Caché MultiValue Commands Reference

Version 2018.1
2018-12-14

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Caché MultiValue Commands Reference
Caché Version 2018.1 2018-12-14

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book provides reference material for the command line commands of the Caché MultiValue implementation.

This book contains the following sections:

- MultiValue Commands
- The MVIMPORT Command
- The PROTOCLASS Command
- Command Stack Commands
- Error Messages

There is also a detailed Table of Contents.

Other related topics in the Caché documentation set are:

- *Using the MultiValue Features of Caché*
- *Operational Differences between MultiValue and Caché*
- *Caché MultiValue Query Language (CMQL) Reference*
- *The Caché MultiValue Spooler*

For general information, see *Using InterSystems Documentation*.

1

MultiValue Commands

This chapter provides an alphabetical listing of the command line commands supported by the Caché MultiValue Shell. In MultiValue database systems, these commands are also known as “verbs”. Several of these commands have an alternate name that includes a **.VERB** suffix. These suffix forms are not listed here.

Most command names that contain punctuation exist in two variant forms: the hyphen form and the dot form. For example, **CREATE-ACCOUNT** and **CREATE.ACCOUNT**. InterSystems supports these two forms for compatibility with different vendor versions of MultiValue code. In most cases, these two forms are synonymous. When this is the case, only one of the forms is listed in this chapter. The most notable exception is **SP-EDIT** and **SP.EDIT**, which provide different functionality.

Command names, command name keywords, letter code options, and many command arguments are not case-sensitive. Account names and item ID values are case-sensitive.

Letter code options are provided for some command line commands. These letter codes are always optional. Some commands can specify more than one letter code option. Multiple letter codes can be specified in any order. A letter code, or series of letter codes, is preceded by an open parenthesis; the closing parenthesis is not required. The letter codes must be the final item in the command syntax; keyword options, such as DET-SUPP, must appear before the letter codes.

Note: Some MultiValue flavors (D3, jBASE, and UniVerse) permit letter code options enclosed in parentheses before the final item in the command syntax (for example, `LIST VOC (P) 'BASIC'`). Caché MultiValue emulation of these flavors *does not* support this syntax; letter codes options must be the final item in the command syntax.

In most cases, only one MultiValue command can be specified on a MVShell command line. You can, however, specify multiple **;** commands on the same command line.

1.1 # (pound sign)

The **#** command causes the statement following it to be interpreted as an ObjectScript command.

```
# commandline
```

A ObjectScript command line can consist of one or more ObjectScript statements, separated by blank spaces. This ObjectScript command line is immediately executed and the results returned to the MultiValue prompt at the Terminal.

The **#** must be separated with or one or more spaces from the following ObjectScript command line.

If *commandline* changes the current namespace (for example, by issuing a ZNSPACE command), the Caché MultiValue Shell restores the initial namespace upon exiting ObjectScript.

The **#** *should not* be followed by the ObjectScript **MV** MultiValue Shell invocation command. Nested MultiValue Shell invocations may cause unexpected problems.

The following is an example of the **#** command:

```
USER:#  SET x="Hello World!" WRITE !,x
```

The **#**, **[**, and **COS** commands are synonyms.

See Also: COS

1.2 ; (semicolon)

The **;** command causes the statement following it on the command line to be interpreted as an MVBasic statement.

```
; basicstatement
```

This MVBasic statement is immediately executed and the results returned to the terminal.

The **;** can be immediately followed by an MVBasic statement, or one or more spaces can be placed between them. Unlike most MultiValue Shell commands, you can specify multiple **;** commands on the same command line. Each MVBasic statement requires its own **;** command.

Thus, the following is a valid use of the **;** command:

```
USER:  ;   PRINT "hello" ;PRINT "world"
```

You can issue the **;** command either from the MultiValue Shell prompt, as shown above, or from the MultiValue debug prompt following an MVBasic DEBUG statement, as in the following example:

```
USER:;myvar="ABC"
USER:;DEBUG

<BREAK>+1^MVBASIC1048.mvi
Source Id: File: Line:0
USER 7d1>;CRT "my variable",myvar
my variable     ABC
USER 7d1>
```

See Also: BASIC, RUN

1.3 [(left square bracket)

The **[** command causes the statement following it to be interpreted as an ObjectScript command.

```
[ commandline
```

A ObjectScript command line can consist of one or more ObjectScript statements, separated by blank spaces. This ObjectScript command line is immediately executed and the results returned to the MultiValue prompt at the Terminal.

The **[** can be immediately followed by an ObjectScript command line, or one or more spaces can be placed between them.

If *commandline* changes the current namespace (for example, by issuing a ZNSPACE command), the Caché MultiValue Shell restores the initial namespace upon exiting ObjectScript.

The **[** *should not* be followed by the ObjectScript **MV** MultiValue Shell invocation command. Nested MultiValue Shell invocations may cause unexpected problems.

The following is an example of the **[** command:

```
USER:[  SET x="Hello World!" WRITE !,x
```

The **[**, **#**, and **COS** commands are synonyms.

See Also: COS

1.4 << ... >> (inline prompting)

The << . . . >> command causes the Caché MultiValue Shell to prompt for an input value.

```
<<[code,]prompt[,patcode]>>
```

The *prompt* is a text prompt that requests a user input value. It can include blanks spaces and any character except the comma.

The optional *code* is separated from the prompt by a comma. You can specify multiple comma-separated *code* values. The following *code* values are supported:

A	Prompt even when prompt was previously issued.
C*n*	Uses the *n*th word on the command line as an argument in an inline prompt. This option allows the user to enter responses to inline prompts at TCL on the same line following the paragraph name.
F(*filename*,*record*[,*att.num*, *value.num*,*subvalue.num*]	Retrieves input from record in *filename*, and optionally from *att.num*, *value.num*, and *subvalue.num*. Prompt text is optional.
I*n*	Uses the *n*th word on the command line as an argument in an inline prompt. This is the same as C*n* when *n* is supplied. Else, prompts for *n* if *n* is not specified.
R	Prompts repeatedly for multiple values. Prompt repeats until the user responds to the prompt with the Enter key. R(text) does the same thing, but inserts *text* between the user input values.
@(BELL)	Ring the bell.
@(CLR)	Clear the screen.
@(TOF)	Move cursor to top of form.
@(col,row)	Move cursor to specified column and row. Rows and columns are counted from 0.

The optional *patcode* can take two kinds of values:

- A *patcode* without parentheses matches the input value with a pattern match code. For example 6A requires that the input value consist of 6 alphabetic characters. If the input value does not match the *patcode*, an error message is displayed and the user is prompted again until a valid input (or no input) is specified. The available *patcode* pattern match values are listed in the MATCH pattern matching operator reference page in the *Caché MVBasic Reference*.

- A *patcode* with parentheses matches the input value with an **ICONV** conversion type code. For example, (D) requires that the input value be a valid date, such as 2/28/2009 or 2009–02–28. The input value is validated, but not converted. If the input value does not pass *patcode* validation, an error message is displayed and the user is prompted again until a valid input (or no input) is specified. The available *patcode* conversion validation values are listed in the ICONV function reference page in the *Caché MVBasic Reference*.

Inline prompting can be used by itself or within another MV command or MVBasic statement. For example:

```
USER:<<input a command>>
input a command=

USER:SLEEP <<seconds>>
seconds=
```

The user input value can be a literal, or the name of a defined variable.

In the following MVBasic example, several << ... >> prompts are used:

```
USER:;PRINT "The quick <<color>> <<animal>> jumped over the <<adjective>> dog."
color=brown
animal=fox
adjective=lazy
The quick brown fox jumped over the lazy dog.
```

Note that a *prompt* value is requested once but can be used multiple times:

```
USER:;PRINT "The quick brown <<animal>> jumped over the lazy <<animal>>."
animal=fox
The quick brown fox jumped over the lazy fox.
```

To avoid this reuse of a prompt value, use the *code* value of A. This forces prompting of a previously defined *prompt*:

```
USER:;PRINT "A <<flower>> is a <<flower>> is a <<A,flower>>."
flower=rose
flower=tulip
A rose is a rose is a tulip.
```

See Also: CLEARPROMPTS

1.5 Ctrl-X

The **Ctrl-X** (Ctrl key + “X”) command clears the current command line, resetting the cursor to column 1. The letter X can be uppercase or lowercase.

1.6 ABORT

The **ABORT** command terminates the current process and returns to either the MV Shell or the EXECUTE command that invoked the process.

```
ABORT (expr (, expr) )
```

ABORT runs the ON.ABORT paragraph, if present.

1.7 ASSIGN

The **ASSIGN** command assigns an I/O configuration setting. The following assignments are supported:

```
ASSIGN form-queue TO LPTR n [-WAIT]
```

Assigns a form queue spool device to a LPTR device. The –WAIT keyword is a no-op. The *form-queue* can be specified either by name or by number. The default form queue has the name STANDARD, and a form queue number of 0. It can be specified as “STANDARD”, “0”, “F0”, “FN0”, or “FQ0”. The *n* device number variable can take an integer value between 0 and 255 (inclusive).

```
ASSIGN n TO SYSTEM(5)
```

Assigns a page number *n* for page headers and footers. The page number displayed with a HEADING is *n*-1; the next time HEADING is executed, the page number increments to *n* as the current page number. For further information, refer to the SYSTEM function in *Caché MVBasic Reference*.

```
ASSIGN termname TO SYSTEM(7)
```

Assigns a terminal type *termname* to the current process. For further information, refer to the **CHOOSE.TERM** command in this manual, and the SYSTEM function in *Caché MVBasic Reference*.

See Also: UNASSIGN

1.8 ATTACH.ACCOUNTS

The **ATTACH.ACCOUNTS** command searches all existing namespaces for MultiValue accounts and records them in the table of MV accounts.

```
ATTACH.ACCOUNTS
```

Specifying **ATTACH.ACCOUNTS** does not overwrite any existing MultiValue account information; it only adds account information for accounts that have not yet been recorded. It is run to make accounts copied or moved from a different system or Caché instance visible in the SYSTEM file. For details on the relationship between accounts and namespaces, and the naming conventions used for each, refer to “MV Accounts and Caché Namespaces” in *Operational Differences between MultiValue and Caché*.

See Also: CREATE.ACCOUNT

1.9 AUTOLOGOUT

The **AUTOLOGOUT** command sets and displays the time setting for automatic logout.

```
AUTOLOGOUT [minutes]
```

Specifying **AUTOLOGOUT** with no operand returns the current autologout setting: either “Automatic logout is set for *x* minutes” or “Automatic logout is disabled”. The default is “disabled”. **AUTOLOGOUT** with no operand returns the most recent setting; it does not return the number of minutes remaining.

Use the optional *minutes* argument to set an automatic logout time. Automatic logout can be set to a positive integer number of minutes. A fractional number of minutes is truncated to its integer portion. Setting *minutes* to 0, a fraction less than 1, or a negative number disables automatic logout.

1.10 BASIC

The **BASIC** command compiles one or more MVBasic programs stored in a file.

```
BASIC filename [itemspec] [(SXZ]
```

The *filename* specifies a file created using CREATE.FILE. The *itemspec* specifies one or more existing MVBasic programs within *filename*. (You can create an *itemspec* MVBasic program within *filename* using ED.) If the *filename* file doesn't exist **BASIC** generates a [201] error message. If the specified *itemspec* item doesn't exist **BASIC** generates a [202] error message.

When an *itemspec* item successfully compiles **BASIC** displays the item name and generates a [B0] message. When an *itemspec* item fails to compile **BASIC** displays the source code line where the error occurred and generates a [B100] message. If any compilation failed during the **BASIC** operation, **BASIC** generates a [258] error at the end of the compile with a message indicating how many source file items failed to compile.

