.

SETPTR Parameters

SETPTR Modes (UniData for UNIX)

The following table lists the SETPTR modes:

Mode	Description
1	Sends output to the line printer.
2	Directs output to the serial device indicated by the DEVICE option.
3	Sends output to the _HOLD_ file.
6	Sends output to the _HOLD_ file and to the line printer. **Tip**: To print records from the _HOLD_ file, use the ECL "SPOOL" command. Use in conjunction with BANNER or BANNER UNIQUE to store the output in a record you name.
9	Sends output to the line printer and suppresses terminal display of the _HOLD_ entry name.

SETPTR Modes

SETPTR Options (UniData for UNIX)

The following table lists the SETPTR options.

Option	Description
BANNER [*string*]	By default, SETPTR adds a banner page that shows the owner's user ID. You can override the default display with the BANNER option where *string* is a message for the banner page. If you redirect the output to the _HOLD_ file, the print record identifier in the _HOLD_ file becomes P_*string_n*. (The default record identifier in the _HOLD_ file is P_*unit_n*.) *string* can be as long as 96 characters, but cannot contain spaces. It must be followed by a comma, if options follow.
BANNER UNIQUE (*string*)	Places string in the record identifier. By default, the record identifier is P_*unit_n*, where unit is the printer unit number, and *n* is a 4-digit number that increments automatically. If you indicate string, the identifier becomes P_*string_n*. **Note**: This counter is stored in DICT _HOLD_NEXT.HOLD (Attribute 1). The counter automatically rolls back to 1 after incrementing to 9999. Users must have write permissions to DICT _HOLD_ to use this option. *string* can be as long as 96 characters, but cannot contain spaces. It must be followed by a comma, if options follows.
BRIEF	Suppresses the verification prompt.
COPIES *n*	Prints *n* copies.
DEFER [*time*]	Delays printing until the specified *time* by passing the job to the UNIX **at** command. Make sure you know what time zone your machine uses — it may differ from your local time. This option requires that you be signed in as root. **Tip**: For the syntax for time, see your UNIX documentation or the man pages for information on the "at" command.

SETPTR Options

Option	Description
[DEST | AT] *unit*	Directs the print job to print queue *unit*, rather than to a device number. For example, if you have multiple printers set up to run only checks, you can use this option to have checks print on the first available check printer. In this case, your spooler must support classes.
DEVICE *filename*	Directs output to the UNIX device indicated by *filename*. Used only with mode 2.
EJECT	Ejects a blank page at the end of the print job.
NOEJECT	Suppresses the form feed at the end of the print job.
FORM *form*	Assigns a previously defined form to the spooler. DEST and FORM are concatenated to designate the print queue name.
LNUM	Prints line numbers in the left margin.
NFMT | NOFMT	Suspends all UniData print formatting. Use this if you intend to control print formatting with an application.
NHEAD | NOHEAD	Suppresses the banner.
NOMESSAGE	Suppresses messages from the UNIX lp spooler.
OPEN	Sends output to a file until an SP.CLOSE statement executes for this print *unit*. This allows you to save multiple reports in one file.

SETPTR Options (continued)

Tip: Some of the SETPTR options are configurable in the UDTSPOOL.CONFIG file located in udthome/sys. For more information about editing this ASCII text file, see Administering UniData.

Examples (UniData for UNIX)

To find out the current SETPTR settings for unit 0, enter the SETPTR command without any options. In the example that follows, notice the Spooler & options setting which is set for the lp UNIX spooler command and the -c spooler option.

```
:SETPTR
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 1
Options are:
Spooler & options: lp -c
:
```

In the following example UniData assigns the following printer parameters:

- Column width of 45 characters
- Page length of 10 lines
- Top margin of 15 lines
- Leave the bottom margin undefined (notice the extra comma, which acts as a placeholder)
- Use mode 3, which directs output to the _HOLD_ file
- Use the BANNER option to name the _HOLD_ file record Summary
- Suspend system formatting

```
:SETPTR 0,45,10,15,,3,BANNER Summary,NOFMT
Unit 0
Width 45
Length 10
Top margin 15
Mode 3
Options are:
Banner Summary
Nfmt
OK to set parameters as displayed?(enter Y/N) y
Hold Entry _HOLD_/SummaryUnit 0
:
```

Printing Multiple Reports in a Single Print Job

The next example uses the OPEN option with SETPTR to print multiple reports, which UniData recognizes as a single print job, to a printer or the _HOLD_ file. Once all the print statements have been issued, you must use the ECL SP.CLOSE command to spool the print the job.

This sample SETPTR command sequence accomplishes the following:

1. Controls settings for report formatting and printing, including:
   - Leaves settings for page width, length, top margin, and bottom margin unchanged (the commas act as placeholders for these parameters)
   - Sends output to the _HOLD_ file by using mode 3
   - Labels the _HOLD_ file record Multiples by using the BANNER option
   - Opens a print statement input session by using the OPEN option, thus allowing the user to enter multiple print statements
2. Uses LIST commands to generate multiple reports (including the LPTR keyword after each statement to direct the statements to the printer spooler)
3. Uses the SP.CLOSE command to close the print statement input session and prints the job to the _HOLD_ file

```
:SETPTR 0,,,,,3,BANNER Multiples,OPEN
Unit 0
Mode 3
Options are:
Banner Multiples
OPEN
OK to set parameters as displayed?(enter Y/N) Y
Hold Entry _HOLD_/Multiples
:LIST CLIENTS LNAME LPTR
:LIST INVENTORY PROD_NAME LPTR
:LIST ORDERS GRAND_TOTAL LPTR
:SP.CLOSE
:
```

Now, if you look at the contents of the _HOLD_ file you will see that it contains the job called Multiples.

```
:LS _HOLD_
Multiples
:
```

Tip: To see the contents of a record in the _HOLD_ file, use your system text editor or the SP.EDIT command.

SETPTR (UniData for Windows Platforms)

On UniData for Windows Platforms, the SETPTR command maps printers defined in Windows systems (either local printers or network print devices) to logical unit numbers.

With SETPTR, you can define up to 31 logical printer units in a single UniData session. Throughout UniData, you can define up to 255, but only 31 can be defined in a single user session.

The default print unit in UniData is unit 0. You can map this default unit to a particular device with SETPTR. If you do not map it explicitly, unit 0 is automatically mapped to one of two printers:

- The default printer for your Windows system. Check **Settings > Printers** to determine which printer is the default.
- A printer identified by the system environment variable UDT_DEFAULT_PRINTER. This definition overrides the default printer for the Windows NT system. Use the MS-DOS SET command or select **Settings > Control Panel > System > Environment** to display or modify UDT_DEFAULT_PRINTER.

Parameters (UniData for Windows Platforms)

The following table describes each parameter of the syntax.

Parameter	Description
unit	Logical printer unit number; internal to UniData; you can map this to a Windows printer with the DEST option. Valid values range from 0 through 254. The default is 0.
[width]	The number of characters per line: must be from 0 to 256. The default is 132.
[length]	The number of lines per page. Valid values range from 1 to 32,767 lines. The default is 60.

SETPTR Parameters

Parameter	Description
[topmargin]	The number of lines to leave blank at the top of each page. Valid values range from 0 to 25. The default is 3.
[bottommargin]	The number of lines to leave blank at the bottom of each page; must be from 0 to 25. The default is 3.
[mode]	The output direction. The default is 1. See separate table.
[*"spooler_options"]*	Options that are valid with the Windows spooler. See separate table for list of supported options. Enclose these options in quotation marks.
[options]	Report formatting and printer control options. See "SETPTR Options" in this section.

SETPTR Parameters (continued)

Note: Users familiar with Pick® conventions should be aware that printer unit numbers set with SETPTR are not the same as Pick® printer numbers. SETPTR enables you to define logical printer units, which may be, but are not necessarily, linked to specific printers. UniData printer unit numbers are used with the PRINT ON statement in UniBasic to allow multiple concurrent jobs. Use the DEST option of SP.ASSIGN to specify Pick® printers and forms.

The following table describes modes for SETPTR.

Mode	Description
1	Directs output to a printer only. Default mode.
2	Must be used with DEVICE option. Directs output to the serial device specified by the DEVICE option.
3	Directs output to a _HOLD_ file only.
6	Directs output to both a _HOLD_ file and a printer.
9	Directs output to a printer. Suppresses display of the _HOLD_ entry name.

SETPTR Modes

The next table describes options for the SETPTR command.

Option	Description
BANNER [*string*]	Modifies the default banner line (which is the Windows user id). Depends on MODE setting; also modifies _HOLD_ entry name.
BANNER UNIQUE (*string*)	Modifies the default banner line and automatically uses attribute 1 (NEXT.HOLD) in the dictionary for the _HOLD_ file to create unique entry names for jobs sent to _HOLD_.
BRIEF	Suppresses the verification prompt.
COPIES *n*	Prints *n* copies. Does not work with mode 3. Default is 1.
DEFER [*time*]	Delays printing until the specified *time*. Specify the time in HH:MM format. Does not work with mode 3.
[DEST | AT] *unit*	Directs output to a specific printer or queue. The *unit* may be either a local printer or a network printer.
DEVICE *name*	Used with mode 2 only. Directs output to the Windows device (for instance, a COM port) identified by *name*.
EJECT	Ejects a blank page at the end of the print job.
NOEJECT	Suppresses the form feed at the end of the print job.
LNUM	Prints line numbers in the left margin.
NFMT | NOFMT	Suspends all UniData print formatting.
NHEAD | NOHEAD	Suppresses the banner.
OPEN	Opens a print file, and directs output to this file until the file is closed by the SP.CLOSE command.

SETPTR Options

The next table describes spooler options you can specify in a quoted string.

Option	Description
Orientation	The paper orientation. Must be PORTRAIT or LANDSCAPE. Defaults to the setting in the Default Document Properties sheet for the printer.
PaperSource	The default paper source; must match an available paper source listed on the **Device Settings** tab of the printer's Properties Sheet.
Duplex	Must be NONE, HORIZONTAL, or VERTICAL; default is NONE. **Note**: If the print device does not support duplex printing, this option is ignored. Jobs print singlesided and no error message displays.
Form	The form to use (for instance, Letter). Must match an available paper size listed on the Device Settings tab of the printer's Properties Sheet.
Mode	RAW or WINDOW. Default is RAW, meaning that printer-specific escape sequences are required for all formatting. **Note**: Specifying formatting options (Form, Font, FontSize, Orientation, FontStyle, DefaultSource, or Duplex) in a quoted string automatically switches Mode to WINDOW.
Prefix	The printer-specific escape sequence, specified as a decimal (not ASCII) value. Valid in RAW mode only.
Font	The font name, for instance, "Courier New." **Note**: The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows platforms attempt to find an appropriate font to use from the ones installed on your computer.
FontSize	The font size in points (for instance, 8, 9, 10, 11). **Note**: The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows platforms attempt to find an appropriate font to use from the ones installed on your computer.

SETPTR Spooler Options

Option	Description
FontStyle	Must be Regular, Italic, Bold, Underline, or StrikeOut. Default is Regular. **Note**: The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows NT and Windows 2000 attempt to find an appropriate font to use from the ones installed on your computer.
LeftMargin	The left margin of the page, in inches.
RightMargin	The right margin of the page, in inches.
TopMargin	The top margin of the page, in inches. **Note**: TopMargin is measured beginning at the value of the SETPTR *topmargin* option (default is 3 lines). If *topmargin* is 3 lines (the default) and TopMargin = 1, the first printed line is one inch below the third line of the page.
BottomMargin	Bottom margin of the page, in inches. **Note**: BottomMargin is measured beginning at the value of the SETPTR *bottommargin* option (default is 3 lines). If *bottommargin* is 3 lines (the default) and BottomMargin = 1, the first printed line is one inch above the third line from the end of the page.
Priority	Must be from 1 to 99, where 1 is minimum priority and 99 is maximum priority.
JobState	The only valid value is PAUSE, which stops all jobs to the print unit. There is no way to reverse this action.

SETPTR Spooler Options (continued)

Examples (UniData for Windows Platforms)

To display information about printers on your Windows system, double-click **My Computer**, and then select **Printers**. Otherwise, if you prefer, from the **Start** menu, select **Settings**, and then click **Printers**. In the following example, there are three local printers and one network print device defined. The local printers may point to the same physical print device or to different physical print devices. The **Printers** dialog box appears:

RECAPTURE EXAMPLE

Tip: You can print from UniData to any network print device available to you. A print device does not need to be visible in the Printers dialog box.

You can define local or network printers to UniData by using the SETPTR command, as shown in the following examples.

```
:SETPTR 0,,,,,1,AT
LETTER,"TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12"
Unit 0
Mode 1
Options are:
Destination LETTER
Lp options : TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12
OK to set parameters as displayed?(enter y/n) y

:SETPTR 0
Unit 0
Width 105
Length 31
Top margin 3
Bot margin 3
Mode 1
Options are:
Destination LETTER
Lp options : TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12

:SETPTR 1,,,0,0,1,AT \\DENVER4\hpzone3,"Priority=99"
Unit 1
Top margin 0
Bot margin 0
Mode 1
Options are:
Destination \\DENVER4\hpzone3
Lp options : Priority=99
OK to set parameters as displayed?(enter y/n) y

:SETPTR 2,,,,,1,AT LEGAL
Unit 2
Mode 1
Options are:
Destination LEGAL
OK to set parameters as displayed?(enter y/n) Y

:SETPTR 3,,,,,1,AT \\DENVER4\hpzone2,"Form=A4"
Unit 3
Mode 1
Options are:
Destination \\DENVER4\hpzone2
Lp options : Form=A4
OK to set parameters as displayed?(enter y/n) y
:
```

Notice the following points:

- The default print device (printer unit 0) is now mapped to the local printer LETTER. If you use the PRINT command or LPTR with no print unit specified, your print job is directed to LETTER.
- Use SETPTR unit to display the current settings for a print unit.
- When you specify spooler options (TopMargin, BottomMargin), UniData automatically recalculates the width and length, taking these into account. Also, when you specify formatting options in a quoted string, UniData implicitly changes the spooler Mode from RAW (the default) to WINDOW.
- You can specify spooler options in a quoted string either before or after SETPTR options like AT, DEFER.
- You can map a printer unit to a network print device even if that device is not displayed in your Printers dialog.

After you have defined printers with SETPTR, you can display a list with the LISTPTR command, as shown below:

```
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0     LETTER                \\DENVER4\hpzone3      ‘Running
1     \\DENVER4\hpzone3     hpzone3                 Running
2     LEGAL                 \\DENVER4\hpzone3       Running
3     \\DENVER4\hpzone2     hpzone2                 Running
```

Notice that, in the previous example, the two local printers point to the same network print device.

Use PTRDISABLE and PTRENABLE (STOPPTR and STARTPTR) to control the local printers:

```
:PTRDISABLE LETTER
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0     LETTER                \\DENVER4\hpzone3      Paused
1     \\DENVER4\hpzone3     hpzone3                Running
2     LEGAL                 \\DENVER4\hpzone3      Running
3     \\DENVER4\hpzone2     hpzone2                Running
:PTRENABLE LETTER
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0     LETTER                \\DENVER4\hpzone3      Running
1     \\DENVER4\hpzone3     hpzone3                Running
2     LEGAL                 \\DENVER4\hpzone3      Running
3     \\DENVER4\hpzone2     hpzone2                Running
:
```

Only users with Full Control permissions on a printer can control the printer with PTRDISABLE and PTRENABLE. Check **Permissions** on the **Security** tab of the printers **Properties** sheet to determine who has permissions.

Notice that the argument for PTRDISABLE and PTRENABLE is the name of the local printer (as specified with DEST or AT in SETPTR).

Tip: In the examples in this chapter, the local printers point to a network print device. PTRDISABLE and PTRENABLE pause or resume the local printer only. They do not affect the underlying network print device, and they do not affect other local printers that point to the print device.

You can use the ECL SP.STATUS command to display information about printers defined with SETPTR and print jobs started from your UniData session.

The following example shows SP.STATUS output:

```
:SP.STATUS
Device for LETTER: \\DENVER4\hpzone3
LETTER is Running.
Device for \\DENVER4\hpzone3: hpzone3
\\DENVER4\hpzone3 is Running.
Device for LEGAL: \\DENVER4\hpzone3
LEGAL is Running.
Device for \\DENVER4\hpzone2: hpzone2
\\DENVER4\hpzone2 is Running.

JobId.... User............ Size.... Status... Unit..
Printer..................
241 terric 543 Defered 3 \\DENVER4\hpzone2 \
:
```

The status of all the printers is Running, and the network print device has a deferred job.

Depending on how a print device was configured, users in console sessions may see printer notification messages when a job completes. The following example shows such a message:

RECAPTURE EXAMPLE

Note: The Printing Notification displays only if you are logged in to a console session. If you are logged in to UniData via TELNET, you will not see the notification.

Redefining the Default UniData Print Unit

To keep UniBasic applications general, developers typically use (or assume) printer unit 0, which is the default. You can redefine unit 0 to direct output from different parts of an application to different physical printers or queues or to change formatting options with the SETPTR command.

The following example is a very simple paragraph that redefines the default print unit for different reports:

```
:CT VOC OUTPUT
VOC:
OUTPUT:
PA
SETPTR 0,80,78,3,3,1,AT LEGAL
RUN BP REPORT_PRINT
SETPTR 0,80,60,3,3,1,AT LETTER
RUN BP LETTER_PRINT
:
```

Submitting Concurrent Print Jobs

With SETPTR, you can define up to 31 logical printer units per UniData session. You can use this functionality to submit concurrent print jobs from a UniBasic application. One common implementation follows:

- Define two logical printer units (for instance, 0 and 1) that point to different physical print devices.
- Direct all lines of a report to one printer with the UniBasic PRINT ON command (for instance, PRINT ON 0 PRINT.LINE).
- Direct summary (break) lines to the second printer (PRINT ON 0 PRINT.LINE followed by PRINT ON 1 PRINT.LINE).

In this way, you can print a summary report and a detail report at the same time.

SETTAPE

Syntax

SETTAPE *unit* [*no_rewind_driver*][*rewind_driver*][*block*]

Description

The ECL **SETTAPE** command initializes a pointer to a tape unit for use by the current process. You must initialize a tape unit with the SETTAPE command before you can access it. If you include unit without any other parameters, UniData displays the current settings for that tape unit.

On UniData for Windows NT, the SETTAPE command establishes a link between a UniData internal tape unit number and an NTFS tape device. You can use SETTAPE to relate unit number to tape devices, or to NTFS or FAT disk files.

Note: If you are using an NTFS tape drive on a Windows platform, you must identify the tape drive with its name in UNC format. If you are using a disk file, you may identify it by its path and file name. The disk file must already exist.

SETTAPE creates an editable ASCII file located in *udthome*/sys/tapeinfo on UniData for UNIX and *udthome*\sys\tapeinfo on UniData for Windows Platforms. If you attach a tape and change the block size from that specified in tapeinfo, UniData creates another file in the same directory, tapeatt, which takes precedence over tapeinfo.

Note: To initialize or update a pointer to a tape unit, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
unit	Number, 0–9, indicating the tape unit to be initialized. *unit* without any other parameters displays the current settings for *unit*.
no_rewind_driver	Path and device name of the "no rewind" device driver for *unit*. On UniData for Windows Platforms, the driver must be specified in the UNC format if the device is a tape drive.
rewind_driver	Path and name of the "rewind" device driver for *unit*. On UniData for Windows Platforms, the driver must be specified in the UNC format if the device is a tape drive.
block	Block size in bytes. Must be a multiple of 512. If you do not stipulate a block size, UniData uses 4096.

SETTAPE Parameters

Example (UniData for UNIX)

The following example displays the settings for tape unit 1:

```
:SETTAPE 1
unit # = 1.
non rewind device:/dev/rmt/0mn
rewind device :/dev/rmt/0m
block size =4096
:
```

The next example initializes a pointer to UNIX disk file:

```
:SETTAPE 4 /tmp/diskfile1 /tmp/diskfile1 16384
unit # = 4.
non rewind device:/tmp/diskfile1
rewind device :/tmp/diskfile1
block size =16384
:
```

Example (UniData for Windows Platforms)

In the following example, UniData displays the settings for tape unit 1:

```
:SETTAPE 1
unit # = 1.
non rewind device:\\.\tape0
rewind device :r\\.\tape0
block size =4096
:
```

In the next example, UniData establishes a tape unit that is actually a NTFS disk file:

```
:SETTAPE 0 \\.\tape0 R\\.\tape0 4096
:
```

Related Commands

T.ATT, T.DET

shmconf

Syntax

shmconf

Description

The system-level **shmconf** command runs the interactive shmconf (shared memory configuration) utility, which sets UniData shared memory configuration parameters. shmconf is supported on UniData for UNIX only.

For detailed information about shared memory configuration, see *Administering UniData.*

Note: Use this command at the operating system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Tip: Use the udtconf command to set all UniData configuration parameters.

Example

The following example shows the shmconf display:

```
% shmconf
AutoConf ChecKConf SaVeConf CaLcCTL SysParam Exit

Users Licensed: 32 Platform: 0000479670 OS: AIX 2 3
NUSERS.....: 64 SHM_LPINENTS....: 10 MIN_MEMORY_TEMP.: 256
SHM_GNTBLS.: 16 SHM_LMINENTS....: 8 COMPACTOR_POLICY: 1
SHM_GNPAGES: 32 SHM_LCINENTS....: 100 VARMEM_PCT......: 50
SHM_GPAGESZ: 1024 SHM_LPAGESZ.....: 8
SHM_FREEPCT: 25 AVG_TUPLE_LEN...: 4
SHM_NFREES.: 1 EXPBLKSIZE......: 64
SHMMAX......: 268435456 SHM_ATT_ADD.: 1073741824
SHMMIN......: 1 SHM_LBA.....: 268435456
PressCtrl +{A |K |V |L |P |E}toperform a command.
Press PF1 to get help information about a field.
```

showconf

Syntax

showconf [-o | -O] *filename*][-h | -H]

Description

The system-level command **showconf** displays current settings for UniData configuration parameters. These values may differ from the settings listed in utdconfig, for example, if a value specified in udtconfig is inadequate, UniData recalculates it.

Note: showconf is supported on UniData for UNIX only.

Execute this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
[-o | -O] *filename*	Directs output to *filename.*
-h | -H	Displays the command usage. If you use this with the other options, UniData recognizes only the -h option.

showconf Parameters

Example

The following sample output illustrates the three lists of parameters:

- Section 1 Neutral parameters
- Section 2 Non-RFS related parameters

- Section 3 RFS related parameters

```
:!showconf
## Unidata Configuration Parameters
#
# Section 1 Neutral parameters
# These parameters are required by all Unidata installations.
#
# 1.1 System dependent parameters, they should not be changed.
LOCKFIFO=1
SYS_PV=3
# 1.2 Changable parameters
NFILES=60
NUSERS=20
WRITE_TO_CONSOLE=0
TMP=/tmp/
NVLMARK=
FCNTL_ON=0
TOGGLE_NAP_TIME=161
NULL_FLAG=0
N_FILESYS=200
N_GLM_GLOBAL_BUCKET=101
N_GLM_SELF_BUCKET=23
GLM_MEM_ALLOC=10
GLM_MEM_SEGSZ=4194304
# 1.3 I18N related parameter
UDT_LANGGRP=255/192/129
#
# Section 2 Non-RFS related parameters

#
# 2.1 Shared memory related parameters
SBCS_SHM_SIZE=1048576
SHM_MAX_SIZE=67108864
SHM_ATT_ADD=0
SHM_LBA=4096
SHM_MIN_NATT=4
SHM_GNTBLS=40
SHM_GNPAGES=32
SHM_GPAGESZ=256
SHM_LPINENTS=10
SHM_LMINENTS=32
SHM_LCINENTS=100
SHM_LPAGESZ=8
SHM_FREEPCT=25
SHM_NFREES=1
# 2.2 Size limitation parameters
AVG_TUPLE_LEN=4
EXPBLKSIZE=16
MAX_OBJ_SIZE=307200
MIN_MEMORY_TEMP=64
# 2.3 Dynamic file related parameters
GRP_FREE_BLK=5
SHM_FIL_CNT=2048
SPLIT_LOAD=60
```

```
MERGE_LOAD=40
KEYDATA_SPLIT_LOAD=95
KEYDATA_MERGE_LOAD=40
MAX_FLENGTH=1073741824
PART_TBL=/disk1/ud50/parttbl
# 2.4 NFA server related parameter
EFS_LCKTIME=0
# 2.5 Journal related parameters
JRNL_MAX_PROCS=1
JRNL_MAX_FILES=400
# 2.6 UniBasic file related parameters
MAX_OPEN_FILE=500
MAX_OPEN_SEQF=150
MAX_OPEN_OSF=100

MAX_DSFILES=1000
#2.7 UniBasic related parameters
MAX_CAPT_LEVEL=2
MAX_RETN_LEVEL=2
COMPACTOR_POLICY=1
VARMEM_PCT=50
# 2.8 Number of semaphores per semaphore set
NSEM_PSET=8
# 2.9 Index related parameters
SETINDEX_BUFFER_KEYS=0
SETINDEX_VALIDATE_KEY=0
# 2.10 UPL/MGLM parameter
MGLM_BUCKET_SIZE=50
#
# Section 3 RFS related parameters
# These parameters are only used for RFS which is turned by
# setting SB_FLAG to a positive value.
#
# 3.1 RFS flag
SB_FLAG=1
# 3.2 File related parameters
BPF_NFILES=80
N_PARTFILE=500
# 3.3 AFT related parameters
N_AFT=200
N_AFT_SECTION=1
N_AFT_BUCKET=101
N_AFT_MLF_BUCKET=23
N_TMAFT_BUCKET=19
# 3.4 Archive related parameters
ARCH_FLAG=1
N_ARCH=2
ARCHIVE_TO_TAPE=0
ARCH_WRITE_SZ=0
# 3.5 System buffer parameters

N_BIG=233
N_PUT=8192
# 3.6 TM message queue related parameters
```

```
N_PGQ=10
N_TMQ=10
# 3.7 After/before image related parameters
N_AIMG=2
N_BIMG=2
AIMG_BUFSZ=102400
BIMG_BUFSZ=102400
AIMG_MIN_BLKS=10
BIMG_MIN_BLKS=10
AIMG_FLUSH_BLKS=2
BIMG_FLUSH_BLKS=2
# 3.8 Flushing interval related parameters
CHKPNT_TIME=300
GRPCMT_TIME=5
# 3.9 Sync Daemon related parameters
N_SYNC=0
SYNC_TIME=0
#
# Section 6 Century Pivot Date
#
CENTURY_PIVOT=1930
LOG_OVRFLO=/liz1/ud52/log/log_overflow_dir
```

SG.LIST

Syntax

SG.LIST [item] [FROM [*list.number*]]

Description

The **SG.LIST** command executes the SAVE.LIST command immediately followed by a GET.LIST command.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
item	The name of the savedlist you want to create.
FROM *list.number*	An active list number between 0 and 9. If you do not specify *list.number*, UniData assumes 0.

SG.LIST Parameters

showud

Syntax

showud

Description

The system-level **showud** command lists all active UniData daemons.

Note: showud is supported on UniData for UNIX only.

For more information about showud and recoverable files, see *Administering the Recoverable File System.*

Note: Use this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Examples

The following displays UniData background processes (daemons) that are running for a UniData installation with RFS disabled (udtconfig parameter SB_FLAG = 0):

```
# $UDTBIN/showud
UID PID TIME COMMAND
root 3527 0:00 /disk1/ud60/bin/cleanupd -m 10 -t 20
root 3525 0:00 /disk1/ud60/bin/sbcs -r
root 3520 0:00 /disk1/ud60/bin/smm -t 60
```

smmtest

Syntax

smmtest

Description

The system-level **smmtest** command tests the UNIX and UniData configuration values. This process takes 10 to 20 seconds to complete.

Note: smmtest is supported on UniData for UNIX only.

Use this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Example

The following example shows a smmtest display:

```
:!smmtest
Testing udt configuration values ...
End of parameter checking!
*** NUSERS (40)*3 mustbe<=SEMMNU (100)
End of IPC checking!
==> Please do the following:
(a) Adjust your Unix kernel parameters and reconfigure the kernel

(b) Modify your '/usr/ud60/include/udtconfig' file if necessary
#
```

If you are not logged on as root when you execute this command, you may get messages related to permissions, as in the next example:

```
% smmtest
Open /dev/kmem error: Permission denied
Testing udt configuration values ...
End of parameter checking!
Open /dev/kmem error: Permission denied
*** SHM_LNTBLS (50) * 3 must be <= SEMMNU (0)
End of IPC checking!
==> Please do the following:
(a) Adjust your Unix kernel parameters and reconfigure the kernel
(b) Modify your '/usr/ud60/include/udtconfig' file if necessary
:
```

smmtrace

Syntax

smmtrace [-d | -e]

Description

The system-level **smmtrace** command enables or disables tracing of shared memory management. If tracing is enabled (-e parameter), and the system is running smoothly, UniData writes messages to the smm.errlog file at the shared memory managers (smm) checking intervals. When tracing is disabled (-d parameter), UniData sends messages to smm.errlog only when a shared memory problems arises. If you do not include an option, UniData displays usage.

The smm checking interval is platform-dependent.

Note: To execute the smmtrace command, you must log on as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Use this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL (colon) prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-d	Disables tracing of shared memory management.
-e	Enables tracing of shared memory management.

smmtrace Parameters

Example

When you execute this command, UniData does not display a response. The following example displays the contents of smm.errlog by changing to the udtbin directory and executing the UNIX more command.

```
% cd $UDTBIN
% more smm.errlog
SMM trace: Checking IDs of the IPC facilities...
SMM trace: Checking process groups...
SMM trace: Fixing GCTs...
SMM trace: Checking memory utilization...
SMM trace: Receiving messages...
SMM trace: Interrupted.
...
```

sms

Syntax

sms [-h | -g [*gct*] | -G[*shmid*] | -l[*lct*] | -L[*pid*] | -S*shmid* | -d] [-f]

Description

The system-level **sms** command displays the contents of shared memory segments or of global or local control tables.

For information about local control tables, global control tables, and managing shared memory, see *Administering UniData*.