The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection].
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file; the default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the program to be compiled is stored in a named data section.

The *itemspec* specifies the item ID of each program to be compiled. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all MVBasic programs in the file should be compiled. If *itemspec* is omitted, **BASIC** uses the active select list as the source for the list of item IDs. If there is no active select list, **BASIC** issues a "Item Id:" user prompt.

As each item is compiled, **BASIC** updates the @RECORD special variable, a dynamic array consisting of the lines of the last *itemspec* successfully compiled.

The optional letter codes specify compilation options. You can specify one or more letter codes in any order. Letter codes must be preceded by an open parenthesis; a closing parenthesis is not required. Following compilation, S produces an active select list of all programs that failed to compile. The X and Z letter code options reduce the size of the compiled program; they should only be used when a compiled program exceeds maxsize. X removes the variable names; this means that the debugger cannot display variables, and the variable name is not reported if an <UNDEFINED> error occurs. Z removes the line table; this table is used to derive the source line number when there is an error to report. Without the line table, an error location cannot be determined.

After compiling a MVBasic program, you can execute it using the **RUN** command. You can catalog it in the VOC using the **CATALOG** command, after which it can be executed simply by invoking it as a verb.

See Also: ; (semicolon), CATALOG, MVI, RUN

1.11 BLOCK.PRINT

The **BLOCK.PRINT** command prints a text as large-format letters.

```
BLOCK.PRINT text
```

You can specify any printable character(s) as *text*, including a text containing blank spaces; no enclosing quotes are required. **BLOCK.PRINT** uses multiple “X” characters to print each character of *text* as a large block character. By default, it inserts a line break each time it encounters a space character or a string of space characters. An error message is returned if a string of characters is too long to print on one line.

A *text* does not require delimiters. The *text*, or a substring within *text*, can optionally be delimited by either single quotes (') or double quotes ("). These delimiter characters do not print; they specify that the text within the delimiters is to be printed exactly. Thus, to print a string containing one or more space characters as a single line, enclose the string in quote characters. For example: `"one line"`. To include one or more quote characters as literals in the printed string, enclose the string with the other type of delimiter character. For example: `"won't"`.

The **BLOCK.PRINT** command outputs to the current printer. The **BLOCK.TERM** command outputs to the current terminal. These commands are otherwise identical.

See Also: BLOCK.TERM

1.12 BLOCK.TERM

The **BLOCK.TERM** command displays a text as large-format letters.

```
BLOCK.TERM text
```

You can specify any printable character(s) as *text*, including a text containing blank spaces; no enclosing quotes are required. **BLOCK.TERM** uses multiple “X” characters to represent each character of *text* as a large block character. By default, it inserts a line break each time it encounters a space character or a string of space characters. **BLOCK.TERM** returns an error message if a string of characters is too long to display on one line; by default this maximum string length is 10 characters. The *text* can be of any length, provided no substring not containing a space character is greater than 10 characters.

A *text* does not require delimiters. The *text*, or a substring within *text*, can optionally be delimited by either single quotes (') or double quotes ("). These delimiter characters do not display; they specify that the text within the delimiters is to be displayed exactly. Thus, to display a string containing one or more space characters as a single line, enclose the string in quote characters. For example: `"one line"`. To include one or more quote characters as literals in the displayed string, enclose the string with the other type of delimiter character. For example: `"won't"`.

The **BLOCK.TERM** command outputs to the current terminal. The **BLOCK.PRINT** command outputs to the current printer. These commands are otherwise identical.

See Also: BLOCK.PRINT

1.13 BREAK

The **BREAK** command (and its variants) enable or disable terminal keys that can pause program execution.

```
BREAK [ON | OFF]
BREAK.KEY.ON
BREAK.KEY.OFF
BREAK.KEY.ENABLE
ENABLE.BREAK.KEY
```

When **BREAK** is enabled (ON), the Interrupt, Suspend, and Quit keys will cause program execution to be suspended. When **BREAK** is disabled (OFF) these keys have no effect on program execution. **BREAK** is enabled by default. The same operation is performed by the MVBasic BREAK statement.

Issuing any of these statements increments or decrements a counter. Thus multiple BREAK OFF statements (of any type) must be reversed by an equal number of BREAK ON statements.

In jBASE emulation, these statements simply enable or disable (toggle) without maintaining a counter.

See Also: MVBasic BREAK statement.

1.14 BSELECT

The **BSELECT** command generates a select list of non-null items that satisfy the query criteria.

```
BSELECT [DICT] filename [field1 [field2 ...] [dict [dict2 ...] | ALL] [query] [TO
listnum] [(FPYZ]
```

BSELECT and **SELECT** are identical, except that **BSELECT** does not select null items; **SELECT** selects all items including null items. **BSELECT** copies non-null items selected from *filename* to a select list. If *filename* is not an existing file, **BSELECT** generates a [200] error. If *filename* is an empty file, **BSELECT** generates a [401] error and no select list is returned.

The optional DICT keyword specifies that *filename* is accessing a dictionary file; otherwise, the *filename* is assumed to be accessing a data file. If there are multiple defined data sections (data files), you can specify *filename* as `filename,datasection`. **BSELECT** can specify any valid Caché MultiValue SQL (CMQL) query.

The optional *field* arguments permit you to specify which DICT entries to select. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. If you omit the *field* argument, all DICT entries in *filename* are selected, or all DICT entries are selected that pass the condition tests in *query*. If no items pass the *query* condition test, **BSELECT** generates a [401] error and no select list is returned.

The optional *dict* arguments permit you to specify which DICT entries to select for each *field*. You can specify one or more *dict* entry names separated by blank spaces. DICT entry names are *not* enclosed with quote characters. If you omit the *dict* argument, only the @ID (VOC) dictionary entry for each *field* is selected. If you specify ALL, all the *dict* attribute values for each field are selected.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

You can use the TO clause to specify a numeric select list. Valid *listnum* values are 0 through 10. By default, **BSELECT** uses select list 0. An invalid *listnum* generates a [819] error.

The following are supported letter code options:

- (F suppresses “not on file” message generation. Because select lists are implemented as SQL joins that only return rows that are in both the select list and the file, Caché MultiValue compares each item in the list with the file; items that don’t match are added to the error list, unless suppressed using this option.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **BSELECT** operation.

Upon successful completion, **BSELECT** returns a message such as the following: `2 Items selected to list #0`. The first successful **BSELECT** to any select list sets the boolean flag **$MVV(210)** to 1. **$MVV(210)** remains set to 1 until explicitly reset. The $MVV special variable is described in the *Caché ObjectScript Reference*.

See Also: CLEARSELECT, NSELECT, QSELECT, SEARCH, SELECT, SSELECT

1.15 BUILD.INDEX

The **BUILD.INDEX** command builds (populates) either an index for a specified file attribute, or indices for all of the attributes of the file.

```
BUILD.INDEX filename attribute | ALL | *
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **BUILD.INDEX** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the index to be built is stored in a named data section.

You can specify a single *attribute*, or a series of *attribute*s, separated by blank spaces. You can use either the ALL keyword or the asterisk (*) to build all indices.

If the *attribute* is not specified, **BUILD.INDEX** returns a [211] message. If *attribute* is invalid, or there are no indices defined for this *filename*, **BUILD.INDEX** returns a [842] message. If the index has already been built, **BUILD.INDEX** overwrites the previous index data.

Before you can build an index, you must create the index for *filename* using **CREATE.INDEX**.

See Also: CREATE.INDEX, LIST.INDEX

1.16 CATALOG

The **CATALOG** command catalogs one or more compiled MVBasic programs, storing a pointer to each in the VOC as a verb.

```
CATALOG filename [itemspec] [(L | N | G | GA ]
```

CATALOG catalogs one or more compiled MVBasic programs in the VOC so that they can be executed by specifying just the item ID, the same as any other MultiValue command (verb). Before cataloging a program, you must compile it using the BASIC command. You can execute a compiled MVBasic program without cataloging it by using the RUN com-

mand. There are three ways to catalog a program: Local (L), Normal (N), or Global (G or GA). When you change a Local or Global cataloged MVBasic program you must recompile it; you do not have to re-catalog it.

If you **CATALOG** a compiled MVBasic file, the *filename* is listed in the VOC as a file (F1=F) and can be displayed using LISTF. The MVBasic *itemspec* programs are listed in the VOC as verbs (F1=V).

If the *filename* file doesn't exist **CATALOG** generates a [201] error message. If the specified *itemspec* item doesn't exist **CATALOG** generates a [41] error message. If the specified *itemspec* item has not been compiled **CATALOG** generates a [40] error message.

filename is the file to search for the *itemspec* MVBasic program(s). The *filename* is the name of an existing file, which is created as VOC F1=F entry. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the program is stored in a named data section of the file.

itemspec specifies one or more compiled MVBasic programs by item ID. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all compiled MVBasic programs in the file should be cataloged. If *itemspec* is omitted, **CATALOG** uses the active select list as the source for the list of item IDs. If there is no active select list, **CATALOG** issues a "Item Id:" user prompt.

If specified, a letter code option must be prefaced by an open parenthesis. Only one letter code option may be specified. The available letter codes are: L = Local Catalog, N = Normal Catalog, G = Global Catalog, GA = Global Catalog with Account Name. The default is Local Catalog.

- `(L`: Local Catalog adds an entry to the VOC and points to the program's object code. A Local Catalog operation will fail if *program* has already been cataloged with Normal Catalog.
- `(N`: Normal Catalog adds an entry to the VOC and points to *a copy* of the program's object code. Normal Catalog allows users to continue to run a copy of the original code while you edit and test the program code itself. Therefore, changes made to the original code do not affect the cataloged copy until the object code is re-cataloged. Users can continue to run the cataloged copy of the original compiled code by:
  - Running it from the Terminal command line using the cataloged name reference to the copy of the code
  - Running it from MVBasic, issuing an EXECUTE, PERFORM, or CHAIN of the cataloged name

  You can use ED to edit and BASIC to compile the original program without affecting users of the Normal Catalog version. You can use RUN to execute and debug the most recently compiled version.

  You can edit, compile, and debug via Studio without affecting users of the Normal Catalog version. Note that although compile via Studio does a Local Catalog by default, it does not replace the Normal Catalog VOC entry. Instead, it provides the following message: `WARNING : "PROG" already cataloged without L option - you must recatalog it to use updated code`. If you use #PRAGMA ROUTINENAME, that name is used for the .mvi and .obj code, and that is what is used by Studio to run and debug. Studio does not debug the copy of the original code created by a Normal Catalog. Thus one disadvantage of using Normal Catalog is that Studio does not see the catalogued copy, and therefore cannot debug that code, unless you directly open the *.mvi routine in attribute 2 of the VOC entry for the program.
- `(G`: Global Catalog adds an entry to the global catalog (not the VOC) and points to a copy of the program's object code in a location accessible for access from all accounts. Specifying Global Catalog removes all corresponding Local Catalog and Normal Catalog entries from the VOC.

- (GA: Global Catalog with Account Name adds a global catalog entry prefaced with the account name delimited by asterisks. Thus using (GA to catalog the program MYPROG while in account USER would create the global catalog entry *USER*MYPROG. Using (GA does not delete any earlier global catalog entry created using (G, nor does it remove Local Catalog and Normal Catalog entries from the VOC.

See Also: BASIC, DECATALOG, PRINT.CATALOG

1.17 CEMU

The **CEMU** command changes the emulation of the current account.