Note: Use this command at the system prompt, or use the ECL ! (bang) command to execute it from the ECL prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-h	Displays configuration values for global and local control tables and shared memory segments. On UniData for UNIX, UniData retrieves the values from the /usr/52/include/udtconfig file. On UniData for Windows Platforms, UniData retrieves the information from the *udthome*\include\udtconfig file. UniData also displays the interprocess communication facility identifiers when smm starts.
-g [*gct*]	Default. Displays global control table use. Each global control table controls a shared memory segment. Each shared memory segment is divided into equally-sized global pages. UniData displays the number of global control tables in use and marks them with a shared memory identifier. It also displays free global control tables and marks these with -l. If you indicate a global control table number (*gct*). UniData displays the contents of the global control table.
-G [*shmid*]	Displays global control table use. If you stipulate a shared memory identifier (*shmid*) with this parameter, UniData displays page information for a specific global control table.
-l [*lct*]	Default. Displays local control table use or the contents of a local control table. Each local control table controls a shared memory segment, and each shared memory segment is divided into equal-sized local pages. UniData displays the number of local control tables in use and marks them with a shared memory identifier. It also displays free local control tables and marks these with -l. If you indicate a local control table number (*lct*), UniData displays the contents of the local control table.

sms Parameters

Parameter	Description
-L [*pid*]	Displays local control table use. If you stipulate a process ID (*pid*) with this parameter, UniData displays additional information for a specific table in the following subtables: ■ Process Info ■ Counter ■ Memory Info ■ Control Info **Tip**: To learn the process ID, enter **sms -l**. The process ID is the first number in the leftmost column of the display. **Note**: UniData does not display unused entries under Memory Info and Control Info.
-S *shmid*	Displays local control table use of a shared memory identifier created by a user (*shmid*). UniData displays additional information for a specific table in the following subtables: ■ Process Info ■ Counter ■ Memory Info ■ Control Info **Tip**: To display the shared memory identifiers, use the **ipcstat** command. **Note**: UniData does not display unused entries under Memory Info and Control Info.
-d	Displays values for open UniData dynamic files system-wide, including: ■ Device number ■ I-node number ■ Flag – a scan flag. If set to 1, splitting and merging is blocked. ■ Modulo – current modulo ■ Counter – users who have the file open
-f	Displays if a file system is NFS.

sms Parameters (continued)

Example

The following example shows an sms display that results from the -h parameter:

```
% sms -h
Shmid of CTL: 22202
------------------------------- IDs ----------------------------
----
smm_pid smm_trace PtoM_msgqid MtoP_msgqid ct_semid (values)
232 1 6900 6701 5696 (1,1,1)
-------------------- GENERAL INFO ---------------------

SHM_GNTBLS = 40 (max 40 global segments / system)
SHM_GNPAGES = 32 (32 global pages / global segment)
SHM_GPAGESZ = 256 (128K bytes / global page)
NUSERS = 40 (max 40 process groups / system)
SHM_LPINENTS = 10 (max 10 processes / group)
SHM_LMINENTS = 32 (max 32 global pages / group)
SHM_LCINENTS = 100 (max 100 control entries / group)
SHM_LPAGESZ = 8 (4K bytes / local page)
SHM_FREEPCT = 25
SHM_NFREES = 1
SHM_FIL_CNT = 2048
JRNL_BUFSZ = 53248
N_FILESYS = 200
%
```

SORT

Syntax

SORT(*var*)

Description

The **SORT** function enables users to sort a dynamic array. *var* is the name of the dynamic array.

The elements in the dynamic array are sorted in ascending order, left-justified. The dynamic array is sorted by the highest system delimiter in the array.

- If the dynamic array contains any attribute marks, the sort is by attribute,. Values and sub-values remain with the original attribute.
- If the dynamic array contains value marks and no attribute marks, the sort is by value. Subvalues are unaffected and remain with the original value.
- If the dynamic array contains subvalue marks and neither attribute marks nor value marks, the sort is by subvalue.

SORT.TYPE

Syntax

SORT.TYPE [*option*]

Synonym

SORT-TYPE

Description

The ECL **SORT.TYPE** command sets the sort type used throughout UniData for the current session.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
0	Default. Attributes specified as right-justified in the dictionary are sorted in numeric order. Nonnumeric data is sorted as 0.
1	Sort order is determined by ASCII value.
2	Numeric characters are sorted before nonnumeric characters. Nonnumeric characters and symbols are sorted by ASCII value.

SORT.TYPE Parameters

Examples

Note: Before executing the following examples, the demo database file ORDERS was modified by the addition of data in the CLIENT_NO attribute to better illustrate the various sort types.

The following example demonstrates SORT.TYPE 0 sort type 0 sorts characters and symbols as if they were 0. Notice that default sort type, 0, is displayed when the user enters the command without an option.

```
:SORT.TYPE
SORT.TYPE 0
:SORT ORDERS CLIENT_NO BY CLIENT_NO
SORT ORDERS CLIENT_NO BY CLIENT_NO 09:52:47 Jun 15 1999 1
Client
ORDERS.... Number....
ABC -10
000 000
100000 !
817 A
820 [
823 #
825 a
831 r
836 K
855 {
888 K
889 :
901 <
954 &
BC BC
CDE CDE
001 001
002 003
003 003
...

822 10026
826 10043
816 10045
824 10060
202 records listed
```

The following example demonstrates SORT.TYPE 1, which sorts all data by ASCII value:

```
:SORT.TYPE 1
:SORT ORDERS CLIENT_NO BY CLIENT_NO
SORT ORDERS CLIENT_NO BY CLIENT_NO 09:53:00 Jun 15 1999 1
Client
ORDERS.... Number....
100000 !
823 #
954 &
ABC -10
000 000
001 001
002 003
003 003
862 9965
844 9966
...
824 10060
889 :
901 <
817 A
BC BC
CDE CDE
836 K
888 K
820 [
825 a
831 r
855 {
202 records listed
```

This example demonstrates SORT.TYPE 2, which sorts numbers before characters and symbols; numbers and symbols are then sorted by ASCII value.

```
:SORT.TYPE 2
:SORT ORDERS CLIENT_NO BY CLIENT_NO
SORT ORDERS CLIENT_NO BY CLIENT_NO 09:53:17 Jun 15 1999 1
Client
ORDERS.... Number....
ABC -10
000 000
001 001
002 003
003 003
862 9965
844 9966
...
816 10045
824 10060
100000 !
823 #
954 &
889 :
901 <
817 A
BC BC
CDE CDE
836 K
888 K
820 [
825 a
831 r
855 {
202 records listed
```

SP.ASSIGN

Syntax

SP.ASSIGN [B] [C*n*][F[*unit* | *form*] | [R*unit*]] [H] [HS] [O] [I*print_job*]

Synonym

SP-ASSIGN

Description

The ECL **SP.ASSIGN** command assigns print job output. This command provides Pick ® -like syntax to achieve SETPTR operations. If you enter this command without any options, UniData does not print a verification prompt upon execution (equivalent to SETPTR 0,,,,,BRIEF).

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
B	Equivalent to SETPTR,,,,,BRIEF
C*n*	Print *n* (number of) copies.
F[*unit* | *form*]	Equivalent to SETPTR,,,,,[UNIT | FORM]
R*unit*	Resets the options. Equivalent to SETPTR unit,,,,, *unit*–Printer unit number, from 0 through 255. (The default is zero).
H	Sends the output to the _HOLD_ file and the printer. Equivalent to SETPTR,,,,,6

SP.ASSIGN Parameters

Parameter	Description
HS	Sends output to the _HOLD_ file. Equivalent to SETPTR,,,,,3
O	Opens a print file and sends printer output to it until SP.CLOSE is executed. Equivalent to SETPTR,,,,,OPEN.
I*print_job*	Disregards the queueing order and size of the job and moves it to the head of the print queue. *print_job* is the identifier associated with a print job. You must have adequate permissions to use this option.

SP.ASSIGN Parameters (continued)

Examples

In the following example, taken from UniData for Windows NT, SP.ASSIGN maps the default print unit to a network print device:

```
:SP.ASSIGN F\\DENVER4\hpzone3
:SETPTR 0
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 1
Options are:
Destination \\DENVER4\hpzone3
:
```

In the next example, SP.ASSIGN opens a print file:

```
:SP.ASSIGN O
:SETPTR 0
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 3
Options are:
Destination \\DENVER4\hpzone3
OPEN
:
```

In the following example, SP.ASSIGN resets all SETPTR options to their default values:

```
:SP.ASSIGN R
:SETPTR 0
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 1
Options are:
:
```

Notice that in each example SETPTR 0 displayed the current settings.

SP.CLOSE

Syntax

SP.CLOSE [*unit*]

Synonym

SP-CLOSE

Description

The ECL **SP.CLOSE** command closes an open print process for *unit*.

This command executes the final step to complete a single print process that results from one or more print commands. The process begins with SETPTR...OPEN, continues with print commands, and finishes with a SP.CLOSE command.

Example

The following example opens a print process that prints records from the CLIENTS, INVENTORY, and ORDERS demo files to a _HOLD_ file and then closes the print process. The NOHEAD option suppresses the printing of a header.

```
:SETPTR 0,,,,,3,OPEN
Unit 0
Mode 3
Options are:
OPEN
OK to set parameters as displayed?(enter Y/N) y
:LIST CLIENTS LNAME WITH LNAME LIKE “P...” LPTR
:LIST INVENTORY PROD_NAME WITH COLOR = “Gold” LPTR
:LIST ORDERS GRAND_TOTAL WITH GRAND_TOTAL > 10000 LPTR

:SP.CLOSE
:
```

Use the LS command to check the contents of the _HOLD_ file. Then, use the SPOOL command to display the output of the print job to the terminal, as in the next example:

```
:LS _HOLD_
P_0000
:SPOOL _HOLD_ P_0000 -T
...
#### ## ##### #### # # #
######### ##
# # ## ## ## # #
# ###### ##### # # # # ## #
######### ####
#### # # # # #### ###### # #
Fri Jun 8 17:08:01 MDT 1999
...
LIST CLIENTS LNAME WITH LNAME LIKE “P...” LPTR 17:08:38 Jun 08
1999 1
CLIENTS... Last Name......
10035 Primm
10016 Pooley
9965 Phillips
10039 Primm
10005 Pappas
10084 Pilano
10047 Parker
7 records listed
...
LIST INVENTORY PROD_NAME WITH COLOR = “Gold” LPTR 17:09:12 Jun 08
1999 1
Product
INVENTORY. Name......
No records listed.
...
LIST ORDERS GRAND_TOTAL WITH GRAND_TOTAL > 10000 LPTR 17:09:25 Jun
08 1999 1
ORDERS.... Grand Total...
941 $13,999.90
805 $47,555.29
834 $825,159.96
802 $16,983.24
833 $69,057.73
...

35 records listed
...
```

SP.EDIT

Syntax

SP.EDIT [*record*]

Synonym

SP-EDIT

Description

The ECL **SP.EDIT** command starts a system editor from which you can display, edit, or print a record in the _HOLD_ file. If you do not enter a record name, UniData prompts for it.

After you enter SP.EDIT and a record ID, UniData prompts for an action code. After each action except quit, UniData returns to the action code prompt (?). If you do not indicate filename, UniData prompts for each file in the _HOLD_ file in sequence, starting with the earliest entry first.

Action Codes

The following table lists the SP.EDIT action codes.

Code	Name	Description
T | t	terminal	Displays file on terminal.
F | f	find	Prompts for a search string. After you enter the string, UniData displays the file, beginning with the line containing the string, and then returns to the ? prompt. **Note**: Do not enclose the string in quotation marks.
S | s	spool	Spools the file to the printer.
D | d	delete	Deletes the file
Q | q	quit	Returns to the ECL colon prompt (:).

SP.EDIT Action Codes

Example

In the following example, UniData opens a record in the _HOLD_ file and then prompts for an action code. The user responds by entering t for terminal display, and UniData displays the first page of the record:

```
:SP.EDIT P_0000
Hold item P_0000 - (t) terminal (f) find (s) spool (d) delete or
(Q) quit ?
t
...
#### ## ##### #### # # #
######### ##
# # ## ## ## # #
# ###### ##### # # # # ## #
######### ####
#### # # # # #### ###### # #
Fri Jun 8 17:08:01 MDT 1999
...
LIST CLIENTS LNAME WITH LNAME LIKE “P...” LPTR 17:08:38 Jun 08
1999 1
CLIENTS... Last Name......
10035 Primm
10016 Pooley
Enter h for help, <CR> for next page
```

SP.KILL

Syntax

SP.KILL *job*

Synonym

SP-KILL

Description

The ECL **SP.KILL** command stops a UniData print job. When you use the LPTR keyword to print a job from within UniData, the job number displays on your terminal.

If your operating system directs print jobs to printers linked to another machine, this command may not cancel the print job.

Example

In the following example, taken from UniData for Windows NT, SETPTR displays the characteristics of the default print unit. A query is spooled to the default printer, then SP.KILL removes the print job:

```
:SETPTR
Unit 0
Width 80
Length 56
Top margin 3
Bot margin 3
Mode 1
Options are:
Defer 19:00
Destination \\DENVER4\hpzone3
Lp options : Form=LETTER

:LIST VOC LPTR
request id is 225
:SP.KILL 225
SP-KILL of Job ‘225’ succeeded.
:
```

SP-LISTQ

SP-LISTQ is a synonym for the LISTPEQS command. For more information, see LISTPEQS.

Synonym

LISTPEQS

SP.STATUS

Syntax

SP.STATUS

Synonym

SP-STATUS

Description

The ECL **SP.STATUS** command displays the current status of all printers.

Example

The following example shows an SP.STATUS display on UniData for UNIX:

```
:SP.STATUS
scheduler is running
system default destination: hpzone3
device for hpzone4: /dev/null
device for hpzone3: /dev/null
device for parallel: /dev/c1t0d0_lp
hpzone4 accepting requests since Dec 10 10:21
hpzone3 accepting requests since Dec 10 10:22
parallel accepting requests since Apr 1 14:12
printer hpzone4 is idle. enabled since Dec 10 10:21
fence priority : 0
printer hpzone3 is idle. enabled since Dec 10 10:22
fence priority : 0
printer parallel is idle. enabled since Apr 1 14:12
fence priority : 0
no entries
(EOF)Enter h for help, <CR> for next page
```

The next example shows an SP.STATUS display on UniData for Windows Platforms:

```
:SP.STATUS
Device for \\DENVER4\hpzone3: hpzone3
\\DENVER4\hpzone3 is Running.
JobId.... User............ Size.... Status... Unit..
Printer..................
230 terric 10143 Defered 0 \\DENVER4\hpzone3
Device for LETTER: \\DENVER4\hpzone3
LETTER is Running.
:
```

SPOOL

Syntax

SPOOL *filename record* [*recordM...recordN*][-O][-T]

Description

The ECL **SPOOL** command prints the contents of a record or records.

Even though SETPTR mode may be set to 3 or 6 (route to _HOLD_ file), SPOOL directs output only to the print queue or terminal.

Tip: The SPOOL command is useful for printing text files, such as _PH_ and _HOLD_ records and for printing UniBasic programs.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The UniData file to be printed.
record	The record ID in *filename*. You can list more than one record by separating the record IDs with a space.
-O	Suppresses display of the file name and the record ID in the output.
-T	Displays output to the terminal rather than the printer.