```
CEMU [emulation]
```

Specifying **CEMU** with no operand returns the current emulation. For example, “Emulation for account 'USER' is 'CACHE'.”

Specifying **CEMU** with an operand sets the current emulation for the current account, and returns a message. The available *emulation* values are: Cache, D3, IN2, INFORMATION, jBASE, MVBase, PICK, PIOpen, Prime, R83, POWER95, Reality, UDPICK (UniData running in PICK mode), Ultimate, UniData, and UniVerse. The *emulation* argument is not case-sensitive. Both “Prime” and “Information” *emulation* values set an emulation of “INFORMATION.” An *emulation* value of “Default” sets an emulation of “CACHE”. An invalid *emulation* value returns an 812 error.

CEMU sets emulation for the current account (namespace) only. **CEMU** cannot set emulation for the SYSPROG account. This account is always in CACHE emulation. Attempting to change SYSPROG emulation returns an [815] error message.

All Caché accounts are initialized to an *emulation* of Cache. However, once you have set the emulation for an account, this emulation is persistent. It is retained after the process that sets it terminates, and applies to all future processes accessing that account until explicitly changed. Restarting Caché *does not* revert the emulation setting.

You can determine the current emulation from MVBasic as an integer value. You can return the integer value for the current emulation using the SYSTEM(1001) and SYSTEM(1051) functions, as described in the *Caché MultiValue Basic Reference*. You can also specify emulation within MVBasic by using the $OPTIONS command, as described in the *Caché MultiValue Basic Reference*.

For further details, refer to the CEMU section of the *Operational Differences between MultiValue and Caché* manual.

See Also: CREATE.ACCOUNT, LOGTO

1.18 CENTURY.PIVOT

The **CENTURY.PIVOT** command specifies how two-digit year values are interpreted process-wide.

```
CENTURY.PIVOT [year]
```

Upon MV Shell initialization, the MV Shell displays a [150] message indicating the default two-digit year date range. By default, a two-digit year is interpreted as being in the range 1900 to 1999. Specifying **CENTURY.PIVOT** with no operand returns the current date range for two-digit years.

You can use **CENTURY.PIVOT** to set any hundred-year range for two-digit years. To set this date range, specify a four-digit *year* as the beginning year of the century range. This four-digit *year* must be between 1841 and 9900, inclusive. A [152] message is generated indicating the new two-digit year date range. An invalid *year* value generates an [801] error. Once set, a two-digit year date range applies to the current process for the duration of the current process, or until set again.

It applies to all accounts (namespaces). It continues to apply for the current process across quitting and re-invoking of the MultiValue Shell.

1.19 CHECK.DICT

The **CHECK.DICT** command checks for a field in a file's DICT file.

```
CHECK.DICT filename [itemspec]
```

CHECK.DICT checks one or more *item* names in the *filename* DICT file.

The *filename* can be specified using filespec syntax, as follows:

```
[accountname,]filename[,datasection]
```

CHECK.DICT always references the DICT file.

itemspec is DICT item, or a list of DICT items to be checked. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the DICT file should be checked. If *itemspec* is omitted, **CHECK.DICT** uses the active select list as the source of DICT item names to check.

As each item is accessed, **CHECK.DICT** updates the @RECORD special variable.

If all of the specified items are present, **CHECK.DICT** completes without a message. If specified items are not present, **CHECK.DICT** returns a [202] message for each item not found in DICT.

If you omit the *itemspec* argument with no active select list, **CHECK.DICT** prompts you for it with the `Item Id:` prompt. At this prompt you can specify multiple items separated by blank spaces.

You can use `LIST DICT filename` to list all of the fields in the DICT file.

See Also: COMPILE.DICT, LIST

1.20 CHECK.PROC

The **CHECK.PROC** command checks that a PROC can be successfully compiled.

```
CHECK.PROC filename [itemspec]
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **CHECK.PROC** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the PROC to be checked is stored in a named data section.

itemspec is the name of a PROC, or a list of PROCs to be checked. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all PROCs in the file should be checked. If *itemspec* is omitted, **CHECK.PROC** uses the active select list as the source of PROC names to check.

As each item is accessed, **CHECK.PROC** updates the @RECORD special variable.

CHECK.PROC checks one or more named PROCs in a file. If a named PROC can be compiled, **CHECK.PROC** compiles it. If named PROC cannot be compiled, **CHECK.PROC** returns an error message.

1.21 CHOOSE.TERM

The **CHOOSE.TERM** command allows you to choose a terminal type. It displays a list of supported terminal types, then prompts you to specify the desired terminal type.

```
CHOOSE.TERM
```

After listing all the supported terminal types, **CHOOSE.TERM** prompts you to select a terminal type by either specifying its name (Term Name), its number in the listing, or (in some cases) a Short Name (a single uppercase letter). You can specify a Term Name with any combination of letter case: if all letters are specified in lowercase the terminal type is set as specified; if one or more letters are specified in uppercase the terminal type is set in all uppercase. To exit without changing the current terminal type, press Enter at the prompt.

You can determine your current terminal type by calling the MVBasic SYSTEM(7) function or the TERM command. The **TERM** command can also be used to choose a terminal type and/or to change certain terminal and printer parameters. For further details, refer to the Terminal Output chapter of the *Caché MV Terminal Independence* manual, and the CHOOSE.TERM section of the *Operational Differences between MultiValue and Caché* manual.

See Also: COMPILE.TERM, TERM

1.22 CLEAR.CMQL.CACHE

The **CLEAR.CMQL.CACHE** command clears the query cache for the current account.

```
CLEAR.CMQL.CACHE [(I]
```

This command should only be used during code analysis to ensure that the query cache has been cleared. It should not be used as part of normal MultiValue execution. This command should never be executed when the current account is being actively used.

Issuing **CLEAR.CMQL.CACHE** displays a warning message and an acceptance prompt. To execute **CLEAR.CMQL.CACHE**, input “Y” at the prompt; to cancel **CLEAR.CMQL.CACHE** input “N” or just press Return.

The optional `(I` letter code causes **CLEAR.CMQL.CACHE** to clear the ITYPE cache as well. After using **CLEAR.CMQL.CACHE (I** you must recompile classes for files with indices based on I-types. Failure to do so results in index corruption.

1.23 CLEAR.FILE

The **CLEAR.FILE** command clears all data from a file.

```
CLEAR.FILE filename
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if you wish to clear only the data stored in a named data section.

If the *filename* is not specified, **CLEAR.FILE** returns a [200] message. If the *filename* is not valid, **CLEAR.FILE** returns a [201] message. If the *filename* is valid, **CLEAR.FILE** returns a [433] message indicating that the file has been cleared, even if there was no data in the file. If the file was indexed, **CLEAR.FILE** also returns a [1] message “indexes purged”.

See Also: DELETE

1.24 CLEAR.LOCKS

The **CLEAR.LOCKS** command clears locks that are held by the current process.

```
CLEAR.LOCK [lock]
```

Use the optional *lock* argument to specify a specific lock. The default (with no argument) releases all locks held by the process.

CLEAR.LOCKS releases process locks only. It does not release system locks, readu locks, file locks, or any other lock type. Use the Management Portal to release all types of locks.

Locks are established using the MVBasic LOCK command or by opening a sequential file. When establishing a lock, you can name it with a numeric value or with a quoted string name. List all current locks using **LIST.LOCKS**. Process locks appear in the **LIST.LOCKS** output as `LOCK` *name*. When clearing a lock, *lock* must correspond to the lock name. Lock names are case-sensitive. When specifying *lock*, quote characters are optional.

If the lock specified by *lock* is not a current lock, **CLEAR.LOCKS** performs no operation and completes without issuing an error.

See Also: LIST.LOCKS, MVBasic LOCK, MVBasic UNLOCK

1.25 CLEARDATA

The **CLEARDATA** command clears the data stack.

```
CLEARDATA
```

1.26 CLEARPROMPTS

The **CLEARPROMPTS** command clears the value established for an inline prompt.

```
CLEARPROMPTS
```

Once a value has been input for an inline prompt, that value is used for every instance of that prompt, unless a **CLEARPROMPTS** command is issued to clear the inline prompt value.

See Also: << ... >> (inline prompting)

1.27 CLEARSELECT

The **CLEARSELECT** command clears the contents of the specified select list.

```
CLEARSELECT {listnum | ALL}
```

CLEARSELECT clears the specified numbered select list, or clears all numbered select list by specifying the ALL keyword. A select list is specified as an integer from 0 through 10. This command is identical to the CLEARSELECT Caché MVBasic Command.

See Also: BSELECT, NSELECT, QSELECT, SEARCH, SELECT, SSELECT

1.28 CLR

The **CLR** command clears the screen and sets the cursor to the first line.

```
CLR
```

The **CLR** and **CS** commands are synonyms.

See Also: CS

1.29 COMO

The **COMO** command copies terminal output to a record in the &COMO& file.

```
COMO ON record [HUSH]
COMO OFF COMO LIST
COMO DELETE record | *
COMO SPOOL record
```

If you specify **COMO** with no arguments, it returns a series of prompts requesting an option keyword and the record name.

COMO ON creates the specified record in the &COMO& file. Caché stores the &COMO& file using the ^COMO global. The optional HUSH keyword suppresses terminal display. **COMO DELETE** deletes the specified &COMO& file record. **COMO DELETE *** deletes all the &COMO& file records.

You can use **LIST &COMO&** to list the records in the &COMO& file.

See Also: LIST

1.30 COMPILE.DICT

The **COMPILE.DICT** command compiles I-descriptors in dictionary records.

```
COMPILE.DICT filename [itemspec]
```

The *filename* can be specified using filespec syntax, as follows:

```
[accountname,]filename[,datasection]
```

COMPILE.DICT always references the DICT file.

itemspec is DICT item, or a list of DICT items to be compiled. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the DICT file should be compiled. If *itemspec* is omitted, **COMPILE.DICT** uses the active select list as the source of DICT item names to compile.

As each item is accessed, **COMPILE.DICT** updates the @RECORD special variable.

See Also: CHECK.DICT, ICOMP

1.31 COMPILE.TERM

The **COMPILE.TERM** command compiles terminal definitions.

```
COMPILE.TERM [[filename [item-list] | *] [(TV] ]
```

COMPILE.TERM, with no arguments, compiles all the items in the %MV.TERMDEFS file. **COMPILE.TERM filename** compiles all the items in the *filename* file. **COMPILE.TERM *** compiles all the items in the %MV.TERMDEFS file and adds/replaces the results in the %MV.TERMCMP file. It does not delete existing items from TERMCMP. If you want to

eliminate some entries from TERMCMP (for example, to eliminate them from the CHOOSE.TERM listing) you should manually DELETE them from TERMCMP (and optionally from TERMDEFS).

The (T letter code option performs a tree search on *filename*; this assumes that *filename* is a directory in UNIX®/UniVerse format. For example: COMPILE.TERM //C:/terminfo (T. Note the use of the // prefix to directly reference a directory. The (V letter code option performs the compile in verbose mode.

For further details, refer to the Terminal Definition chapter of the *Caché MV Terminal Independence* manual.

See Also: CHOOSE.TERM, TERM

1.32 CONTROL.CHARS

The **CONTROL.CHARS** command sets input filtering of control characters.

```
CONTROL.CHARS [(F[S] | (N[S]]
```

CONTROL.CHARS with no option returns the current control character filtering setting (ON or OFF).

- The (F letter code option filters out control characters from the data received by the MVBasic INPUT statement. This returns the status string "CONTROL filtering is ON". **CONTROL.CHARS (F** activates input filtering for the process; the MVBasic INPUTCTRL ON statement activates input filtering for the current program.
- The (N letter code option deactivates filtering of control characters. This returns the status string "CONTROL filtering is OFF". **CONTROL.CHARS (N** deactivates input filtering for the process; the MVBasic INPUTCTRL OFF statement deactivates input filtering for the current program.
- The (S letter code option suppresses the status string display when activating or deactivating input filtering of control characters.

1.33 COPY

The **COPY** command copies items from file to file. It has two distinct syntax forms: non-interactive and interactive:

Non-interactive format:

```
COPY FROM sourcefile [TO targetfile]
          {item1[,newname1] [item2[,newname2] [...]] | ALL}
          [DELETING | OVERWRITING | UPDATING | SQUAWK]
```

COPY copies the specified items from *sourcefile* to *targetfile*. You must specify one or more items, or specify the ALL keyword, which copies all items in *sourcefile*. If you omit the TO clause, **COPY** copies the item(s) from *sourcefile* to *sourcefile*. Both *sourcefile* and *targetfile* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. If there are multiple defined data sections, you can specify *sourcefile* and/or *targetfile* as `filename,datasection`. File names and item names are case-sensitive.

You can specify one or more items to copy, separating the items with a blank space. You can specify a new name for the copied item, associating the original *sourcefile* name and the new *targetfile* name with a comma.

As each item is accessed, **COPY** updates the @RECORD special variable.

You can specify one or more of the following keyword options. Keywords must be specified in all uppercase letters. The DELETING keyword specifies that the original item in *sourcefile* is deleted after it is copied. The OVERWRITING keyword specifies that the copy occurs even if the named item exists in *targetfile*; the existing item with that name is overwritten. The UPDATING keyword is the same as the OVERWRITING keyword. The SQUAWK keyword displays the names of the files and the settings of the keyword options when performing the copy operation.

If the specified *sourcefile* or *targetfile* is not valid, **COPY** returns a [201] message. If the specified *item* is not valid, **COPY** returns a [202] message. If the specified *item* already exists in *file* and OVERWRITING is not specified, **COPY** returns a [415] message.

Interactive format:

```
COPY sourcefile [item1 [item2 [item-n]][(D | O | U | S]
TO:[(destinationfile ]newname1[ newname2[ ...]]
```

If you omit the item(s), **COPY** prompts you with `Item ID:`. It then prompts you for the destination file with `TO:`. You can either:

1. Specify an item list,
2. Specify a destination file (preceded by an apostrophe (') and an item list, or
3. Press **Enter** to copy the items to the Terminal display.

The D, O, U, and S options correspond to the keywords DELETING, OVERWRITING, UPDATING, and SQUAWK. If you specify a *destinationfile* but no item list, the original names are used. If you specify fewer destination names than source names, COPY attempts to use the original names for the extra source items. If more destination item names than source item names are specified, the extras are ignored.

See Also: COPY.FILE, COPYI, COPYP

1.34 COPY.FILE

The **COPY.FILE** command copies a file.

```
COPY.FILE [FROM] sourcefile [TO targetfile] [(D]
```

COPY.FILE creates a new file (*targetfile*) and copies the contents of *sourcefile* to *targetfile*.

- If you specify TO *targetfile*, **COPY.FILE** copies *sourcefile* to *targetfile*. **COPY.FILE** creates the *targetfile*, then copies the contents of *sourcefile* to *targetfile*.
- If you specify TO and omit *targetfile*, **COPY.FILE** copies *sourcefile* to the VOC in the current account.
- If you omit both TO and *targetfile*, **COPY.FILE** prompts you to specify a *targetfile* with a `TO:` prompt. To copy *sourcefile* and assign it a new name, specify *targetfile* at the prompt. To copy *accountname,sourcefile* from a different account to the current account, retaining the same file name, press **Enter** at the `TO:` prompt.

For all invocations of **COPY.FILE** the *targetfile* must not be an existing file. Both *sourcefile* and *targetfile* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the file to be copied is stored in a named data section. Some emulations support single level files; if

sourcefile is a single level file, **COPY.FILE** copies both the dictionary and data information. See CREATE.FILE concerning single level files.

COPY.FILE assigns the *targetfile* name to the *targetfile* @ID dictionary item, rather than copying the *sourcefile* @ID value.

The FROM keyword is optional.

The (D letter code option deletes the *sourcefile* after performing the copy operation. This is used for renaming a file.

If *sourcefile* contains an index reference, **COPY.FILE** does not copy this index reference to *targetfile*.

By default, **COPY.FILE** creates a target file regardless of whether the Attribute 2 ^filename and Attribute 3 ^DICT.filename globals are already in use. You can modify the **COPY.FILE** verb to prevent this. Edit **COPY.FILE** to add an Attribute 5 value of "m" (lowercase m). Now attempting to create a target file when one of these globals is already in use generates a [448] error.

See Also: COPY

1.35 COPY.LIST

The **COPY.LIST** command copies a saved select list from the &SAVEDLISTS& file.

```
COPY.LIST [filename [listname] ]
```

The optional *filename* is the destination file where the select list is to be copied. If you omit *filename*, **COPY.LIST** prompts you for the destination file name. The optional *listname* is the name of an existing select list; the default is select list 0. The *listname* select list is saved in the &SAVEDLISTS& file. Caché stores this file using the ^SAVEDLISTS global.

See Also: DELETE.LIST

1.36 COPYI

The **COPYI** command is the non-interactive form of the **COPY** command. The syntax is identical, except that the FROM keyword is optional.

See Also: COPY

1.37 COPYP

The **COPYP** command is the interactive form of the **COPY** command.

See Also: COPY

1.38 COS

The **COS** command causes the statement following it to be interpreted as an ObjectScript command.

```
COS commandline
```

COS issues an ObjectScript command without exiting the MultiValue Shell. The *commandline* can be any valid ObjectScript command line. A *commandline* cannot be specified as a variable, nor can it be enclosed in quotation marks.

If *commandline* changes the current namespace (for example, by issuing a ZNSPACE command), the Caché MultiValue Shell restores the initial namespace upon exiting ObjectScript.

COS *should not* be followed by the ObjectScript **MV** MultiValue Shell invocation command. Nested MultiValue Shell invocations may cause unexpected problems.

You can use **SH** to issue an operating system command without exiting the MultiValue Shell.

The **COS** command is functionally identical to the **#**, and **[** commands.

See Also: #, [, SH

1.39 COUNT

The **COUNT** command counts the items that satisfy an SQL query.

```
COUNT [DICT] filename [field1 [field2 ...]] [query] [(PYZ]
```

COUNT returns an integer count of the *field* items found in *filename*, or the items selected from *filename* by *query*. The optional DICT keyword causes the command to count the DICT entries in the *filename* dictionary file; otherwise, *filename* is assumed to be a data file.

COUNT *filename* returns the total number of items in *filename*.

COUNT *filename* with *field* arguments returns the total number of specified *field* items that are found in *filename*. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. Only items that are found in *filename* are counted; items not found in *filename* are displayed as a “not found” message.

COUNT *filename query* returns the number of items in *filename* that fulfill the specified criteria. For example, `STAT VOC WITH F1="V"` returns the count of verbs (V) in the VOC file. The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code options:

- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **COUNT** operation.

COUNT provides part of the functionality of the **STAT** command. **STAT** can return results from an AVG or ENUM CMQL clause; **COUNT** cannot return values for these CMQL clauses. The similar **LIST** command with the DET-SUPP keyword also returns the total number of *filename* items.

See Also: LIST, SELECT, STAT

1.40 CREATE.ACCOUNT

The **CREATE.ACCOUNT** command creates an account (namespace).

```
CREATE.ACCOUNT account [emulation] [directory]
```

The *account* is the name to assign to the namespace and the account. An account name commonly consists of letters, numbers, and the percent (%), hyphen (-), and underscore (_) characters. (Other punctuation characters are allowed, but should be avoided.) Percent (%) should only be used as the first character; hyphen (-) and underscore (_) should not be used as the first character. Underscore (_) cannot be used as the last character.

The system creates a corresponding namespace, as follows:

1. Caché namespace names cannot begin with a number as the first character. If the account name begins with a number, Caché strips the leading number(s) from the *account* name to create the corresponding namespace name.
2. Caché reserves namespace names that begin with % . Therefore, if the account name begins with %, Caché strips the % from the *account* name to create the corresponding namespace name.
3. Caché namespace names cannot contain punctuation characters other than the hyphen (-) and underscore(_). If the account name contains other punctuation characters, Caché strips them from the *account* name to create the corresponding namespace name.
4. Caché converts the resulting name to all uppercase characters. Namespace names are not case-sensitive, but account names are case-sensitive.

If you have specified an account name with the same name as an existing namespace that has no corresponding account, or if you have specified an account name that differs from an existing account name only in capitalization or in characters removed during namespace conversion, the system creates a unique namespace name. It creates this unique namespace name by appending an underscore and a sequential number, starting with the "_1" suffix.

Note that the SYSPROG account corresponds to the %SYS namespace.

For details on the relationship between accounts and namespaces, and the naming conventions used for each, refer to "MV Accounts and Caché Namespaces" in *Operational Differences between MultiValue and Caché*.

If the specified account already exists, **CREATE.ACCOUNT** returns an [810] message. Upon successful completion, **CREATE.ACCOUNT** returns an [814] message.

The optional *emulation* argument specifies the MultiValue emulation type. The default is Cache.

The optional *directory* argument specifies the location in which to create the account database. Specify a fully-qualified pathname. The default is \Mgr\accountname in the Caché installation location.

Note: Account creation and deletion requires **%Admin_Manage** privileges. This is normally associated with the SYSPROG account, which is the %SYS namespace. For information on the Caché security model, see the Caché Security Administration Guide. For specific information on roles and privileges, please consult the chapters on Roles and Privileges and Permissions.

See Also: ATTACH.ACCOUNTS, CEMU, DELETE.ACCOUNT, LOGTO

1.41 CREATE.BFILE

The **CREATE.BFILE** command creates a MultiValue Basic (MVBasic) source code file.

```
CREATE.BFILE [DATA | DICT] filename[,datasection] [ANODE | INODE]
```

The *filename* is the name of the Basic source code file, which is created as a Caché global (^filename). Caché global names are case-sensitive. Creating a file also creates a VOC entry for the file.

By default, **CREATE.BFILE** creates both a data file and a dictionary file; that is, a file with a dictionary (DICT) and a data section. The optional DICT and DATA keywords enable you to specify the creation of just a dictionary file or just a data file. Creating a dictionary also places a default record @ID in the dictionary.

Creating a data file creates a default data section. You can specify *datasection* if you wish to create a named data section. The *datasection* argument can only be specified for a data file; attempting to specify it for a dictionary file (DICT keyword) returns a [424] message.

By default, **CREATE.BFILE** creates a file of type INODE (item node); both the data section and DICT are of type INODE. You can override this default by specifying ANODE (attribute node). When you create both a data file and a dictionary file, the ANODE keyword only applies to the data section; the DICT is still created as type INODE. To create a DICT of type ANODE you must specify both the DICT and ANODE keywords.

CREATE.BFILE is a specific application of the more general **CREATE.FILE** command.

See Also: CREATE.FILE, DELETE.FILE

1.42 CREATE.FILE

The **CREATE.FILE** command creates a file.