SPOOL Parameters

Examples

In the following example, UniData displays the contents of three records from the ORDERS demo file to the terminal:

```
:SPOOL ORDERS 801 912 941 -T
ORDERS:
801:
10133
59640
...
10009
50000
Gray
10
139999
:
```

The following example displays the UniBasic program TEST, which is stored in the BP directory file:

```
:SPOOL BP TEST -T
BP:
TEST
PRINT 'HELLO THERE'
:
```

SQL

Syntax

SQL

Description

The ECL **SQL** command invokes UniData SQL, the UniData ANSI Structured Query Language.

For more information about using UniData's structured query language, see *Using UniData SQL*.

Tip: You can open a UniData SQL session and execute a UniData SQL statement on the same command line from the UniData colon prompt, as in :SQL SELECT GRAND_TOTAL FROM ORDERS; You can also execute a script file containing UniData SQL commands in the same manner, as in

:SQL filename

Example

The following example initiates UniData SQL:

```
:SQL
sql>
```

STACKCOMMON

Syntax

STACKCOMMON [ON | OFF]

Description

The ECL **STACKCOMMON** command controls whether UniBasic programs share unnamed common when one program uses the EXECUTE, PERFORM, or MDPERFORM command to call a second program.

If you enter STACKCOMMON without any parameters, UniData displays the setting: ON or OFF.

STACKCOMMON has no effect on common areas when the UniBasic CALL command is used to call programs.

For more information about assigning variables in UniBasic, see *Developing UniBasic Applications* or see the COMMON command in the *UniBasic Commands Reference*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
ON	Unnamed common is not shared with executed programs. The unnamed common of the second program is initialized to 0. When control is passed back to the first program, unnamed common is restored to settings for that program. Unnamed common is never passed to a phantom program.
OFF	Unnamed common is shared with programs called with the ECL EXECUTE command.

STACKCOMMON Parameters

Example

The following example displays the programs, test.common and executed.program:

```
PROGRAM test.common
COMMON A,B,C,D
A =1
B =2
C =3
D =4
PRINT “In test.common, A,B,C,D = “:A:B:C:D
EXECUTE “RUN BP executed.pgm”
PRINT “Back in test.common, A,B,C,D = “:A:B:C:D
END


PROGRAM executed.pgm
COMMON A,B,C,D
PRINT “In executed.pgm. A,B,C,D = “:A:B:C:D
RETURN
```

In the following test run, we set STACKCOMMON OFF before executing test.common, causing variables in unnamed common to be passed to the executed program. Finally, we set STACKCOMMON ON, so that common variables are no longer passed.

```
:STACKCOMMON OFF
:RUN BP test.common
In test.common, A,B,C,D = 1234
In executed.pgm. A,B,C,D = 1234
Back in test.common, A,B,C,D = 1234
:STACKCOMMON ON
:RUN BP test.common

In test.common, A,B,C,D = 1234
In executed.pgm. A,B,C,D = 0000
Back in test.common, A,B,C,D = 1234
```

STARTPTR

STARTPTR is a synonym for the PTRENABLE command. For information, see PTRENABLE.

Synonym

PTRENABLE

startud

Syntax

startud [-i] [-m]

Description

The system-level **startud** command starts the UniData background processes (smm, sbcs, and cleanupd). If the SB_FLAG is set to 1, UniData also starts the Recoverable File System (RFS) daemons. This command ensures that the UniData daemons start up in the correct sequence.

For information about startud and starting the UniData background processes, see *Administering UniData*. For more information about startud with the RFS, see *Administering the Recoverable File System*.

Note: To execute the startud command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-i	Bypasses the automated crash recovery sequence on systems running RFS. **Warning**: To avoid file corruption on systems running RFS, IBM recommends that you avoid using the -i parameter unless directed to do so by IBM Technical Support.
-m	Executed the ECL command mediarec to restore archived changes made since the last backup. See the mediarec command in this manual for more information about that command, and Administering the Recoverable File System for information about the recovery process.

startud Parameters

Example

In the following example, UniData starts the UniData daemons with RFS turned on (SB_FLAG = 1).

```
# $UDTBIN/startud
Using UDTBIN=/disk1/ud60/bin
All output and error logs have been saved
to /disk1/ud60/bin/saved_logs directory.
SMM is started.
SBCS is started.
CLEANUPD is started.
SM is started.
Unirpcd is started
UniData R6.0 has been started.
#
```

The next example, taken from UniData for Windows NT, starts the UniData services:

```
D:\Informix\ud60\Bin>startud
Wait for Unidata Service to be started ...
The Unidata Service has been started successfully.
```

STATUS

Syntax

STATUS

Synonym

STAT

Description

The ECL **STATUS** command lists information about users logged onto the system. For each user, UniData displays user ID, tty device ID, and date and time of login. UniData also displays a list of the file systems and disk space information.

On UniData for UNIX, the STATUS command display is equivalent to the combined display of the WHO and AVAIL commands.

Example

The following example shows a STATUS display on UniData for UNIX:

```
:STATUS
carolw pts/1 Jun 6 07:55
carolw pts/4 Jun 6 08:29
amyc pts/5 Jun 6 08:59
amyc pts/6 Jun 6 09:20
/disk1 (/dev/vg01/lvol2 ): 313910 blocks 349051 i-nodes
/home (/dev/vg01/lvol1 ): 1667012 blocks 304276 i-nodes
/opt (/dev/vg00/lvol5 ): 54340 blocks 27470 i-nodes
/tmp (/dev/vg00/lvol6 ): 79036 blocks 28995 i-nodes
/usr (/dev/vg00/lvol7 ): 86260 blocks 54787 i-nodes
/var (/dev/vg00/lvol8 ): 117722 blocks 68583 i-nodes
/stand (/dev/vg00/lvol1 ): 58230 blocks 7659 i-nodes
/ (/dev/vg00/lvol3 ): 147210 blocks 14863 i-nodes
```

The next examples shows a STATUS display on UniData for Windows NT:

```
:STATUS
terric pts/1 14:06:27 Jun 30 1999 (192.245.120.102)
Drive Free Bytes / Total Bytes
C: 188530688/649576448
D: 669504000/1496968704
:
```

Related Commands

AVAIL, WHO

STOPPTR

STOPPTR is a synonym for the PTRDISABLE command. For more information, see PTRDISABLE.

Synonym

PTRDISABLE

stopud

Syntax

stopud [-f]

Description

The system-level **stopud** command stops all UniData background processes. The -f option forces UniData daemons to stop unconditionally, which kills all active udt processes. For information about stopud with recoverable files, see *Administering the Recoverable File System*.

Note: To execute the stopud command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

Warning: Use this command with the -f option only as a last resort. It could cause file corruption.

Examples

In the next example, taken from UniData on UNIX, UniData stops all UniData daemons. In this example, the Recoverable File System is ON:

```
# $UDTBIN/stopud -f
Using UDTBIN=/disk1/ud60/bin
The Last archive file (/disk1/archive/ud500) is LSN -- 0
SM stopped successfully.
CLEANUPD stopped successfully.
SBCS stopped successfully.
SMM stopped successfully.
Unirpcd stopped successfully
Unidata R6.0 has been shut down.
#
```

The next example, taken from UniData for Windows NT, stops all UniData services:

```
D:\Informix\ud60\Bin>stopud
Stop Unidata Service now ...
```

The Unidata Service has been stopped successfully.

stopudt

Syntax

stopudt *pid* [*pidM...pidN*]

Description

The system-level **stopudt** command stops one or more UniData processes. This command sends a signal to the process requesting that the process terminate in an orderly manner.

pid represents the process identification number for the process or processes you intend to halt.

Tip: Use the ECL LISTUSER command or the system-level listuser command to display a list of users and their processes.

Example

The following example demonstrates using LISTUSER to list all users on the system, then execute stopudt against user 6372. The final LISTUSER display demonstrates that this user has been eliminated from UniData:

```
:LISTUSER
Licensed/Effective # of Users Udt Sql Total
32 /32 2 0 2
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
1 15885 0 root udt pts/1 14:01:57 Jun 05 2000
2 15903 1172 claireg udt pts/2 14:02:28 Jun 05 2000
:!$UDTBIN/stopudt 15903
:LISTUSER

Licensed/Effective # of Users Udt Sql Total
32 /32 1 0 1
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
1 15885 0 root udt pts/1 14:01:57 Jun 05 2000
```

Related Commands

deleteuser, LISTUSER

SUPERCLEAR.LOCKS

Syntax

SUPERCLEAR.LOCKS *pid* [*locknum*]

Synonym

SUPERCLEAR-LOCKS

Description

The ECL **SUPERCLEAR.LOCKS** command deletes semaphore locks set by the user executing the command. This command can be executed from a different process or terminal than the one from which the locks were set. You can clear only the semaphore locks set by your process ID.

For information on UniData locks, see *Developing UniBasic Applications* or *Administering UniData.*

Note: If you are logged in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms, you can execute SUPERCLEAR.LOCKS to clear semaphore locks set by other users.

The LIST.LOCKS command displays the locks that are currently active. The GETUSER command lists your user number.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
pid	Specified the process ID of the user that set the lock.
locknum	Specifies the number of the lock to be released. If you do not specify *locknum*, UniData releases all locks set by pid.

SUPERCLEAR.LOCKS Parameters

Example

In the following example, SUPERCLEAR.LOCKS deletes locks set by user carolw (user ID 1283) from UniData session 2253:

```
:LIST.LOCKS
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
1 2253 1283carolw ts/1 semaphor -1 0 1 X 10:44:29 Jun 31
6 2365 1283carolw ts/6 semaphor -1 0 2 X 10:44:29 Jun 31
:SUPERCLEAR.LOCKS 2253
:LIST.LOCKS
UNO UNBR UID UNAME TTY FILENAME INBR DNBR RECORD_ID M TIME DATE
6 2365 1283carolw ts/6 semaphor -1 0 2 X 10:44:29 Jun 31
:
```

Related Command

SUPERRELEASE

SUPERRELEASE

Syntax

SUPERRELEASE *pid* [*inbr devnum* | *record_ID*]

Description

The ECL **SUPERRELEASE** command clears exclusive file and record locks set by the user executing the command. This command can be executed from a different process than the one in which the locks were set.

Tip: Use the GETUSER command to list user number, user name, and user ID. Use the LIST.READU command to display record locks that are active.

Parameters

The following table describes each parameter of the syntax.

Example

In the following example, the SUPERRELEASE command releases the record lock set by user number 14435 on the file with an i-node number of 1121 and a device number 45:

```
:SUPERRELEASE 14435 1121 45
```

Related Command

SUPERCLEAR.LOCKS

sysmon

Syntax

sysmon [-b | -m] [-o *outputfile*][-t*nnn*]

Description

The system-level **sysmon** utility monitors the performance of the Recoverable File System.

This information may help you make decisions about how to set UniData configuration parameters. To learn more about sysmon and the Recoverable File System, see *Administering the Recoverable File System*.

Note: You can use the ECL ! (bang) command to execute this command from the ECL (colon) prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-b	Displays detailed information about the Block Index table (BIG) in shared memory. You cannot use -m with -b.
-m	Displays detailed information about user requests. You cannot use the -b option with the -m option.
-o *outputfile*	Directs sysmon output to *outputfile*.
-t *nnn*	Samples the data at intervals of every *nnn* seconds.

sysmon Parameters

Example

The following example shows a sysmon display:

```
% sysmon
======= BLOCK INDEX GROUP (BIG) STATISTICS ======= Thu Jun 25
18:59:08 1999
PinRead :1579 TmRead :121 Dirty:0 Hits :1539
PinWrite :67 TmWrite:16 Neat :80 HitRate:93.50%
PinWaitQ :0 CmRead :0 Total:80
PinWaitRate:0.00% CmWrite:8
=============== LATCHING STATISTICS =============== === TM STATUS
===
Type----WaitQ---Latches-WaitRate-PollCall-PollRate Tm# :2 Req#:204
Big : 0 12720 0.00% 0 0.00% ActTm:2
Aft : 0 248 0.00% 0 0.00%
Aimg: 0 252 0.00% 0 0.00% === SHM INFO ====
Bimg: 0 162 0.00% 0 0.00% ShmPV:197 Total:197
========================== LOG FILE STATISTICS
==========================
TmBimgFlush:29 WaitQ0:58 LogCkSuccess:8089 BimgRawBlks:41
TmAimgFlush:29 WaitQ1:0 LogCkFail :58 AimgRawBlks:29
CmBimgFlush:10 WaitQ2:0 LogOvrflos :0 TotRaw :70
CmAimgFlush:10 WaitQ3:102 LogSwitchd :5
LogID-Total-Length
4 1 1 ========== RECORD INFO ========== === TRANS INFO ===
5 1 1 RecRead : 865 AvgRead : 61 Committed: 76
6 1 1 RecWrite : 0 AvgWrite: 0 Aborted : 0
7 1 1 RecDelete: 0
TotLength:4
```

systest

Syntax

systest [-m*n*][-s*n*][-u][-f*filename*][-v][c {n | r}]

Description

The system-level **systest** command, available only on UniData for UNIX, checks all parameters in the udtconfig file located in /usr/ud60/include. For more information about setting UniData configuration parameters, as well as systest, see *Administering UniData*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
[-m*n*]	Changes memory map display by about *n* MB. Highly platform dependent. Do not use this unless advised by IBM.
[-s*n*]	Changes memory map display by about *n* MB. Highly platform dependent. Do not use this unless advised by IBM.
[-u]	Creates UniData configuration parameters if they do not already exist in the udtconfig file.
[-f *filename*]	Creates a file with the specified *filename* that contains the UniData configuration parameters that systest would calculate if you specified the -u parameter.
[-v]	Displays detailed (verbose) output.
[-c {n | r}]	Checks current kernel parameter settings against our recommendations. Specify the -cr parameters to compare against recommendations for the Recoverable File System. Specify the -cn parameters if you will not be using recoverable files.

systest Command Parameters

Note: Prior to Release 4.1, the systest -u command may have updated values that already existed in the udtconfig file. Beginning with Release 4.1, existing values are no longer updated, but parameters that do not exist in the udtconfig file are added by systest -u. To change existing values to recommended values, IBM recommends using the udtconf command.

Examples

This example demonstrates executing systest -f (followed by the UNIX diff command) to find out what changes systest -u would make to udtconfig:

```
# ./systest -f /tmp/testconfi
...
#diff/tmp/testconfig /usr/ud60/include/udtconfig
33c33
<SHM_MAX_SIZE=16777216

...
```

Notice that diff output includes lines like

33c33

which shows the edit command necessary for correcting differences. In this example, systest would have changed the value of SHM_MAX_SIZE. This is the type of correction to udtconfig you would expect if you change the shmmax kernel parameter after installing UniData or since you last ran systest.

systest -f does not update LOCKFIFO, PART_TBL, or WRITE_TO_CONSOLE in output.If they were present in your udtconfig file (and they usually are after installation) they always show up in diff output.

You can use this information to decide how you want to change the live udtconfig file. Remember, you need to stop and start UniData for the changes to take effect.

systest -f updates NFILES, so this is also a great, noninvasive way to check NFILES when that setting is suspect.