```
CREATE.FILE [DATA | DICT] filename[,datasection] [ANODE | INODE] [DIR directory]
[(RUX]
```

The *filename* is the name of the file to create. **CREATE.FILE** creates the file as a Caché globals (^filename and ^DICT.filename). Caché global names are case-sensitive. Creating a file also creates a VOC entry for the file. If *filename* already exists, **CREATE.FILE** generates a [413] error.

By default, **CREATE.FILE** creates both a data file and a dictionary file; that is, a file with a dictionary (DICT) and a data section. The optional DICT and DATA keywords enable you to specify the creation of just a dictionary file or just a data file. Creating a dictionary also places a default record @ID in the dictionary.

Creating a data file creates a default data section. You can specify *datasection* if you wish to create a named data section. The *datasection* argument can only be specified for a data file; attempting to specify it for a dictionary file (DICT keyword) returns a [424] message.

By default, **CREATE.FILE** creates a file of type INODE (item node); both the data section and DICT are of type INODE. You can override this default by specifying ANODE (attribute node). When you create both a data file and a dictionary file, the ANODE keyword only applies to the data section; the DICT is still created as type INODE. To create a DICT of type ANODE you must specify both the DICT and ANODE keywords.

The DIR keyword specifies that the data section is set to use the specified directory.

The `(R` letter code option allows you to create the file in an account other than your current account. If you specify `(R`, **CREATE.FILE** prompts you to specify an account name.

The (U letter code option specifies that item names added to the file are to be untranslated when recorded in the directory. Name translation converts punctuation characters into hexadecimal codes (for example an _ (underscore) in a name is represented by 5F). By default, item names are translated.

The (X letter code option specifies that item names added to the file will include the three-character extension .mvb. Item names with this extension are parsed as MVBasic program files. A file created with this letter code option has a VOC entry Attribute 6 value of "B" and "X".

You can create an &SAVELISTS& or &HOLD& file as an ANODE type file (attribute of an item node), as follows:

```
USER:DELETE.FILE &SAVEDLISTS&
USER:CREATE.FILE &SAVEDLISTS& ANODE
```

You can create an &SAVELISTS& file as a DIR type file in an account, as follows:

```
USER:DELETE.FILE &SAVEDLISTS&
USER:CREATE.FILE &SAVEDLISTS& DIR SAVEDLISTS
```

Caché stores the &SAVEDLISTS& file using the ^SAVEDLISTS global. If the specified *filename* already exists, **CREATE.FILE** returns a [413] message.

By default, **CREATE.FILE** creates a file regardless of whether the Attribute 2 ^filename and Attribute 3 ^DICT.filename globals are already in use. You can modify the **CREATE.FILE** verb to prevent this. Edit **CREATE.FILE** to add an Attribute 5 value of “m” (lowercase m). Now attempting to create a file when one of these globals is already in use generates a [448] error.

1.42.1 Emulation

In D3, IN2, jBASE, MVBase, Pick, Reality, R83, POWER95, and Ultimate emulations, the command CREATE.FILE DICT *filename* creates a single level file. This is a file in which the dictionary section and the data section are the same.

You can delete a single level file using either DELETE.FILE *filename* or DELETE.FILE DICT *filename*. You cannot delete a single level file using DELETE.FILE DATA *filename*.

See Also: CATALOG, CLEAR.FILE, CREATE.BFILE, DELETE.FILE

1.43 CREATE.INDEX

The **CREATE.INDEX** command creates an index on a file.

```
CREATE.INDEX filename indexfield [indexfield2 [...]]
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the created index is to be stored in a named data section. The *filename* must be an INODE file; you cannot create an index on a directory file or an ANODE file.

You can use *indexfield* parameters to specify one field or more than one field to index; If you specify multiple fields, a separate index is created on each. When **CREATE.INDEX** creates an index it returns an [866] message. If *indexfield* does not exist, it returns a [202] error. If *indexfield* cannot be indexed, it returns a [5] invalid format error. If *indexfield* has already been indexed, an ERROR #5805: ID key not unique error is generated.

The *indexfield* is the itemid of an item in the dictionary of *filename*. It can be any length up to the maximum subscript size (because it is used as a subscript in the global). Creating the index creates both a class property and an index name entry in the associated Caché class. The property is named by converting *indexfield* to a case-sensitive name by omitting punctuation and using an uppercase letter to indicate where the punctuation was removed. For example, the *indexfield* START.DATE would correspond to the property name StartDate. The index name is created by appending the string "`index`" to the *indexfield* property name, then truncating the resulting name at 31 characters. Therefore, an *indexfield* name must be unique within the first 26 characters.

CAUTION: When creating an index for a MultiValue file, it is strongly recommended that every unique attribute have a corresponding class property. Any field/attribute that does not have a corresponding class property will become empty when the **%Save()** method is invoked. (Attributes that are mere synonyms do not require a corresponding class property.)

CREATE.INDEX assigns SqlString(150) collation to both the index and the created *indexfield* property. However, if the property already exists, **CREATE.INDEX** assigns the index the same collation type as the *indexfield* property.

CREATE.INDEX creates a class with a property named ItemId, which describes the item id of the original MultiValue file. You can change the names of other properties in the generated class (assuming that you also change the name anywhere that the property is referenced by other properties, indices, or methods) but the ItemId property must be named ItemId. Otherwise subsequent **CREATE.INDEX** commands will fail and leave the class in an uncompileable state.

All properties created using **CREATE.INDEX** contain an MVAUTO parameter which is assigned the "I" letter code. For further MVAUTO details, refer to the PROTOCLASS chapter of this manual.

If the specified *indexfield* is composed entirely of virtual fields, **CREATE.INDEX** automatically creates an additional property called dummyAttribute, so that the resulting class has at least one real (storage) attribute. If **CREATE.INDEX** adds real attributes later, dummyAttribute is automatically deleted. If you manually add real attributes later, you must manually delete dummyAttribute and associated storage.

You do not need to use **CREATE.INDEX** to add indices to a class. You can define indexes by either using **CREATE.INDEX** or by manually defining indexes by editing the class definition for the file using Studio. But if you intend to use the indexes with MVBasic statements and functions (for example INDICES(), SELECTINDEX, BSCAN, OPENINDEX, and SELECT with the ATKEY clause) you must create them using the format that **CREATE.INDEX** generates. **CREATE.INDEX** does not create a new index if there is already an index with the same MVNAME class property attribute as the specified DICT item. If you manually define indexes in a class, it is preferable to avoid the use of **CREATE.INDEX** and **DELETE.INDEX** on that file to avoid any unintended deletions.

When you create an index, the system creates a MultiValue index global with the following format:

```
^I.filename("indexD1"," ABC","NAME")="ABC"
```

The first node is the index name, here `indexD1`, created by appending the string "`index`" to the property name D1. The result is truncated at 31 characters.

The second node is the index key, here with a prefixed space to maintain the SqlString collation that MultiValue expects. This index key is truncated at 150 characters.

The third node is the itemid of the item being indexed. This node's data is a copy of the index key, without collation modification or truncation. This is the value that is returned by MVBasic indexing statements. This itemid can be any length up to the maximum Caché global subscript size.

If you create an index on a file that contains data, you must populate the index using **BUILD.INDEX**. If you create an index on a file that is currently open, you must close and reopen the file for MultiValue to be aware of the index. This close/reopen is necessary to activate operations such as automatically updating the index when you perform a **WRITE**. These index activation steps are required for Caché MultiValue, UniVerse emulation, and jBASE emulation.

For further details, refer to the CREATE.INDEX section of the *Operational Differences between MultiValue and Caché* manual.

See Also: BUILD.INDEX, DELETE.INDEX, LIST.INDEX, the PROTOCLASS chapter of this manual.

1.44 CREATE.TRIGGER

The **CREATE.TRIGGER** command creates a trigger that calls a subroutine when an event of a specified type occurs.

```
CREATE.TRIGGER filename event subroutine [(AOT]
```

The *filename* is the name of an existing file. The trigger is specific to events occurring to this file. The *event* is the operation that invokes (pulls) the trigger. The trigger can be designed to be invoked before or after the execution of a variety of MVBasic commands. It can be one of the following: *, POSTOPEN, PREREAD, PREINSERT, PREUPDATE, PREWRITE, PREDELETE, PRECLEAR (or PRECLEARFILE), POSTREAD, POSTINSERT, POSTUPDATE, POSTWRITE, POST-DELETE, POSTCLEAR (or POSTCLEARFILE). To invoke the trigger upon any of the event types, specify an asterisk (*) as the *event* value.

The trigger code that is executed when the trigger is pulled is located in *subroutine*, which is an MVBasic subroutine.

CREATE.TRIGGER supports the following letter code options:

- (A allows the trigger to alter the record created or updated by the event operation before the record is saved. This is a trigger execution option.
- (O allows **CREATE.TRIGGER** to overwrite an existing definition of this trigger (if one exists). This option has no effect on trigger execution.
- (T allows a pre-event trigger to terminate (abort) the event operation before it occurs. This is a trigger execution option.

Caché MultiValue also supports UNIX-style option syntax: a hyphen followed by the option as a lowercase letter, as shown in the following example:

```
CREATE.TRIGGER -a TestFile POSTREAD TriggerSub
```

Specifying an invalid *filename* returns a [201] message. Specifying a *subroutine* that has not yet been cataloged returns a [825] message.

The *subroutine* syntax is as follows:

```
SUBROUTINE triggersub(filename,eventnum,prerc,flags,recordkey,record,userrc)
```

Within the trigger handler *subroutine* the event value is passed as an integer code, not a keyword. The *eventnum* integer codes that correspond to *event* keywords are as follows: POSTOPEN=1, PREREAD=2, POSTREAD=3, PREDELETE=4, POSTDELETE=5, PRECLEAR=6, POSTCLEAR=7, PREWRITE=8, POSTWRITE=9, PREINSERT=10, POSTINSERT=11, PREUPDATE=12, POSTUPDATE=13. Note that an INSERT or UPDATE event is also processed as a corresponding WRITE event; thus the user can handle the event as an INSERT or UPDATE, as a WRITE, or both.

prerc is the status return code for the action performed by the subroutine. If the event is a PRE event, the *prerc* is always 0; if the event is a POST event, *prerc* is 0 if the action was successful, non-zero if the action failed. *flags* is a no-op. *recordkey* is the item-id of the record being written or deleted; *recordkey* is null for an OPEN or CLEAR operation. *record* is the record currently being written; *record* is null for OPEN, DELETE, and CLEAR operations; *record* is null for PREREAD, assigned a value for POSTREAD. *userrc* is a user-defined status return code used only with the (T option. If (T is specified, a non-zero value for *userrc* causes a PRE event trigger to terminate (abort) the event operation. A negative integer *userrc* aborts the event operation without invoking the event handler; a positive integer *userrc* aborts the event operation and is passed as an error code to the error handler.

When an MVBasic READV statement is executed on a file with a POSTREAD trigger, the entire contents of the file record being read are passed to the trigger routine, not just the value of the field specified in READV.

CREATE.TRIGGER creates a trigger wrapper routine, which it names MVTW followed by the file name. If the file name is greater than 22 characters, **CREATE.TRIGGER** truncates it to 22 characters then adds the checksum integer for the full file name (see the ObjectScript **$ZCRC** function).

Note that Caché MultiValue triggers are completely separate from Caché SQL triggers. An SQL update will not fire a MultiValue trigger; a MultiValue update will not fire an SQL trigger.

Note: Caché stores pointers to trigger code in the data file header. When a file is opened, that information is stored in the file pointer. If a trigger is added or deleted while the file is open, the system attempts to follow the trigger definitions that existed when the file was opened. This could result in a new trigger not firing, or an attempt to fire a deleted trigger failing and rejecting the WRITE. Therefore, triggers should not be maintained while the corresponding data file is open.

See Also: DELETE.TRIGGER, LIST.TRIGGER

1.45 CS

The **CS** command clears the screen and sets the cursor to the first line.

```
CS
```

The **CS** and **CLR** commands are synonyms.

See Also: CLR

1.46 CT

The **CT** command displays one or more records on the terminal screen.

```
CT filename [itemspec] [(P]
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. If there are multiple defined data sections, you can specify *filename* as `filename,datasection`.

itemspec is the name of a record, or a list of records, to display. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all records in the file should be displayed. If *itemspec* is omitted, **CT** uses the active select list as the source of record names to display. If you omit the *itemspec* argument and there is no active select list, **CT** prompts you for items with the `Item Id:` prompt. At this prompt you can specify multiple *field* items, separated by blank spaces.

As each item is accessed, **CT** updates the @RECORD special variable.

The `(P` letter code option redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.

See Also: LIST.ITEM

1.47 DATE

The **DATE** command returns the current local date and time.