T.ATT

Syntax

T.ATT [*cn*] [BLKSIZE *block*] [TAPELEN *length*]

Synonym

T- ATT

Description

The ECL **T.ATT** command attaches a tape drive for exclusive use by the current process. Before you can use any tape commands, the tape unit must be defined. See SETTAPE for information about initializing a tape unit.

Tip: If you have trouble reading tapes from non-UniData systems, try varying the block size.

Parameters

The following table lists the T.ATT parameters.

Parameter	Description
nn	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.
BLKSIZE *block*	Indicates block size. *block* is a valid block size. If you do not stipulate BLKSIZE, UniData uses the block size set by the SETTAPE command.
TAPELEN *length*	Indicates a tape length for multi-reel tape processing. *length* is the desired tape length in megabytes. **Note**: TAPELEN applies only to tapes created in UniData. UniData cannot read multi-reel TDUMP tapes made on legacy systems.

T.ATT Parameters

Example

In the following example, UniData attaches tape unit 4 without indicating a block size. (For the block size, UniData uses the block size set by the SETTAPE command.)

```
:T.ATT 4
tape unit 4 blocksize = 16384.
:T.STATUS
UNIT STATUS UDTNO USER CHANNEL ASSIGNED
NUMBER NAME NAME BLOCKSIZE
1 AVAILABLE
2 AVAILABLE
3 AVAILABLE
5 AVAILABLE
8 AVAILABLE
4 ASSIGNED 3 root /tmp/diskfile1 16384
:
```

Related Commands

SETTAPE, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ,T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.BAK

Syntax

T.BAK [*n* | MU[*cn*]

Synonym

T-BAK

Description

The ECL **T.BAK** command moves the pointer to a tape backward *n* files.

Before you can execute any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
n	The number of files to move the pointer back.
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.

T.BAK Parameters

Related Commands

SETTAPE, T.ATT, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL,T.READ,T.REW,T.SPACE, T.STATUS, T.UNLOAD,T.WEOF

T.CHK

Syntax

T.CHK [*cn*]

Synonyms

T-CHK, T.CHECK

Description

The ECL **T.CHK** command reads the contents of a tape that was produced with the T.DUMP command and checks for tape errors such as physical damage and block size corruption.

The first digit of *nn* represents the conversion code number. The second digit is the unit number. If you do not indicate *nn*, UniData uses 00. UniData allows up to 10 unit numbers, from 0 through 9.

Note: Before you can execute any tape commands, the tape system must be configured. See SETTAPE for information about initializing a tape.

Parameters

The following table lists the T.CHK parameters.

Parameter	Description
c	Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion ■ 2 – Invert high-bit ■ 3 – Swap bytes **Note**: Do not separate the conversion code from the tape unit with a space.
n	Tape unit number. UniData allows up to 10 unit numbers, from 0 through 9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). **Note**: Do not separate the conversion code from the tape unit with a space.

T.CHK Parameters

Related Commands

SETTAPE, T.ATT, T.BAK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ, T.REW, T.SPACE,T.STATUS, T.UNLOAD, T.WEOF

T.DET

Syntax

T.DET *[n]*

Synonym

T- DET

Description

The ECL **T.DET** command releases a tape unit that was attached with the T.ATT command. *n* is the tape unit number. UniData allows up to 10 unit numbers, from 0 through 9.

Before you can use any tape commands, the tape system must be configured. See SETTAPE for information about initializing a tape.

Example

In the following example, UniData releases tape unit 8:

```
:T.DET 8
```

Related Commands

SETTAPE, T.ATT, T.BAK,T.CHK, T.DUMP,T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ, T.REW,T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.DUMP

Syntax

T.DUMP [DICT] *filename* [MU *cn*][*record*[*recordM...recordN*] | *select_criteria*] [[PICK | pick] [HDR.SUP]]

Synonym

T-DUMP

Description

The ECL **T.DUMP** command copies the contents of a file to a tape that was attached with the T.ATT command. UniData writes an end-of-file mark at the end of the file.

T.DUMP works with an active select list. If you wish to copy a sorted subset of records, create a select list before using T.DUMP. For information about creating select lists, refer to *Using UniQuery*. If a record ID is included in a saved list that does not exist in the file, UniData displays a message that the record was not found and not copied.

Before you can execute any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

Note: UDT.OPTIONS 50 allows you to choose the ASCII characters used as the end-of-record mark. When this option is on, UniData uses character 251, a UniData text mark. When this option is off, UniData uses character 254, an attribute mark, followed by the text mark. This feature provides compatibility with Pick® on Ultimate systems.

Tip: Due to the differences in Pick ® operating systems and manufactured tapes, IBM suggests that you use the HDR.SUPP keyword when using the T.DUMP command, and when using the Pick® T-LOAD command to avoid inconsistencies in tape labels.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Indicates the dictionary portion of the file. If you do not stipulate DICT, UniData copies only the data portion of the file.
filename	The UniData file to be copied.
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.
record	The records within *filename* to copy.
select_criteria	The record IDs, a select list of the record IDs, or a selection condition. If you do not indicate *select_criteria*, UniData copies all records within *filename*.
PICK | pick	Produces a tape that can be loaded on a Pick® system. To avoid incompatibility in tape label format, suppress the label by including HDR.SUP.
HDR.SUP	Suppresses the generation of a tape label.

T.DUMP Parameters

Examples

The following example copies all records from the ORDERS demo file to default tape unit 0 with no conversion:

```
:T.DUMP ORDERS
193 record(s) dumped to tape
```

In the following example, UniData sends the contents of the ORDERS demo file to tape unit 0 with no conversion. Since only one number is indicated on the command line, UniData uses that number for the conversion code and uses 0 for the tape unit:

```
:T.DUMP ORDERS MU 1
193 record(s) dumped to tape
```

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK, T.DET, T.EOD, T.FWD, T.LOAD, T.RDLBL,T.READ, T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.EOD

Syntax

T.EOD [*n*]

Synonym

T-EOD

Description

The ECL **T.EOD** command moves the file pointer for tape unit *n* to the end of the file. UniData allows up to 10 tape unit numbers, from 0 through 9.

Before you can execute any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.FWD, T.LOAD, T.RDLBL, T.READ,T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.FWD

Syntax

T.FWD *n*

Synonym

T- FWD

Description

The ECL **T.FWD** command moves the file pointer for tape unit *n* to the beginning of the next file.

Before you can use any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

T.LOAD

Syntax

T.LOAD [DICT] *filename* [MU *cn*][*select_criteria*]] [OVERWRITING] [TAPELEN *length*] [PICK | pick]

Synonym

T-LOAD

Description

The ECL **T.LOAD** command loads to *filename* records that were stored on tape using the T.DUMP command. UniData cannot read files from tapes that were created using a tape command other than T.DUMP.

UniData can read Pick ® system tapes that were created with T.DUMP and the PICK (or pick) option without tape labels. To avoid incompatibility between systems with different tape label formats, suppress the tape label when performing the T.DUMP operation.

The tape unit must have been attached using T.ATT before being loaded with the T.LOAD command.

Before you can use any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

Note: UDT.OPTIONS 50 selects ASCII characters that UniData can use as the end-of-record mark. When this option is on, UniData uses character 251, the UniData text mark. When this option is off, UniData uses character 254, the attribute mark, followed by the text mark. This feature provides compatibility with Pick ® on Ultimate systems.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Dictionary records will be loaded.
filename	The target disk file. *filename* must exist in the UniData account.
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.
select_criteria	The record IDs, a select list of the record IDs, or a selection condition. If you do not indicate *select_criteria*, UniData copies all records within *filename*. **Tip**: For information about creating select lists, see *Using UniQuery*.
OVERWRITING	Overwrites records that already exist in the target file.
TAPELEN *length*	Indicates the tape length for multi-reel tape processing. *length* represents the desired tape length in megabytes for multi-reel tape processing.
PICK | pick	Removes special end-of-block characters from tapes to expedite the conversion process to UniData.

T.LOAD Parameters

Example

The following example loads records stored on tape unit 0 to file ORDERS_LOAD. UniData loads only the records that meet the selection criteria ORD_DATE < 01/01/96.

```
:T.LOAD ORDERS_LOAD WITH ORD_DATE < 01/01/96
56 records loaded to ORDERS_LOAD
:
```

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.RDLBL, T.READ, T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.RDLBL

Syntax

T.RDLBL [MU *cn*]

Synonym

T-RDLBL

Description

The ECL **T.RDLBL** command reads the tape label (the first 80 characters) of a file that was saved to tape by the T.DUMP command. The label displays on the terminal.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.

T.RDLBL Parameters

Example

The following is a T.RDLBL display:

```
:T.RDLBL
L4000 16:01:40 14 Jun 1999 ORDERS
```

Related Commands

SETTAPE, T.ATT, T.BAK,T.CHK, T.DET,T.DUMP, T.EOD, T.FWD, T.LOAD, T.READ, T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.READ

Syntax

T.READ [*-code*][*cn*]

Synonym

T-READ

Description

The ECL **T.READ** command reads the next record from tape and displays it on the display. The tape unit must have been attached using T.ATT. To quit processing the tape, enter q at the prompt;T.READ reads a tape to end-of-file.

Before you can use any tape commands, the tape system must be configured. For information about initializing a tape, see SETTAPE.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-code	On UniData for UNIX, enables you to specify special options associated with the UNIX "od" command. These options control the format and display the records retrieved. Refer to your host operating system manual for an explanation of the operating system commands and their options.
cn	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.

T.READ Parameters

Example

The following example shows a T.READ display:

```
:T.READ 04
0000000 377 L 4 0 0 0 1 6 :01 :4 0
0000020 1 4 J u n 1 9 9 6 O R DE
0000040 R S
0000060
0000100 0 0
0000120 9 1 2376 1 0 24 037645 0 0 0376
0000140 9 9 84376 5 30 0 0376N /A3766
0000160 376 1 2 9 9 5 376 373 8 0 1376 1 0 13
0000200 3 376 5 9 6 4 0376 1 0 01 837611
0000220 0 0 0376 G r ay37613761 7 9 90
0000240 0 376 373 9 4 1 376 1 0 2 41376 5 40
0000260 0 0 376 1 0 0 09376 5 00 0 0376G
0000300 r a y376 1 03761 3 9 99 93763738

0000320 0 5 376 1 0 1 40376 4 02 6 03769
...
```

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK,T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL,T.REW, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.REW

Syntax

T.REW [*n*]

Synonym

T-REW

Description

The ECL **T.REW** command rewinds a tape unit to the beginning of the tape. *n* is the tape unit number. UniData allows up to 10 unit numbers, from 0 through 9.

Before you can use any tape commands, the tape system must be configured. See SETTAPE for information about initializing a tape.

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL,T.READ, T.SPACE, T.STATUS, T.UNLOAD, T.WEOF

T.SPACE

Syntax

T.SPACE [*n* | MU *cn*]

Synonym

T-SPACE

Description

The ECL **T.SPACE** command moves the file pointer *n* files forward on the tape.

Before you can use any tape commands, the tape unit must be configured. See SETTAPE for information about initializing a tape.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
n	The number of files to move the file pointer.
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ■ 0 – Default. No conversion. ASCII is assumed. ■ 1 – EBCDIC conversion. ■ 2 – Invert high-bit. ■ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.

T.SPACE Parameters

Example

In the following example, UniData moves forward two files on a tape:

```
:T.SPACE 2
2 FILE ARE SKIPPED
:
```

Related Commands

SETTAPE, T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ, T.REW, T.STATUS, T.UNLOAD, T.WEOF

T.STATUS

Syntax

T.STATUS [*n*]

Synonym

T- STATUS

Description

The ECL **T.STATUS** command displays the current tape device assignment. n is the tape unit number. UniData allows up to 10 unit numbers, from 0 through 9. If you do not include a tape unit number, UniData displays assignments for all tape units defined by SETTAPE.

T.STATUS displays the contents of the file *udthome*/sys/tapeinfo on UniData for UNIX, or *udthome*\sys\tapeinfo on UniData for Windows Platforms.

Before you can use any tape commands, the tape system must be configured. See SETTAPE for information about initializing a tape.

Example

The following example shows a T.STATUS display:

```
:T.STATUS
UNIT STATUS UDTNO USER CHANNEL ASSIGNED
NUMBER NAME NAME BLOCKSIZE
1 AVAILABLE
2 AVAILABLE
3 AVAILABLE
5 AVAILABLE
8 AVAILABLE
4 ASSIGNED 1 terric /tmp/diskfile1 4096
:
```

Related Commands

SETTAPE,T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ,T.REW, T.SPACE, T.UNLOAD, T.WEOF

T.UNLOAD

Syntax

T.UNLOAD [*n*]

Synonym

T-UNLOAD

Description

The ECL **T.UNLOAD** command rewinds and unloads a tape. *n* is the tape unit number. UniData allows up to 10 unit numbers, from 0 through 9.

Before you can use any tape commands, the tape system must be configured. See SETTAPE for information about initializing a tape.

Related Commands

SETTAPE,T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD, T.RDLBL, T.READ, T.REW, T.STATUS, T.STATUS, T.WEOF

T.WEOF

Syntax

T.WEOF [*n*]

Synonym

T-WEOF

Description

The ECL **T.WEOF** command writes an end-of-file mark on the tape. *n* is the tape unit number. UniData allows up to 10 unit numbers, from 0 through 9.

Before you can use any tape commands, the tape system must be configured. For information about initializing a tape, see SETTAPE.

Related Commands

SETTAPE,T.ATT, T.BAK, T.CHK, T.DET, T.DUMP, T.EOD, T.FWD, T.LOAD,T.RDLBL,T.READ, T.REW, T.SPACE, T.STATUS, T.UNLOAD

tandem

Syntax

tandem [-o*xxx*] *udtno*

Description

The UniData system-level **tandem** command displays or controls the input and output displayed on another user's terminal from your terminal.

Note: The tandem command is supported on UniData for UNIX only.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-o*xxx*	Use to specify three single characters to terminate tandem. If you specify the -o option, UniData disables ESC+X as the terminating character sequence.
udtno	The udtno for the user for which you want to display or control input and output.

tandem Parameters

tandem Modes

The tandem command has the following three modes:

- Feed – Enables you to enter commands for another user on your terminal. If you enter data at the same time as the other user, your keystrokes are defined by the system implementation of the terminal driver.

- Message – Enables you to send text to another user's terminal. You cannot control the location or format of the characters displayed on the users terminal. Data is not treated as input for the other user.
- View – Default mode. Shows another user's input and output on your terminal. This display continues when you use message or feed mode.

You can change modes by entering any of the following escape sequences (ESC + an action code).

Escape Sequence	Description
ESC+D	Puts tandem in view mode. Terminates message or feed mode if active. Sends a BREAK to the other user's process.
ESC+F	Puts tandem in feed mode. Terminates message mode, if active. You must log in as root to use feed mode.
ESC+M	Puts tandem in message mode. Terminates feed mode, if active.
Q	In view mode, same as ESC+X. No effect in other modes.
ESC+U	Same as ESC+D.
ESC+V	Puts tandem in view mode. Terminates message or feed mode, if active.
ESC+X	Ends the tandem session. If you did not specify -o *c*1 *c*2 *c*3 on the command line.
ESC+ESC	In message or feed mode, enables you to send ESC to another user's terminal.
ESC+?	Displays information on your terminal, and also displays the status information for the current session.

tandem Action Codes

Note: The COMO command does not capture output produced by tandem. When you use tandem on a phantom process, you cannot use the feed or message mode.

The feed mode for a tandem session is not supported on all UNIX platforms. If this mode is not supported on your operating system, UniData displays a message.

TERM

Syntax

TERM [[*type*] | [A|,B|,C |,D]]

Description

The ECL **TERM** changes the settings for your terminal and printer for the current UniData session. If you do not indicate type or any of the options (A through D), UniData displays the current settings for the terminal, excluding terminal type.

Use a comma as a placeholder for options you are not changing.

Terminal setup is part of the system setup process. IBM recommends that you use .login or .profile files to store terminal settings. For more information about system setup, see *Administering UniData*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
type	The terminal type.
A	The number of characters per line for the display terminal (default: 80).
,B	The number of lines per page for the display terminal (default: 23). TERM ,0 disables pagination, so that UniData does not pause at the end of each display page.
,C	The number of characters per line for the printer (from 1 through 256).
,D	The number of lines per page for the printer (default: 60). 0 disables pagination.

TERM Parameters

Examples

In the following example, UniData displays the current settings for the display terminal and the printer:

```
:TERM
TERM parameters are all numeric:
TERM A,B,C,D
For the terminal
A=number of characters in a line(80).
B=number of lines per page(23).
For the line printer
C=number of characters in a line(132).
D=number of lines per page(60).
:
```

The next example changes all settings for the terminal and one setting for the printer. Notice the comma that acts as a placeholder for option C, which does not change:

```
:TERM 25,10,,40
:TERM
TERM parameters are all numeric:
TERM A,B,C,D
For the terminal
A=number of characters in a line(25).
B=number of lines per page(10).
For the line printer
C=number of characters in a line(132).
D=number of lines per page(40).
:
```

The following example changes the terminal type:

```
:TERM vt100
:
```

TIMEOUT

Syntax

TIMEOUT *nn*

Description

The ECL **TIMEOUT** command automatically logs a user out of a UniData session if input is not received in *nn* seconds. The setting remains in effect for the current UniData session only.

TIMEOUT applies to the following:

- ECL (colon) prompt.
- Proc IP and IBP commands.
- Paragraph inline prompting (except with the I option).
- UniBasic INPUT commands and the IN function.

TIMEOUT does not apply to the prompt that displays after an interrupt in ECL.

UniData executes the LOGOUT paragraph before exiting the session.

Warning: Depending on your application coding, setting TIMEOUT could cause logical database inconsistencies. For example, without transaction processing in effect, an application might update part of a record, prompt for user input, and then time out at the prompt without updating the rest of the record.

Example

In the following example, the TIMEOUT command logs the user off after 59 seconds if UniData receives no input:

```
:TIMEOUT 59
Process will timeout after waiting 59 seconds for input.
:
```

trunclog

Syntax

trunclog[-minline *n* | -minsize *n*] [-verbose] [logfilename...]

Description

The **trunclog** command appends the contents of a UniData log file you specify to its corresponding saved file in the $UDTBIN / saved_logs directory while UniData is running. UniData then truncates the log file to a size of zero, and writes a message similar to the following example in the truncated log file:

```
The file was truncated : Thu Jul 25 11:30:47
```

If you do not specify a file name to truncate, trunclog truncates the following files:

- cleanupd.errlog
- sbcs.errlog
- sm.log (UNIX only)
- smm.errlog
- udt.log (Windows platforms only)
- udtlatch.log

You must be root on UniData for UNIX or Administrator on UniData for Windows Platforms to execute this command, and you must set the UDTBIN environment variable.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-minline *n*	Truncates only the log files which contain at least *n* lines.
-minsize *n*	Truncates only the log files which contain at least *n* bytes.
-verbose	Prints the handling messages.
logfilename	Name of the log file to truncate.

trunclog Parameters

If you do not specify -minline *n* or -minsize *n*, UniData truncates all of the log files you specify.

Example

The following command truncates all log files:

```
# trunclog
```

The following command truncates all log files with a minimum size, in bytes, of 1K:

```
# trunclog -minsize 1024
```

The next command truncates the sm.log file:

```
# trunclog sm.log
```

Warning: Because UniData does not set an exclusive lock while copying and truncating a log file, it is possible that one or more messages, for example, those generated by another UniData daemon, may be lost.

udfile

Syntax

udfile [-r | -s] *filename*

Description

The system-level udfile command converts a UniData file to or from recoverable. If you enter this command without options, UniData displays the type of type (recoverable or nonrecoverable).

Warning: You cannot convert files with this command while UniData is running.

Note: You must have root permissions to change the recoverability type of a file.

Execute this command at the system prompt.

The udfile command will not convert files that were created in 1/2-K blocks. If you attempt to do so, UniData generates an error message indicating that the file cannot be converted to recoverable. You must resize the file to at least a 1-K block size using the ECL RESIZE command or the UniData system-level memresize command. Or, you can create a new file with at least a 1K block size, then copy the contents of the old file into the new one using the ECL COPY command.

For details about converting files to recoverable with udfile, see *Administering the Recoverable File System*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	Name of the UniData file to convert.
-r	Converts a nonrecoverable file to recoverable.
-s	Converts a recoverable file to nonrecoverable.

Examples

The following example displays the fact that the CLIENTS demo file is nonrecoverable:

```
% udfile CLIENTS
File ‘CLIENTS’ is non-recoverable dynamic file.
%
```

The next example changes the nonrecoverable CLIENTS demo file to recoverable:

```
% udfile -r CLIENTS
Non-recoverable file ‘CLIENTS’ changed to recoverable file.
%
```

The following example changes the recoverable ORDERS demo file to nonrecoverable:

```
% $UDTBIN/udfile -s $UDTHOME/demo/ORDERS
Recoverable file ‘/disk1/ud60/demo/ORDERS’ is changed to non-
recoverable file
```

udipcrm

Syntax

udipcrm

Description

The system-level **udipcrm** command removes all interprocess communication (IPC) structures associated with UniData. Execute this command at the system prompt.

If you are running multiple versions of UniData (for example, 5.2 and 6.0), udipcrm removes only the structures associated with the version from which you execute udipcrm.

Note: This command is supported on UniData for UNIX only.

Warning: Running udipcrm stops all UniData background processes and halts all UniData user processes.

udstat

Syntax

udstat [-b] [-l *num* | -L*pid*][*interval* [*count*]]

Description

The system-level **udstat** command displays details about process groups and the sbcs daemon.

Note: This command is supported on UniData for UNIX only.

Use this command at the UNIX prompt, or use the ECL ! (bang) command to execute this command from the ECL (colon) prompt.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-b	Displays benchmark values. Used by IBM only for diagnostic purposes.
-l *num*	Displays a process group using the local control table number (*num*) of the group you want to sample.
-L *pid*	Displays the process group using the group process identification number *pid*.
interval [*count*]	Invokes udstat with a time and count interval. The default interval is 5 seconds. For example, udstat 10 3 displays current statistics every 10 seconds three times, then stops.

udstat Parameters

The following table describes the column headings that display in the output for the udstat command.

Column Heading	Description
pr	Private p_code requests.
gr	Global p_code requests.
po	Overflow pages accessed.
iv	id overflow accessed.
gl	Locks requested in a process group.
gu	Unlocks requested in a process group.
fs	Times of floating to string.
op	Virtual files opened.
rd	Times of reads.
wt	Times of writes.
dl	Times of deletes.
sh	Times of shell command.
sr	Times of subr.
sc	Stings flushed (screen io).
lc	Total locks requested.
ul	Total unlocks requested.
ph	Total physical reads.
vr	Total virtual requests.

udstat Display

When you specify the -b parameter, the following headings are displayed in place of po, iv, gl, gu, fs, op.

Old Heading	-b Heading	Description
po	cr	Times of carriage return input
iv	tm1	setmark 1
gl	tm2	setmark 2
gu	tm3	setmark 3
fs	tm4	setmark 4
op	tm5	setmark 5

Changed Column Headings with -b Option

Examples

The following example shows a udstat display with the -b option:

```
% udstat -b
all process group lcp sbcs

pr gr cr tm1 tm2 tm3 tm4 tm5 rd wt dl sh sr sc lc ul ph vr
6 42 217 0 2k 82 3 12 0 2k 704 721 754 0
%
```

udt

Syntax

udt [*program_name* | RUN *program_name* | *ECL_command*]

Description

The system-level **udt** command starts a UniData session. For UniData to run, the product must be installed and licensed, and the following environment variables must be set correctly:

- UDTBIN must be set to your UniData bin directory
- UDTHOME must be set to your UniData home directory

Note: You must use the udtts command to start a UniData session if you are using device licensing.

Execute this command at the system prompt.

Consult your system administrator for information about setting up your UniData environment.

For a full description of the UniData environment variables, see *Administering UniData.*

You can start a UniData session, execute a UniBasic program or ECL command, then automatically exit the UniData session by entering the program name or ECL command after the udt command from the system-level prompt. In these cases, @USER.TYPE returns 2.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
program_name	Starts a UniData session, executes a cataloged program, then automatically exits the UniData session. Enter the command from the system-level prompt.
RUN *program_name*	Starts a UniData session, executes a noncataloged program, then automatically exits the UniData session. Enter the command from the system-level prompt.
ECL_command	Starts a UniData session, executes an ECL command, then automatically exits the UniData session. Enter the command from the system-level prompt.

udt Parameters

Examples

The following example shows how to start a UniData session:

```
% udt
UniData Release 6.0 Build: (4112)
(c) Copyright IBM Corporation 2002.
All rights reserved.
Current UniData home is /liz1/ud60/.
Current working directory is /home/claireg.
:
```

In the following example, the user attempted to start a UniData session when the UniData daemons had not been started. To correct this problem, you must first start UniData with the startud command.

```
# $UDTBIN/udt
Start SMM first!
```

The next example illustrates how to start a UniData session, execute an ECL command, then automatically exit the UniData session from the system-level.

```
% udt LIST VOC SAMPLE 5
UniData Release 6.0 Build: (4112)
(c) Copyright IBM Corporation 2002.
All rights reserved.
Current UniData home is /liz1/ud60/.
Current working directory is /home/claireg.

:LIST VOC SAMPLE 5
LIST VOC SAMPLE 5 09:48:30 Jun 21 2002 1
VOC.......
LIST.LABEL
IN
NEWPCODE
NO.NULLS
SETLINE
5 records listed
:%
```

Related Command

BYE

udtbreakon

Syntax

udtbreakon *pid*

Description

The system-level **udtbreakon** command enables the interrupt key from another port. With this capability, users can enter the UniBasic debugger to terminate a program that may be stuck in a loop. *pid* represents the udt process id on another port for which you enable the interrupt key.

Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL (colon) prompt.

Tip: Use the LISTUSER command to find the process ID for which you intend to enable the interrupt key. The process ID for the UniData session is shown in the USRNBR column.

Related Commands

ON.BREAK, PTERM -BREAK ON

udtconf

Syntax

udtconf

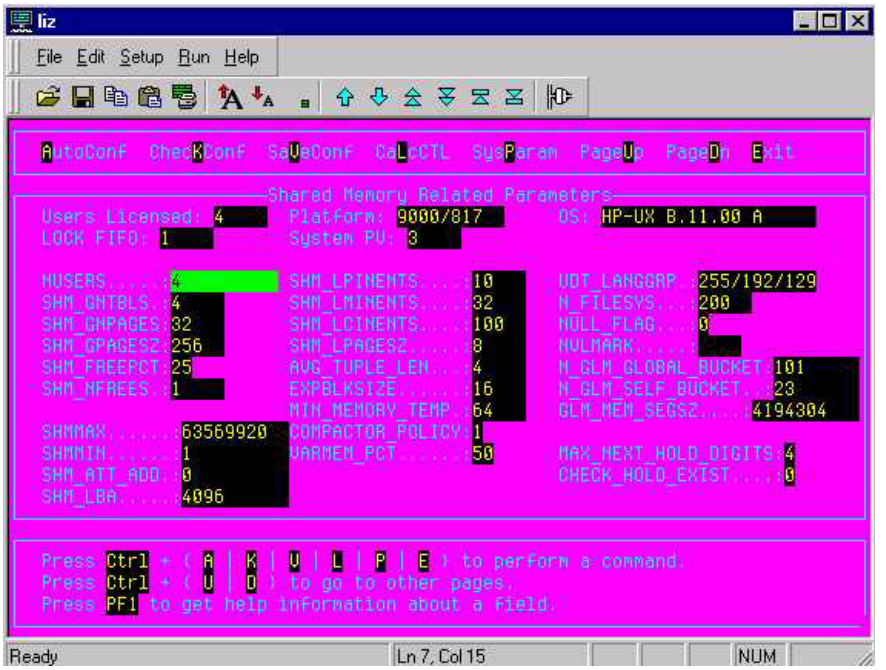
Description

The system-level **udtconf** command automatically sets udtconfig parameters for shared memory. Although shared memory requirements are highly application- and platform-dependent, udtconf can provide suggestions for udtconfig parameters and provide information about the actual state of your system.

For detailed information about udtconf, see *Administering UniData.*

Example (UniData for UNIX)

The following example shows the main screen of the udtconf utility:

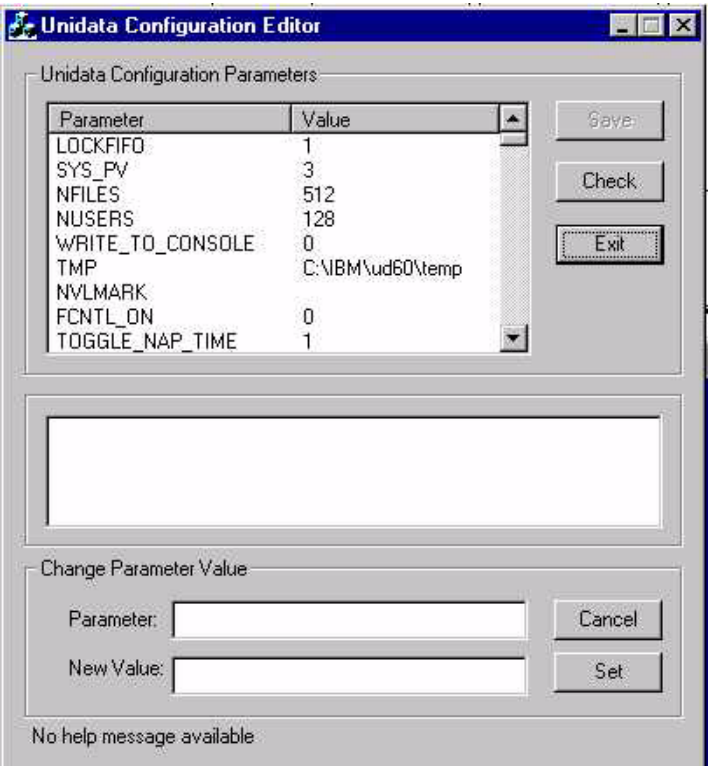


To advance to a field displayed on the screen, press TAB. To page down, press CTRL+D. To page up, enter CTRL+U. The udtconf utility displays warning messages if some of the kernel parameters are not adequate to support the values udtconf calculates. Make sure that the kernel parameter for semaphore undo structures, usually semmnu, is adequate to support the number of authorized users prior to running udtconf.