```
DATE [(P]
```

For example: “Tuesday, October 23, 2007 02:55pm”, The actual date and time format is governed by the **DATE.FORMAT** command.

Caché MultiValue determines local time and date as follows:

- It determines the current Coordinated Universal Time (UTC) from the system clock.
- It adjusts UTC to the local time zone by using the value of the Caché special variable $ZTIMEZONE.
- It applies local time variant settings (such as Daylight Saving Time) for that time zone from the host operating system.

The optional `(P` letter code option redirects output to the STANDARD print queue. You can use **LISTPEQS** to view the print queue.

See Also: DATE.FORMAT

1.48 DATE.FORMAT

The **DATE.FORMAT** command specifies the format used for displaying dates.

```
DATE.FORMAT [ON | OFF] [(I | (D]
DATE.FORMAT INFORM
```

The optional ON keyword specifies international date order; for example: “Tuesday, 23 October 2007 02:55pm”, The optional OFF keyword specifies USA date order; for example: “Tuesday, October 23, 2007 02:55pm”, The default is ON. The `(I` and `(D` letter code options are equivalent to the ON and OFF arguments; the opening parenthesis is mandatory. Specify either ON or OFF or `(I` or `(D`, not both.

The INFORM option sets @SYSTEM.RETURN.CODE to the current date format setting: 0 (USA date order) or 1 (international date order).

For further details, refer to the DATE.FORMAT section of the *Operational Differences between MultiValue and Caché* manual.

See Also: DATE

1.49 DECATALOG

The **DECATALOG** command removes one or more cataloged programs from the VOC.

```
DECATALOG filename [itemspec] [(AGLNV]
```

DECATALOG may be run against the VOC or against a file of MVBasic source code. When run against the VOC, it deletes the specified VOC item, as well as any copies generated for normal catalog. If run against a source file, it deletes

any catalog pointers for the specified file, as well as normal and global catalog copies. If you specify the A *option*, it also deletes the compilation object code.

filename is the file to search for the MVBasic program. The *filename* is the name of an existing file, which is created as a Caché global (^filename). The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the cataloged program is stored in a named data section of the file.

itemspec is the name of the program (or programs) to decatalog. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all programs in the file should be decataloged. If *itemspec* is omitted, **DECATALOG** uses the active select list as the source of program names. If there is no active select list, **DECATALOG** issues a "Item Id:" user prompt. Upon successful completion **DECATALOG** lists the item IDs of the MVBasic programs decataloged.

If the *filename* file doesn't exist **DECATALOG** generates a [201] error message. If you specify an MVBasic file as *filename* and the specified *itemspec* item doesn't exist in that file **DECATALOG** generates a [202] error message. If you specify VOC as *filename* and the specified *itemspec* item isn't currently catalogued in the VOC, **DECATALOG** generates a [202] error message. However, if the specified *itemspec* item has already been decataloged from an MVBasic file, **DECATALOG** completes without error.

If specified, a letter code option must be prefaced by an open parenthesis. The available letter codes are: `(A` = delete all, `(G` = Global Catalog, `(L` = Local Catalog, `(N` = Normal Catalog, and `(V` = verbose mode. If you specify the `(G` letter code, **DECATALOG** looks first in the global catalog for the specified *itemspec* programName, and deletes it if found. If not found, **DECATALOG** then looks in the global catalog for *currentaccount*programName and deletes it. For further details on G, L, and N letter code options, refer to the **CATALOG** command. The default is Local Catalog.

See Also: CATALOG

1.50 DELETE

The **DELETE** command deletes one or more items from a file.

```
DELETE filename [itemspec]
```

DELETE deletes the source code for one or more items. It does not delete the item's compiled object code or remove the item's catalog listing. Therefore, it is possible to execute a MVBasic program after you have deleted it.

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **DELETE** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the item to be deleted is stored in a named data section.

itemspec specifies an item, or a list of items, to delete. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the file should be deleted. If *itemspec* is omitted, **DELETE** uses the active select list as the source of item names to delete. If there is no active select list, **DELETE** issues a "Item Id:" user prompt.

If *item* is not present in the file, **DELETE** generates a [202] error message. If no items are deleted [430] is returned. If one item is deleted [431] is returned. If more than one item is deleted [432] is returned.

See Also: CLEAR.FILE, DELETE.FILE

1.51 DELETE.ACCOUNT

The **DELETE.ACCOUNT** command deletes an account (namespace) and all of the MultiValue files within it.

```
DELETE.ACCOUNT account
```

account is an existing MultiValue account name, which has been assigned to a corresponding Caché namespace. **DELETE.ACCOUNT** deletes all of the MultiValue globals and routines found in *account*, and, if the namespace is empty, deletes *account*. **DELETE.ACCOUNT** *does not* delete globals or routines in the namespace that were not created by MultiValue operations. If the account is not associated with a namespace (for example, a synonym account) an error message is returned.

This is a restricted command. You must be logged in to the SYSPROG account to delete an account; use the **LOGTO** command to log in to SYSPROG. You cannot delete the SYSPROG account, which is the %SYS namespace. Attempting to do so returns a [196] error message.

Note: Account creation and deletion requires **%Admin_Manage** privileges. This is normally associated with the SYSPROG account (the %SYS namespace). For information on the Caché security model, see the Caché Security Administration Guide. For specific information on roles and privileges, please consult the chapters on Roles and Privileges and Permissions.

See Also: CREATE.ACCOUNT, LOGTO

1.52 DELETE.FILE

The **DELETE.FILE** command deletes a file and its VOC entry.

```
DELETE.FILE [DATA | DICT] [accountname,]filename[,datasection]
```

The *filename* is the name of an existing file, which was created as a Caché global (^filename) by **CREATE.FILE** or **CREATE.BFILE**. The file may consist of a dictionary (DICT) and a data section, or be only a data file or only a dictionary file. By default, **DELETE.FILE** deletes both the dictionary and the data section, and deletes the corresponding VOC entry.

The optional DICT and DATA keywords enable you delete just the dictionary (DICT) or just the data (DATA) section. These keywords also update the corresponding VOC entry. Unlike most file commands, the default is to delete both DICT and DATA.

By default, **DELETE.FILE** deletes the data section regardless of whether it is the default data section or a named data section. If you specify the DATA keyword, you must specify *datasection* to delete a named data section. If you specify *datasection*, **DELETE.FILE** only deletes the file if it has a corresponding named data section. Otherwise it returns a [206] error. To explicitly delete a default data section, specify *filename*,*filename*. You cannot specify both the DICT keyword and a *datasection*; doing so returns a [424] error.

You can use *accountname* to delete a file in an account other than the current account.

If *filename* is a single level file, you can delete it using either `DELETE.FILE filename` or `DELETE.FILE DICT filename`. You cannot delete a single level file using `DELETE.FILE DATA filename`. Refer to CREATE.FILE for further details.

See Also: CLEAR.FILE, CREATE.BFILE, CREATE.FILE, DELETE

1.53 DELETE.INDEX

The **DELETE.INDEX** command deletes one or more indexes on a file.

```
DELETE.INDEX filename {indexfield [indexfield2] [...] | * | ALL}
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **DELETE.INDEX** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the index to be deleted is stored in a named data section.

The *indexfield* is the name of a field used as a secondary index key. Specifying *indexfield* deletes the index named `index`*indexfield*. **DELETE.INDEX** matches *indexfield* by searching the following (in this order): an index of that name; a property having the MVNAME parameter of that name; a property of that name. If the *indexfield* field cannot be matched or has not been indexed, **DELETE.INDEX** returns a [212] error message.

You can specify multiple *indexfield* indexes separated by blank spaces. You can delete all indexes for *filename* by specifying either an asterisk (*) or the ALL keyword. If you do not specify either *indexfield*, *, or ALL, **DELETE.INDEX** returns a [211] error message.

You can define indexes either by using **CREATE.INDEX** or by manually defining them in the class definition for the file. **DELETE.INDEX** can delete indexes by *indexfield* name, MVNAME, or property name. However, if you manually define indexes in a class, it is preferable to avoid the use of **DELETE.INDEX** on that file to avoid any unintended deletions.

If you delete an index on a file that is currently open, you must close and reopen the file for MultiValue to be aware of the index deletion. This close/reopen is required for Caché MultiValue, UniVerse emulation, and jBASE emulation.

For further details, refer to the CREATE.INDEX section of the *Operational Differences between MultiValue and Caché* manual.

See Also: CREATE.INDEX, LIST.INDEX

1.54 DELETE.LIST

The **DELETE.LIST** command deletes a select list from the &SAVEDLISTS& file.

```
DELETE.LIST listname | *
```

The *listname* is the name of an existing select list in &SAVEDLISTS&. The asterisk (*) argument deletes all select lists in &SAVEDLISTS&. Caché stores the &SAVEDLISTS& file using the ^SAVEDLISTS global.

See Also: COPY.LIST

1.55 DELETE.TRIGGER

The **DELETE.TRIGGER** command deletes a specified type of trigger from a file. It removes the trigger specification, not the trigger subroutine itself.

```
DELETE.TRIGGER filename event
```

The *filename* is the name of an existing file. The *event* is one of the following: *, POSTOPEN, PREREAD, PREINSERT, PREUPDATE, PREWRITE, PREDELETE, PRECLEAR, POSTREAD, POSTINSERT, POSTUPDATE, POSTWRITE, POSTDELETE, POSTCLEAR. To delete all triggers of any type, specify an asterisk (*) as the *event* value.

Specifying a valid *filename* and *event* returns a [828] message, indicating that the trigger has been deleted, even if the specified trigger does not exist. Specifying an invalid *filename* returns a [201] message. Specifying an invalid *event* returns a [824] message.

Note: Caché stores pointers to trigger code in the data file header. When a file is opened, that information is stored in the file pointer. If a trigger is added or deleted while the file is open, the system attempts to follow the trigger definitions that existed when the file was opened. This usually results in a new trigger not firing..

See Also: CREATE.TRIGGER

1.56 DISPLAY

The **DISPLAY** command displays a line of text on the terminal screen.

```
DISPLAY [text]
```

The *text* is displayed exactly as specified. If *text* is enclosed with delimiters, these delimiters are displayed as part of the text. No parsing of expressions or conversion of numbers to canonical form is performed. If *text* is omitted, this command displays the empty string (a blank line).

1.57 DOS

On Windows systems, the **DOS** command executes a Windows DOS command.

```
DOS [commandline]
DOS -c "commandline"
```

The **DOS** command enters the DOS command prompt environment without exiting the MultiValue Shell. Results are displayed to the MultiValue Shell. No additional windows are opened.

DOS with no argument opens an interactive command prompt from which you can issue multiple DOS commands. To exit this command prompt and return to the MultiValue Shell, issue the Windows EXIT command.

DOS commandline issues a Windows command as a background process. The *commandline* can be any valid Windows command line. A *commandline* cannot be specified as a variable, nor can it be enclosed in quotation marks. A *commandline* cannot exceed 248 characters in length. Upon completion it returns to the MultiValue Shell prompt.

The **DOS -c "commandline"** alternate syntax is equivalent to **DOS commandline**. This syntax is provided for UniVerse compatibility. The double quotes enclosing the *commandline* are mandatory.

The **SH** command is similar to **DOS**, but can also be issued in non-Windows environments. On Windows systems, the **DOS** and **SH** commands are synonyms.

On Windows systems you can use **DOS set** (or **SH set**) to display a list of environment variables. To display individual environment variables, you can use the MVBasic GETENV() function.

See Also: COS, SH

1.58 ED

The **ED** command allows you to edit a record in a file.

```
ED filename itemspec
```

ED is the MultiValue line editor. Caché MultiValue supports the **ED** and **JED** editors.

The *filename* is the name of the file to edit, which is created as a Caché global (^filename). If the specified *filename* is not valid, **ED** returns a [201] message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the program to be edited is stored in a named data section of the file.

itemspec specifies a record, or a list of records, to edit. If you specify multiple records, **ED** starts with the first item you specified. When you exit editing of that item, **ED** accesses the next item in the specified order. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all records in the file should be edited. If *itemspec* is omitted, **ED** uses the active select list as the source of record names to edit.

If *item* is an existing record, **ED** positions the current line pointer to the beginning of this record. If *item* is not an existing record, **ED** creates the specified record.

When **ED** creates or accesses an *item* it updates the @RECORD special variable.

The **ED** command provides a prompt for editing the current record. You can issue a variety of subcommands at the **ED** prompt. For a list of these subcommands, specify HELP at the **ED** prompt. For a list of the attributes of the specified *record*, specify ? at the **ED** prompt.

The following is a list of the **ED** subcommands supported by Caché:

```
-nnn            Decrement current line pointer.
+nnn            Increment current line pointer.
?               Display information about the record.
^               Toggle the up-arrow display mode.
<               Set marker for the FROM lines.
>               Set marker for the THROUGH lines.
! command       Execute a MV shell command.
# command       Execute a COS shell command.
nnn             Set current line pointer.
A text          Append text to end of line.
B               Set current line pointer to Bottom of record.
B string        Break line at position of string.
BLOCK           Toggle the BLOCK verification.
C               Repeat change/replace.
C/from/to/[Gnn]   Change string 'from' with 'to'.
COPY            Copy a BLOCK of lines.
D[nn]           Delete one or more lines of text.
```

```
DE[nn]           Delete one or more lines of text.
DELETE             Delete the record from the file.
DROP              Drop (delete) a BLOCK of lines.
EX[KO]            Exit editing this record.
FD[KO]            Delete the record from the file.
F[[col] string]   Find a string at a specified column.
FI[K]            File record and exit editing this record.
FILE              File record and exit editing this record.
FORMAT [n[m]]     Format BASIC code, tab stop n , initial level m.
FS               File record and continue editing this record.
G[nnn]            Go to line nnn, or next line.
G<               Go to beginning of the current BLOCK.
G>               Go to end of the current BLOCK.
HELP               Display help screen.
HEX               Toggle HEX input/output mode.
I[string]         Insert mode or insert string at current line pointer.
IB[string]         Insert before mode or insert string before current line.
L                Repeat locate (or list one line if no previous locate).
L string           Locate next occurrence of string.
Lnnn               List the next nnn lines.
Lnnn string        Locate all strings in the next nnn lines.
LOAD [file] id     Load lines of code from another item.
MOVE               Move a BLOCK of lines.
OOPS               Undo the previous command.
P[n]              Execute prestore command n.
P[n] Cmd1[<ESC>Cmd2] Set prestore command n.
PP[nn]             Page Print for nn lines.
Q                Exit editing this record.
QUIT               Exit editing this record.
R                Repeat replace.
R newtext          Replace entire line with 'newtext'.
R/from/to/[Gnn]   Replace string 'from' with 'to'.
SIZE               Display information about size of record.
TB n[,n...]       Set tab stop position.
T                Set current line pointer to Top of record.
U[nn]              Move up nn lines.
UNDO               Undo the previous command.
UNLOAD [file] id  Save lines of code to another item.
X                Exit editing this record and return to command line.
XEQ command        Execute a MV shell command.
```

Additional notes on subcommands:

- DE: When you delete a line of code, the current line pointer moves upward. Following a line delete, ED displays the current line. As this current line pointer behavior is not consistent across all MultiValue systems, exercise caution when performing repeated line deletions.
- F: This subcommand finds the first instance of the specified *string* occurring at the specified *col* (column number). Note that there is no space between F and the column number (for example, `F4 bscan`). If *col* is omitted, the default is column number 1. If *col* and *string* are omitted, the previous F command is repeated.
- FIBCR: This subcommand performs a series of operations. It (FI) files the item, (B) compiles the Basic source, (C) catalogs the object code, and finally (R) runs the command. If any one of these steps fails, the subsequent steps are not attempted.
- HEX: A mode toggle to turn on or off display of text in hexadecimal. HEX mode only affects character display and the I (insert) subcommand; it does not affect other subcommands.
- I: To insert a blank line, specify the I subcommand followed by a blank space. To insert a hexadecimal value, go into HEX mode. To insert an @VM character (CHAR(253)) specify **Ctrl-]**. To insert an @SM character (CHAR(252)) specify **Ctrl-**.
- I and R: You can use the ^ (caret) code operator in an insert (I) or replace (R) subcommand to specify a single character by its integer code (values 000 through 255). This is commonly used for non-printable control code characters. This can be used to specify a MultiValue delimiter, such as ^253 for a value mark. However, you cannot specify a ^254, because this specifies a field mark. This use of ^ is disabled by default, and must be enabled.
- P: The Prestore subcommand enables you to assign a single-digit integer to a series of one or more commands. The format is P# command1<ESC>command2<ESC>command3 and so forth. The # is the assigned single-digit integer value, followed by an associated command. Additional commands can be added by separating the commands with an Escape character (<ESC>). This Escape character typed using **ED** is echoed (displayed) as "[".

An Escape character typed using **ED** is echoed (displayed) as "[".

You can use the Up and Down arrow keys to scroll through the **ED** subcommands history for the current editing session.

To abort **ED** and release all locks, use **Ctrl-C**. Any time you exit **ED**, it clears select list 0.

ED is a simple line editor, a subset of the line editors supplied with other MultiValue systems. For more complex editing, the user should use the Studio.

The **ED** and **EDIT** commands are synonyms.

See Also: CREATE.FILE, EDIT, EDIT.LIST, JED

1.59 EDIT

The **EDIT** command allows you to edit a record in a file.

```
EDIT filename itemspec
```

The **EDIT** and **ED** commands are synonyms.

See Also: ED

1.60 EDIT.LIST

The **EDIT.LIST** command allows you to edit a select list in &SAVEDLISTS&.

```
EDIT.LIST [listname]
```

EDIT.LIST allows you to create or modify a select list, using the Caché MultiValue command line editor (**ED**) prompts. **EDIT.LIST** is the same as **ED &SAVEDLISTS& recID**, where *recID* is the record ID of a saved list. If you do not specify a *listname*, you are prompted to supply one. Caché stores the &SAVEDLISTS& file using the ^SAVEDLISTS global.

See Also: ED

1.61 ENABLE.BREAK.KEY

The **ENABLE.BREAK.KEY** and **BREAK ON** commands are functionally identical.

1.62 FORM.LIST

The **FORM.LIST** command allows you to create a select list from elements stored in a record.

```
FORM.LIST [filename] [recID] TO [listnum]
```

FORM.LIST take the record identified by *recID* from *filename*, and uses it to create a numbered select list *listnum*. If you omit *filename* or *recID* you are prompted to supply one. Select lists are numbered 0 through 10; if the TO *listnum* argument is omitted, it defaults to select list 0.

See Also: GET.LIST

1.63 GET.LIST

The **GET.LIST** command copies the specified named select list into a numbered select list.

```
GET.LIST listname [TO n]
```

The *listname* argument specifies a named select list. The optional TO *n* argument is a select list number in the range 0 through 10 (inclusive); if omitted, select list 0 is the default.

If *n* is out of range, **GET.LIST** returns a [209] message.

GET.LIST is the inverse of **SAVE.LIST**. When a select list is saved it becomes unavailable to the MVBasic READNEXT command. To make a select list available again, use **GET.LIST**.

To copy Select List 0 to another numbered select list, use the **PQ.SELECT** command.

See Also: PQ.SELECT, SAVE.LIST

1.64 HUSH

The **HUSH** command suppresses terminal screen display. It suppresses all terminal display, including displaying the terminal prompt.

```
HUSH [ON | OFF]
```

Specifying **HUSH** with no operand toggles display suppression. **HUSH ON** suppresses display. **HUSH OFF** re-enables display.

The **KEYS** command temporarily overrides the **HUSH**. However, when **KEYS** times out, it returns to the prior **HUSH** mode. This may be mistaken for a hang state.

The **HUSH** and **P** commands are synonyms.

See Also: DISPLAY, P

1.65 ICOMP

The **ICOMP** command compiles the I-type dictionary definitions in the specified file.

```
ICOMP [DATA] filename [itemspec]
```

ICOMP compiles the I-type dictionary definitions in *filename*. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

ICOMP by default references the DICT file. If you wish to compile I-types in the data portion of a file, specify the optional DATA keyword. This is most commonly used when compiling the VOC. The default is to compile in the dictionary portion.

itemspec is the name of an I-type item, or a list of items, to compile. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all I-type items in the file should be compiled. If *itemspec* is omitted, **ICOMP** uses the active select list as the source of I-type item names.

As each item is accessed, **ICOMP** updates the @RECORD special variable.

See Also: COMPILE.DICT, ICOMP.ALL

1.66 ICOMP.ALL

The **ICOMP.ALL** command compiles all the I-type dictionary definitions in one or more accounts.

```
ICOMP.ALL [accountname] [(AV]
```

ICOMP.ALL with no arguments compiles all the I-type dictionary definitions in the current account (namespace). **ICOMP.ALL** with the optional *accountname* argument compiles all the I-type dictionary definitions in the specified account (namespace). **ICOMP.ALL** with the `(A` letter code option compiles all the I-type dictionary definitions in all accounts (namespaces). It lists the accounts as it performs the compile. If you specify both *accountname* and the `(A` option, the *accountname* is ignored and all accounts are compiled. The `(V` letter code option returns verbose output while the compile operation executes.

ICOMP.ALL continues to compile I-types despite errors; it reports successful compiles and errors as they occur. Upon successful compile **ICOMP.ALL** generates a [7140] message specifying how many I-types compiled without error. If errors occurred during the compile of an I-type file, it generates a [7141] error message, listing the number of I-types compiled, the number of errors, and the file containing the I-types. If a dictionary item contains a syntax error, **ICOMP.ALL** generates a [7107] error message.

See Also: ICOMP

1.67 JED

The **JED** command allows you to edit a record in a file.

```
JED filename [item]
```

The **JED** command provides a full-screen editor similar to the editor supplied with jBASE. Caché MultiValue supports the **ED** and **JED** editors.

Note: **ED** is the preferred editor for Caché MultiValue. **JED** is provided for compatibility only.

The *filename* is the name of the file to edit, which is created as a Caché global (^filename). If the specified *filename* is not valid, **JED** returns a [201] message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the program to be edited is stored in a named data section of the file.

If *item* is an existing record, **JED** positions the current line pointer to the beginning of this record. If *item* is not an existing record, **JED** creates the specified record. If *item* is omitted, **JED** prompts you for the item name.

When **JED** creates or accesses an *item* it updates the @RECORD special variable.

See Also: ED

1.68 JOBS

The **JOBS** command list all running phantom processes initiated by the current process.

```
JOBS
```

JOBS lists all running processes initiated by the current process; **LIST.JOB** lists all running processes. You can use the **PHANTOM** command to initiate a phantom process.