Settings for the udtconfig parameters NUSERS, SHM_GNTBLS, N_TMQ, and N_PGQ, and N_GLM_GLOBAL_BUCKET are based on the number of authorized users. Although udtconf displays warning messages if kernel parameters are not adequate to support these settings, the udtconfig file is updated with the values you set if you choose to ignore the warnings. In this case, UniData may not be able to start. For more information about configuring your UniData system, see Administering UniData.

Example (UniData for Windows Platforms)

The following example shows the main screen of the udtconf utility:



To view the udtconfig parameters, scroll through the list.The udtconf utility displays warning messages if some of the system-level parameters are not adequate to support the values udtconf calculates. To change the value of a parameter, double-click the parameter, enter the setting in the **New Value** box, and then click **Set**. To save your changes, click **Save**. To verify the settings against operating system limitations, click **Check**. To exit the program without saving changes, click **Exit**.

Settings for the udtconfig parameters NUSERS, SHM_GNTBLS, N_TMQ, and N_PGQ, and N_GLM_GLOBAL_BUCKET, are based on the number of authorized users. Although udtconf displays warning messages if parameters are not adequate to support these settings, the udtconfig file is updated with the values you set if you choose to ignore the warnings. In this case, UniData may not be able to start. For more information about configuring your UniData system, see *Administering UniData*.

udtinstall

Syntax

udtinstall [-f *filename*][-c]

Description

The system-level **udtinstall** command installs UniData.

Note: This command is supported on UniData for UNIX only.

For information about installing UniData, see *Installing and Licensing UniData Products*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-f *filename*	Indicates that all required user input is included in an ASCII file names *filename*.
-c	Automatically invokes confprod after the installation process is complete, using its default options.

udtinstall Parameters

udtlangconfig

Syntax

udtlangconfig

Description

The system-level **udtlangconfig** command completes the following tasks:

- Changes the language group for the current installation of UniData.
- Converts ASCII values used for UniData delimiters and other reserved characters using the UniData convmark command for all files in the demo database and *udthome*/sys directories on UniData for UNIX or *udthome*\sys directory on UniData for Windows Platforms.

  Warning: On UniData for UNIX, these directories may not contain any UNIX links (created with the UNIX ln command). convmark produces an error message and aborts if they do.

- Starts UniData in the language you specify.

Note: To execute the udtlangconfig command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms, and be in the udtbin directory.

Language group numbers correspond to the record mark value (@RM), print at value (@PRINT_AT), and the null value. If you are currently running UniData 3.3.2i, you need to run udtlangconfig to change the print at value from the previous 128 to the new 130 ASCII value.

Example

The following example shows the entire udtlangconfig process. In the section that prompts you to enter the language, UniData sets the default to match the operating system LANG setting. In this example, LANG is set for French.

```
# udtlangconfig

Starting configuration of UniData RDBMS system.
The following prompts have the default answers in brackets [],
press Enter to accept these answers.

Using UDTBIN=/usr/ud60/bin

WARNING: 'stopud -f' will stop the Unidata System with force.
This may not guarantee the consistency of the database files.

Would you like to continue?(y/n) [n]
y
SM stopped successfully.
CLEANUPD stopped successfully.
SBCS stopped successfully.
SMM stopped successfully

Unidata R6.0 has been shut down.

Please select the appropriate language from the list below:
Language Locale
------------------------------
ENGLISH C
ENGLISH_UK english
ENGLISH_G2 C
JAPANESE japanese.euc
FRENCH french
------------------------------
Please enter language from above list [FRENCH]:
FRENCH
Input complete, UniData processing...
Using UDTBIN=/usr/ud60/bin

All output and error logs have been saved
to /usr/ud60/bin/saved_logs directory.
SMM is started.
SBCS is started.
CLEANUPD is started.
SM is started.

UniData R6.0 has been started.

You now have completed the configuration process.
This is the end of the configuration process.
```

udtmon

Syntax

udtmon

Description

The system-level **udtmon** command starts the Monitor / Profile utility, part of the UniData System Administration Manager (USAM). Monitor / Profile is a menu-driven monitoring tool that provides you with information about UniData user and system activity.

To exit the Monitor / Profile utility, continue pressing ESC. The cursor returns to the environment from which you entered the utility.

Note: udtmon is supported on UniData for UNIX only.

You can select from ten different displays that show system resource use, in either text or graphic display. For more information about Monitor/Profile, refer to Using USAM.

Examples

In the following example, UniData opens a USAM session:

```
:!udtmon
UniData Monitor Utility Version 1.1.5
Display Configuration Help

Display statistics on use of Unidata and the system over a time
interval.
```

udtts

Syntax

udtts

Description

The system-level **udtts** command starts a UniData session when you are using device licensing. For UniData to run, the product must be installed and licensed, and the following environment variables must be set correctly:

- UDTBIN must be set to your UniData bin directory
- UDTHOME must be set to your UniData home directory

Note: You must use udtts to enter a UniData session if you are using device licensing. If you use udt, device licensing has no effect.

Execute this command at the system prompt.

Consult your system administrator for information about setting up your UniData environment.For a full description of the UniData environment variables, see *Administering UniData.*

Examples

The following example shows how to start a UniData session:

```
% udtts
UniData Release 6.0 Build: (4112)

(c) Copyright IBM Corporation 2002.
All rights reserved.
Current UniData home is /liz1/ud60/.
Current working directory is /home/claireg.
:
```

In the following example, the user attempted to start a UniData session when the UniData daemons had not been started. To correct this problem, you must first start UniData with the startud command.

```
# $UDTBIN/udtts
Start SMM first!
Related Command
BYE
```

UDT.OPTIONS

Syntax

UDT.OPTIONS [*n* {ON | OFF}]

Description

The ECL **UDT.OPTIONS** command modifies command behavior. The setting remains in effect throughout the UniData session or until you reset it.

By setting various UDT.OPTIONS ON or OFF, you can guide behaviors such as the following:

- How UniData sorts alphanumeric data for right-justified sorts.
- How UniData handles page breaks.
- The kind of message that UniData displays when you delete data from a file using a select list.
- Whether to suppress the echo of a prompt character and data when UniData passes data to a UniBasic program to fill an input statement.
- Where UniData returns control after a Proc executes a UniBasic program.

To use a combination of options, you must set each one separately.

For descriptions of the effects of each UDT.OPTION, see the *UDT.OPTIONS Commands Reference*.

Tip: If you want UniData to set UDT.OPTIONS ON every time you start a UniData session, create a login paragraph that turns them on every time you login. For more information about creating login paragraphs, see Using UniData.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
n	The number of the UDT.OPTION you want to change.
ON	Switches the UDT.OPTION on.
OFF	Switches the UDT.OPTION off.

UDT.OPTIONS Parameters

Examples

To view the current setting of each option, enter the ECL UDT.OPTIONS command at the UniData ECL (colon) prompt:

```
:UDT.OPTIONS
1 U_NULLTOZERO OFF
2 U_PSTYLEECL OFF
3 U_SHLNOPAGE OFF
...
109 U_TELNET_NODELAY OFF
110 U_OCONV_EMPTY_STR OFF
111 U_NT_CTRL_C_IGNORE OFF
```

To set an individual option, use the option number with the UDT.OPTIONS command and indicate whether to turn the option ON or OFF. The next example turns UDT.OPTIONS 2 ON:

```
:UDT.OPTIONS 2 ON
:
```

uniapi_admin

Syntax

uniapi_admin

Description

The system-level **uniapi_admin** command starts the ObjectCall Administration tool.

UNIENTRY

Syntax

UNIENTRY [DICT] *filename record*

Synonyms

ENTRO, UFORM

Description

The ECL **UNIENTRY** command invokes the UniData file-building tool. This command sets an exclusive lock on the file being accessed.

When you use UniEntry to modify the dictionary of a file, UniData uses the DICT.DICT dictionary to format the display of dictionary attributes. For more information about using UniEntry to build UniData files, see *Using UniData.*

UniEntry displays all D-type attributes. To display multivalued attributes, you must select the attribute number and press ENTER.

You cannot use UniEntry to enter the null value into an attribute.

Note: If UniData cannot modify or delete a record due to the presence of a trigger, UniData displays an informational message that the update or delete operation was not executed.

Regarding other editors:

- The ECL AE command invokes the UniData Alternate Editor. You can use this line editor to edit UniData hashed files and UniBasic source programs.
- The ECL ED command invokes the standard operating system editor supported by UniData. See ED, in this manual, for more information.
- The ECL VI command invokes vi, the UNIX System V visual editor, from within UniData.

- You can edit UniData hashed files and DIR-type files with any ASCII text editor. See your operating system documentation for more information on supported editors. Be aware, though, of any changes or conversions the editor might make to files it opens.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Opens the dictionary portion of the file.
filename	A UniData file to access.
record	A record in *filename*.

UNIENTRY Parameters

Example

The following example opens a record in the CLIENTS demo file and displays each attribute in the record.

```
:UNIENTRY CLIENTS 9999

CLIENTS RECORD ID==>9999

0 @ID=9999
1 FNAME=Paul
2 LNAME=Castiglione
3 COMPANY=Chez Paul
4 CITY=Paris
5 STATE=
6 ZIP_CODE=75008
7 COUNTRY=France
8 ==>ADDRESS
9 ==>PHONE_NUM PHONE_TYPE

FROM 8 to 9 ARE MULTI VALUED FIELDS SCREENS.

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=
```

Related Commands

AE, ED

UNSETDEBUGLINE

Syntax

UNSETDEBUGLINE

Description

The ECL **UNSETDEBUGLINE** command releases the port that you were using for dual-terminal debugging in UniBasic.

For more information about UniBasic and the UniBasic debugger, see *Developing UniBasic Applications*.

Related Commands

DEBUGLINE.ATT, DEBUGLINE.DET, SETDEBUGLINE

UNSETLINE

Syntax

UNSETLINE *line*

Description

The ECL **UNSETLINE** command disconnects a communications line that had been initialized with SETLINE for use during the current UniData session. If you do not specify a parameter, UniData displays the current setting.

UNSETLINE modifies the ASCII file *udthome*/sys/lineinfo on UniData for UNIX or *udthome*\sys\lineinfo on UniData for Windows Platforms.

Note: To execute the UNSETLINE command, you must log in as root on UniData for UNIX or as Administrator on UniData for Windows Platforms.

For information about initializing a communication line, see SETLINE and LINE.ATT.

Related Commands

UniData

LINE.ATT, LINE.DET, LINE.STATUS, PROTOCOL, SETLINE

UniBasic

GET, SEND For information, see the *UniBasic Commands Reference*.

UPDATE.INDEX

Syntax

UPDATE.INDEX *filename*

Description

The ECL **UPDATE.INDEX** command applies deferred updates to alternate key indexes when automatic updating was disabled by DISABLE.INDEX. If you are running the Recoverable File System (RFS), the ENABLE.INDEX command automatically updates the index.

You do not have to execute ENABLE.INDEX before updating with UPDATE.INDEX.

Tip: Depending on the number and size of your index, automatic updating may adversely impact system performance. By deferring updating to a time of low activity, you may improve system performance during peak activity times.

Examples

In the following example, UniData applies deferred updates to the index for the ORDERS demo file:

```
:UPDATE.INDEX ORDERS
Total Defferred Updates Applied: 1
:
```

To find out if an index file has updates pending, use the LIST.INDEX command to display data about the file, as shown in the next example. Notice the entry on the line for Index updates. This tells you that automatic updating is disabled and there are pending updates.

```
Alternate Key Index Details for File ORDERS Page 1
File.................. ORDERS
Alternate key length.. 20
Node/Block size....... 4K
OV blocks............. 1 (0 in use, 0 overflowed)
Indices............... 4 (1 D-type)
Index updates......... Disabled, Indices require updating
Index-Name...... F-type K-type Built Empties Dups In-DICT S/M F-
no/VF-expr....
NAME V Txt Yes Yes Yes Yes S TRANS('CLIENTS',
CLIENT_NO,'FNAME','X'): " ": TRANS('CLIENTS',CLIENT_NO,'LNAME','
X')
GRAND_TOTAL V Num Yes Yes Yes Yes S PRICE*QTY; SUM(SUM(@1))
COUNTRY V Txt Yes Yes Yes Yes S TRANS('CLIENTS', CLIENT_NO,'COUNT
RY','X')
PRODUCT_NO D Num Yes Yes Yes Yes M 4
:
```

Note: DELETE.INDEX fails when an index is disabled.

Related Commands

BUILD.INDEX, CREATE.INDEX, DELETE.INDEX, DISABLE.INDEX, ENABLE.INDEX,LIST.INDEX, UPDATE.INDEX

updatesys

Syntax

updatesys

Description

The system-level **updatesys** command updates the installed version of UniData.

Note: This command does everything the system-level udtinstall command does, except that it updates your udthome/sys directory instead of creating a new one. This leaves your global catalog space intact.

For information about installing and upgrading UniData, see *Installing and Licensing UniData Products.*

updatevoc

Syntax

updatevoc [-[A | C | I | O | N | S]][*directory*]

Description

The system-level **updatevoc** command updates the VOC file for an account. If you do not name a directory or use any options, UniData updates the VOC file in the current account and appends notes regarding changes to the vocupgrade file. UniData updates the VOC file from a master file located in *udtbin*/VOCUPGRADE on UniData for UNIX or *udtbin*\VOCUPGRADE on UniData for Windows Platforms.

Depending on the option(s) selected, updatevoc takes one or more of the following actions:

- Compares the account VOC file to VOCUPGRADE.
- Notes differences between the two files, appending them to or overwriting the vocupgrade file, in the current account.
- Displays informational messages and updates records in the local accounts VOC file.
- Updates the version record in the VOC file, which is read by the VERSION command.

Tip: IBM recommends that you run this command on every UniData account after you upgrade to a new version of UniData. For tracking purposes, run updatevoc from within the account for which you are updating the VOC.

Parameters

The following table lists the updatevoc parameters. When you use these options in combination, use the minus sign only once, preceding the first option listed (such as in updatevoc -ACI).

Parameter	Description
directory	UniData account directory that contains the VOC file you intend to update.
-A	Adds new records to the VOC file in the specified directory and in all accounts that reside in the directory, but does not modify existing VOC records. Appends to vocupgrade notes on differences between the two VOC files.
-C	Adds new records to the VOC file in the specified directory, but does not modify existing VOC records. Does not add notes to vocupgrade on differences between the two VOC files. **Note**: The -I parameter overrides the -C parameter.
-I	Prompts for verification before adding or modifying records to the VOC file in the specified directory. Does not add notes to vocupgrade on differences between the two VOC files.
-O	Overwrites existing entries in the account's VOC without prompting for verification. Does not add notes to vocupgrade on differences between the two VOC files.
-N	Adds new records to the VOC file in the specified directory, but does not modify existing VOC records. Appends notes to vocupgrade on differences between the two VOC files.
-S	Adds new records to the VOC file in the specified directory, but does not modify existing VOC records. Suppresses the informational messages ordinarily displayed after each change.

updatevoc Parameters

Examples

The following example, taken from UniData for UNIX, uses the UNIX more command to display the contents of vocupgrade in a demo account.

Tip: Notice the phrases old item and new item that appear next to each entry. old item means that UniData has applied changes to an existing VOC entry. new item notes a difference between the two VOC files for which no change has been made.

```
:!more vocupgrade
BELL~K~BELL | old item |
CP~PA~SPOOL <<I2,FILENAME>> <<I3,ITEMID>> | old item |
CREATE~SQLV~VIEW~TABLE~INDEX | old item |
DEFAULT.LOCKED.ACTION~V~DEFAULT.LOCKED.ACTION | old item |
DROP~SQLV~VIEW~TABLE~INDEX~INTO | old item |
HELP.FILE~F~@UDTHOME/sys/HELP.FILE~@UDTHOME/sys/D_HELP.FILE | old
item |
HELP~V~HELP | old item |
```

The next example updates a demo VOC file. For this example, the VOC record for the SAMPLE keyword has been changed from type K to type V, so that it differs from the entry in VOCUPGRADE.

Notice that UniData adds new entries to the account VOC file but does not change the SAMPLE VOC record. If a change had been made to SAMPLE, the last message would indicate a new item saved to /home/carolw/demo/vocupgrade.

```
:AE VOC SAMPLE
Top of "SAMPLE" in "VOC", 2 lines, 8 characters.
001: V
002: SAMPLE
Bottom.

:
:!updatevoc -C
Now update /home/carolw/demo/VOC ... ...
Adding KEYDATA to VOC file
Adding KEYONLY to VOC file
Adding PARTTBL to VOC file
Adding VERSION to VOC file
Adding version to VOC file
Deleting ERRMSG from VOC file
366 total items in /disk1/ud51/bin/VOCUPGRADE.
6 items updated in VOC file.
1 old items saved to /home/carolw/demo/vocupgrade.
0 new items saved to /home/carolw/demo/vocupgrade.
:
```

The next example updates the local accounts VOC file, but does not record anything to the vocupgrade file.

```
:!updatevoc -O
Now update /home/carolw/demo/VOC ... ...
Adding SAMPLE to VOC file
Adding VERSION to VOC file
Adding ERRMSG to VOC file
Adding version to VOC file
Deleting ERRMSG from VOC file
366 total items in /disk1/ud41/bin/VOCUPGRADE.
5 items updated in VOC file.
0 old items saved to /home/carolw/demo/vocupgrade.
0 new items saved to /home/carolw/demo/vocupgrade.
```

Related Command

VERSION

usam

Syntax

usam

Description

The system-level **usam** command runs USAM (*UniData System Administration Manager*), an interactive utility. For detailed information on this utility, see *Using USAM and USAM Batch/USAM Print*.

To quit the USAM utility, press ESC continuously until you return to the environment from which you entered USAM.

Note: USAM is supported on UniData for UNIX only.

Use this command at the system prompt, or use the ECL ! (bang) command to execute this command from the ECL (colon) prompt.

USHOW

Syntax

USHOW [DICT] *filename* [*attribute* [*attributeN*...]]

Description

The ECL **USHOW** command generates lists of selected attributes from UniData files. This command is an implementation of the Prime Information SHOW command.

Parameters

The following table lists the USHOW parameters.

Parameter	Description
DICT	Lists the dictionary file.
filename	A UniData file.
attribute	The name of an attribute to display.

USHOW Parameters

Examples

The following example shows the result of USHOW with the ORDERS demo file:

```
:USHOW ORDERS PRODUCT_NO
Page no :1 of 13 No.ofrecs. selected 0 of 193
ORDERS|Product Number|
------------------------------------------------------------------------
--------------
1 | 912| | 55040| |

2 | 801| | 11000| |
3 | 941| | 50000| |
4 | 805| | 11140| |
5 | 830| | 55090| |
6 | 970| | 13003| |
7 | 863| | 40005| |
8 | 834| | 40007| |
9 | 861| | 56080| |
10| 890| | 54090| |
11| 914| | 40007| |
12| 803| | 10004| |
13| 832| | 10020| |
14| 972| | 10090| |
15| 860| | 57010| |
------------------------------------------------------------------------
--------------
Command :
S (range) - Select, C (range) - Clear, F - forwars, B - backwards
Range - ALL, VISIBLE, nn-nn, nn, nn-At
the Command : prompt, you can do any of the following:
 S Save a range of record IDs to a select list
 C Clear a range of record IDs
 F Move forward through the USHOW display
 B Move backward through the USHOW display
```

After creating a select list, UniData displays the active select list prompt (>). At this prompt, you can operate on the active select list or enter quit or QUIT to exit the USHOW process and end the UniData session.

To return to the UniData colon prompt, enter CLEARSELECT at the active select list prompt, or press your interrupt key (enable the interrupt key with PTERM -BREAK).

UV_RESTORE

Syntax

UV_RESTORE [-HDYNAMIC0 | -HDYNAMIC1] [-O] [-S] [-R] [-X *char_list]*[-K*n*] [-A *outputfile*] [-F [DICT | DATA | DIR | *filelist*]] [-D *uniVerse_path*] *acct_name*

Description

The system-level **UV_RESTORE** command restores a UniVerse account from tape to disk in UniData format.

The target account directory that you intend to restore must reside on the machine to which you are migrating. UV_RESTORE reads data from an account you specify by a full path (*uniVerse_path*) and restores it to a UniData account. If the UniData account does not exist, UV_RESTORE creates it and names it *acct_name*.

Use this command at the system prompt.

Tip: If very large data files (larger than 1 gigabyte) have been saved from UniVerse, IBM recommends that you create the target UniData files as dynamic before you begin the restore. Assign a modulo to accommodate a file about 40 percent larger than the original UniVerse file.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
-HDYNAMIC0	Creates dynamic files with hash type 0.
-HDYNAMIC1	Creates dynamic files with hash type 1.
-O	Forces overwriting of files in the UniData account. (The default UV_RESTORE behavior is to check the account for existing file names, and if a file exists, UniData prompts to skip or overwrite the file).
-R	Removes each UniVerse file from the disk after conversion. This is a useful parameter if you are short on disk space.
-S	Truncates file names to 12 characters in length. This parameter is not necessary if you run UV_RESTORE on an operating system that automatically shortens file and program names.
-X *char_list*	*char_list* indicates characters to be considered invalid for: ■ file names ■ account names ■ record IDs in DIR-type files While restoring, UniData converts these characters to underscore (_). If the resulting name conflicts with an existing account name, UniData adds a character to the end of the name to make it unique. For example: A&B becomes A_B. If A_B is used by another file, the name becomes A_Ba. Default invalid characters are the following: space * ? / & ‘. You cannot specify nonprinting characters as invalid. Do not separate characters in *char_list* with spaces or commas.
-K *n*	Defines the size of the internal memory buffer (in kilobytes). Default size is 8000 K. System restoration performs best when buffer size is large. Change the size to match the capacity of your operating system.

UV_RESTORE Parameters

Parameter	Description
-A *outputfile*	Creates *filename*, an ASCII text file, in the current directory, containing statistics about each file on the tape. -A does not restore files.
-F [DICT | DATA |DIR | *filelist*]	Loads only certain kinds of files: ■ DICT– dictionary ■ DIR – directory (including DIR and LD) ■ DATA – hashed files (including DATA and LF) ■ *filelist* – files listed in *filelist*, an ASCII text file that you create. To convert files from different UniVerse accounts, specify the path (*uniVerse_path*) in *filelist*. UV_RESTORE converts the files into a single UniData account. **Tip**: Use the -D parameter with this option so that you do not have to include the full path for each file in *filelist*.
-D *uniVerse_path*	Designates the location of the UniVerse account directory.
acct_name	Name of the UniVerse account to be restored.

UV_RESTORE Parameters (continued)

VCATALOG

Syntax

VCATALOG *filename catalogname programname*

Description

The ECL VCATALOG command compares the object file and the compiled program in the global catalog file byte-by-byte. If the source file has been modified after the program was cataloged, VCATALOG returns a negative result.

For more information on UniBasic, see *Developing UniBasic Applications*.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The file that contains the compiled version of the program. This must be a DIR-type record in the VOC file.
catalogname	The cataloged name of the object code for the program. By default, this is the same as the program name, however, a different name may be assigned when the program is cataloged.
programname	The program that is globally cataloged under this name or *catalogname*.

VCATALOG Parameters

Examples

In the following example, UniData verifies a program called PSTLCODE_FMT which resides in the BP_SOURCE file of the demo database:

```
:VCATALOG BP_SOURCE PSTLCODE_FMT PSTLCODE_FMT
Program 'PSTLCODE_FMT' verifies.
:
```

The following example demonstrates VCATALOG returning a negative result, indicating that the source code has been changed since the program was cataloged.

```
:VCATALOG BP TEST
Program 'TEST' does not verify.
```

verify2

Syntax

verify2 [-Y | -y] [[-H | -h] *block address*][[-O | -o]*block address*]

Description

The system-level **verify2** command detects corrupted files. You must have read and write permissions for each file you check with verify2.

Warning: UniData must not be running when you execute this command.

For more information about verify2 and detecting corrupted files, see *Administering UniData.*

Parameters

The following table describes the verify2 parameters.

Parameter	Description
-Y | -y	Writes the file name and @ID of damaged records to a file called /tmp/vrfy2.*pid*, where *pid* is the process ID of the process than ran verify2.
[-H | -h] *block address*	Bypasses checking the block whose hexadecimal address is *block address*. This option allows you to bypass a single damaged block while examining the rest of the file.
[-O | -o] *block address*	Same as -h option, but -o allows you to bypass a block in the overflow portion of a dynamic file.

verify2 Parameters

Related Commands

dumpgroup, fixfile, fixgroup, guide

VERSION

Syntax

VERSION

Description

The ECL **VERSION** command displays the most current UniData product version numbers recorded in the VOC file and the UniData bin directory as well as the current patch level.

Example

In the following example, the UniData displays the versions of all UniData products licensed on a system:

```
:VERSION
Module Name Version Licensed
UniData RDBMS..... 6.0 Yes
UniData OFS/NFA..... 6.0 Yes
UniServer for ObjectCall. 6.0 Yes
RFS/TP..... 6.0 Yes
Device License..... 6.0 Yes
ODBC/UniOLEDB..... 6.0 Yes
UniObjects..... 6.0 Yes
```

VI

Syntax

VI *filename record*

Description

The ECL **VI** command invokes the vi editor on UniData for UNIX, or the MS-DOS editor on UniData for Windows Platforms from within UniData. VI opens the file *filename* and *record* you name. For more information on these editors, see your host operating system documentation.

Regarding other editors:

- The ECL AE command invokes the UniData Alternate Editor. You can use this line editor to edit UniData hashed files and UniBasic source programs.
- The ECL ED command invokes the standard operating system editor supported by UniData. See ED in this manual for more information.
- UniData supplies UniEntry for modifying UniData records.
- You can edit UniData hashed files and DIR-type files with any ASCII text editor. Refer to your operating system documentation for more information on supported editors. Be aware, though, of any changes or conversions the editor might make to files it opens.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The UniData file to be opened by the editor.
record	The record in *filename*.

VI Parameters

Example

The following example shows how UniData invokes the vi editor from within UniData in order to access a record in the CLIENTS demo file:

```
:VI CLIENTS 9999
Paul
Castiglione
Chez Paul
45, reu de Rivoli
Paris
75008
France
3342425544}3342664857
Work}Fax
~
~
```

WAKE

Syntax

WAKE *pid*

Description

The ECL **WAKE** command activates a UniData process (*pid*) that has been paused with either the ECL PAUSE command or the UniBasic PAUSE command. If the process you specify has not been paused, UniData disregards the next PAUSE issued for that process.

Examples

Note: See the ECL PAUSE command for more examples.

The following series of examples demonstrates executing the WAKE command before executing PAUSE.

First, the user executes the listuser command to identify the process ID for the current UniData session. The process ID is the located in the USRNBR column. In this example, 11523 is the process ID for the session to pause:

```
:LISTUSER
Licensed/Effective # of Users Udt Sql Total
32 /32 2 0 2
UDTNO USRNBR UID USRNAME USRTYPE TTY TIME DATE
1 11523 1172 claireg udt ttyp3 11:42:50 Jun 05 1999
2 11528 0 root udt pts/0 11:43:45 Jun 05 1999
```

Next, the user initiates a second UniData session and executes a WAKE against process 11523, the process identified in the preceding step:

```
:WAKE 11523
```

Finally, the user attempts to pause the first session with the PAUSE command, but the command is ignored by UniData because of the previously issued WAKE against this process:

```
:PAUSE
:
```

Related Commands

UniData

LIST.PAUSED, PAUSE

UniBasic Commands

PAUSE, WAKE – For information, see the *UniBasic Commands Reference*.

WHAT

Syntax

WHAT

Description

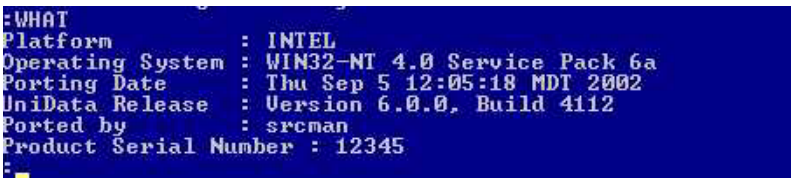
The ECL **WHAT** command displays the system information stored in *udthome*/include/sysconfig on UniData for UNIX or *udthome*\include\sysconfig on UniData for Windows Platforms.

Examples

The following example shows a WHAT command display. The Product Serial Number on the last line in the example is the text entered at the product number prompt during udtinstall or updatesys on UniData for UNIX.

```
:WHAT
Platform         : HPUX11
Operating System : HP-UX hal B.11.00 A 9000/820 2000945515 two-
user license
Porting Date     : Sep. 05, 02
UniData Release  : 60_020905_4112
Ported by        :
Product Serial Number : serial_number
```

The next example illustrates output from the WHAT command on UniData for Windows Platforms:



WHERE

Syntax

WHERE

Description

The ECL **WHERE** command displays the current account.

Example

The following example, taken from UniData for UNIX, shows a WHERE command display:

```
:WHERE
/home/carolw/demo
:
```

WHO

Syntax

WHO

Description

The ECL **WHO** command displays information about users logged onto the system, including:

- User ID
- Port number
- Date of login
- Time of login

Example

In the following example, the UniData lists the users logged into the system:

```
:WHO
carolw pty/ttyv0 Jun 17 11:52
peggys pty/ttyv2 Jun 17 10:59
:
```

XTD

Syntax

XTD *hex*

Description

The ECL XTD command converts a hexadecimal number to its decimal equivalent.

If the input number is longer than 8 digits, only the rightmost 8 digits are converted. If the input contains characters other than 0-9 and A-F, XTD returns 0.

The valid hexadecimal value ranges from 0 to FFFFFFFF. hexadecimal numbers in the range between 80000001 (-2,147,483,647) and FFFFFFFF (-1) are considered negative, and produce a negative decimal result.

XTD performs the inverse operation of the DTX command.

Example

In the following example, various hexadecimal values are translated to decimal values:

```
:XTD FF
255
:XTD 34ab
13483
:XTD Ab2
2738
:XTD K01
0
:
```

Related Command

DTX

Index

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Symbols