See Also: LIST.JOB, PHANTOM

1.69 KEYS

The **KEYS** command sets the terminal into a mode in which each keyboard input character is displayed, along with its hexadecimal and ASCII base-10 equivalents. Compound keys (for example, the F1 key) return their component characters, one line per character. The **KEYS** command is designed to display all characters, including those that would normally terminate input. For this reason, the only way to exit **KEYS** mode is by timing out. This mode terminates automatically after 10 seconds of inactivity.

```
KEYS
```

The **KEYS** mode overrides the **HUSH** mode. However, when **KEYS** times out, it returns to the prior **HUSH** mode. This may be mistaken for a hang state.

For further details, refer to the Terminal Input chapter of the *Caché MV Terminal Independence* manual.

1.70 LIST

The **LIST** command returns a list of items that satisfy the query criteria.

```
LIST [DICT] filename [field1 [field2 ...]] [dict [dict2 ...] | ALL] [query]
[(CDEFHINPYZ]
```

LIST returns a formatted display listing the items selected from a file. At the end of the listing the total number of items listed is returned. **LIST** is a Caché MultiValue SQL (CMQL) query command.

The optional DICT keyword causes the command to access the DICT entries in the *filename* dictionary file; otherwise, *filename* is assumed to be a data file. If there are multiple defined data sections (data files), you can specify *filename* as `filename,datasection`. The DICT keyword and *field* arguments are mutually exclusive.

The optional *field* arguments permit you to specify which data fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. Item IDs correspond to the @ID (VOC) dictionary entry value. If you omit the *field* argument, all fields in *filename* are listed. By default, **LIST** lists *field* items in the order that you specify them. By default, the **SORT** command lists *field* items in ascending collation sequence.

The optional *dict* arguments permit you to specify which DICT entries to list for each *field*. You can specify one or more *dict* entry names separated by blank spaces. DICT entry names are *not* enclosed with quote characters. If you omit the *dict* argument, only the @ID (VOC) dictionary entry for each *field* is listed.

If you specify ALL, all the *dict* attribute values for each field are listed; this is similar to **LIST.ITEM**. In all cases, the @ID (VOC) dictionary entry is automatically listed by default.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- `(C` or COL-HDR-SUPP suppresses both the default page header and the column headers. COL.HDR.SUPP (note two P's) is a synonym for COL-HDR-SUPP.
- `(D` or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- `(E` prevents the listing of data in vertical format when listing more than five *dict* items. Listing remains in horizontal format, regardless of width. This is the opposite of VERT.
- `(F` suppresses "not on file" message generation. Because select lists are implemented as SQL joins that only return rows that are in both the select list and the file, Caché MultiValue compares each item in the list with the file; items that don't match are added to the error list, unless suppressed using this option.
- `(H` or HDR-SUPP suppresses the default page header. It does not suppress a page header specified using the HEADING clause. HDR.SUP and SUPP are synonyms for HDR-SUPP.
- `(I` or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- `(N` or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **LIST** operation.

- *field* COL.HDG *name* substitutes the specified *name* for the default *field* name. Delimit *name* with double quotes or backslash characters. DISPLAY.NAME is a synonym for COL.HDG.
- COL-SUPP suppresses the column headers. COL.SUP is a synonym for COL-SUPP.
- DBL-SPC displays data listed in horizontal format as double-spaced. The default is single-spaced. DBL.SPC is a synonym for DBL-SPC.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.
- VERT displays listed data in vertical format. The default is to list data in horizontal format when listing five or fewer *dict* items, and vertical format when listing more than five *dict* items. VERTICALLY is a synonym for VERT.

After listing each full page of items, **LIST** issues a prompt to the user to display the next page, unless you specified `(N`. To terminate a listing before reaching its end, specify Q at the display prompt. At the end of the listing, **LIST** specifies the total number of items listed, unless you specify NI-SUPP to suppress this total count.

See Also: COUNT, CT, LIST.ITEM, LIST.LABEL, LISTDICT LISTF, SELECT, SORT, STAT

1.71 LIST.INDEX

The **LIST.INDEX** command lists the indices defined for the specified file.

```
LIST.INDEX filename [index | ALL] (D
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **LIST.INDEX** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the index to be listed is stored in a named data section.

If the specified *filename* has no defined indices, **LIST.INDEX** returns an [842] error message.

You can list a specific index, or list all indices defined for the specified file. Note that **LIST.INDEX** lists indices of all types, including bitmapped indices. The Caché MVBasic index commands do not handle bitmapped indices.

The `(D` letter code option lists details about each index.

Note that **LIST.INDEX** has a VOC Attribute 3 value of "M", indicating that it is a Caché class method. See the **%MV.Verbs.ListIndex()** method in the *InterSystems Class Reference*.

See Also: CREATE.INDEX

1.72 LIST.ITEM

The **LIST.ITEM** command lists fields with their attributes.

```
LIST.ITEM [DICT] filename [field1 [field2 ...]] [query] [(DFHNPYZ]
```

LIST.ITEM can return field attributes from all of the fields of a file, for one or more specified fields, or for fields selected by a query. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it

must be enclosed with single quote characters. The attributes of an item are presented as numbered lines. If the *field* is an MVBasic program, the attributes are the numbered lines of the program.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order.

The following are supported letter code and keyword options:

- (D or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- (F suppresses "not on file" message generation. Because select lists are implemented as SQL joins that only return rows that are in both the select list and the file, Caché MultiValue compares each item in the list with the file; items that don't match are added to the error list, unless suppressed using this option.
- (H or HDR-SUPP suppresses the default page header. It does not suppress a page header specified using the HEADING clause. HDR.SUP and SUPP are synonyms for HDR-SUPP.
- (N or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **LIST.ITEM** operation.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.

After listing each full page of items, **LIST.ITEM** issues a prompt to the user to input a character to display the next page, unless you specified (N. This prompted character is not echoed on the list display, regardless of the MultiValue emulation. To terminate a listing before reaching its end, specify Q at the display prompt. At the end of the listing, **LIST.ITEM** specifies the total number of items listed, unless you specify NI-SUPP to suppress this total count.

LIST.ITEM returns fields and their attributes in the order presented. To return fields in alphabetical order, use **SORT.ITEM**.

See Also: LIST, SELECT, SORT.ITEM

1.73 LIST.JOB

The **LIST.JOB** command displays a table listing all running phantom processes.

```
LIST.JOB
```

LIST.JOB lists all running processes; **JOBS** lists all running processes initiated by the current process.

See Also: JOBS

1.74 LIST.LABEL

The **LIST.LABEL** command list values returned from a file in a display format that you specify. One use of this command is to format data for mailing labels.

```
LIST.LABEL [DICT] filename [field1 [field2 ...]] [query] [(DFNPYZ]
```

LIST.LABEL can either format all fields in the file, the specified fields from the file, or the fields from a file selected using a Caché MultiValue SQL (CMQL) query. **LIST.LABEL** prompts you to specify a display format with the following prompt: `COUNT, ROWS, SKIP, INDENT, SIZE, SPACE {,C}?`. You respond to this prompt with a comma-separated series of integers that describe the display format. The simplest prompt response is `1,1`, which means one item per line (COUNT), and one line space per line (ROWS). Format `1,1` is the same format as the **LIST** command. More complex display formats are described below.

By default, **LIST.LABEL** does not display a page header or footer. You can specify a page header using the HEADING clause. You can specify a page footer using the FOOTING clause.

The optional DICT keyword causes the command to access the DICT entries in the *filename* dictionary file; otherwise, *filename* is assumed to be a data file. If there are multiple defined data sections (data files), you can specify *filename* as `filename,datasection`.

The optional *field* arguments permit you to specify which fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. If you omit the *field* argument all fields in the file are listed.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- `(D` or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- `(F` suppresses “not on file” message generation. Because select lists are implemented as SQL joins that only return rows that are in both the select list and the file, Caché MultiValue compares each item in the list with the file; items that don’t match are added to the error list, unless suppressed using this option.
- `(N` or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **LIST.LABEL** operation.

After listing each full page of items, **LIST.LABEL** issues a prompt to the user to input a character to display the next page, unless you specified `(N`. This prompted character is not echoed on the list display, regardless of the MultiValue emulation. To terminate a listing before reaching its end, specify Q at the display prompt.

The following are the prompt values used to specify display format:

COUNT	Number of fields per row, specified as an integer. This value is mandatory, the minimum value is 1.
ROWS	Number of rows per “label” specified as an integer. This value is mandatory, the minimum value is 1. Commonly this is equivalent to vertical line spacing.
SKIP	*Optional* — Number of blank lines between “labels” specified as an integer. Commonly this is equivalent to vertical line spacing. The default is 0.
INDENT	*Optional* — Left indent, in character spaces, specified as an integer. The default is 0.
SIZE	*Optional* — If COUNT>1, specifies how many character spaces to allocate for each item on the line, specified as an integer. If SIZE is larger than an actual field value, blank spaces are appended to comprise the total SIZE; if SIZE is smaller than an actual field value, fields are concatenated and no blank spaces are appended. The default is 0.
SPACE	*Optional* — If COUNT>1, specifies how many character spaces to allocate for each item on the line, specified as an integer. If SPACE is larger than an actual field value, blank spaces are appended to comprise the total SPACE; if SPACE is smaller than an actual field value, fields are concatenated and no blank spaces are appended. SIZE and SPACE are added together. The default is 0.
C	*Optional* — The letter C code character. If COUNT>1, specifies do not print empty fields. The default is to print an empty field as SIZE+SPACE blank spaces.

See Also: LIST, SORT.LABEL

1.75 LIST.LOCKS

The **LIST.LOCKS** command displays a table listing the current locks. It list both system locks and locks established by the current process.

```
LIST.LOCKS
```

For all locks, **LIST.LOCKS** lists the process ID of the process holding the lock and the lock type (X (exclusive) or S (shared)). Locks established using the MVBasic LOCK command are displayed as LOCK *nnn*. Locks established by opening a sequential file are displayed as FILE *filename*. Other locks display the lock global variable and its complete pathname.

See Also: CLEAR.LOCKS

1.76 LIST.TRIGGER

The **LIST.TRIGGER** command lists the triggers defined for the specified file.

```
LIST.TRIGGER filename
```

It lists each trigger in the following format: `event = file subroutine`. For example: PREREAD = BP TRTN.

See Also: CREATE.TRIGGER

1.77 LISTDICT

The **LISTDICT** command displays a table listing the file's dictionary entries.

```
LISTDICT filename
```

LISTDICT displays the DICT entries in the *filename* dictionary file as a table. This is similar to `LIST DICT filename`, but differs in presentation format.

Note that **LISTDICT** has a VOC Attribute 3 value of "M", indicating that it is a Caché class method. See the **%MV.Verbs.ListDict()** method in the *InterSystems Class Reference*.

See Also: LIST

1.78 LISTF

The **LISTF** command lists the MultiValue files in the VOC.

```
LISTF [field [field2 ...]] [ALL] [query] [(CDEHNPYZ]
```

For each file in the VOC, **LISTF** lists the file name, the file type (F or Q), the corresponding data file global, and the corresponding dictionary file global. By default, **LISTF** begins its listing with the current date and time, and ends its listing with the total number of files listed.

If you CATALOG a compiled MVBasic file, that file appears in the **LISTF** listing as file type F. Note that file names are case-sensitive and listed in ASCII order (uppercase letters are listed before lowercase letters).

The optional *field* arguments permit you to specify which file fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. By default, fields are listed in ascending collation order. If you omit the *field* argument all file fields are listed. If you specify ALL, all the attribute values for each field are listed; this is similar to **LIST.ITEM**.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. Note that the first conditional clause cannot be specified as a WITH clause; it must be specified either as a WHEN clause, or with no conditional clause keyword. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- `(C` or COL-HDR-SUPP suppresses both the default page header and the column headers. COL.HDR.SUPP (note two P's) is a synonym for COL-HDR-SUPP.
- `(D` or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- `(E` prevents the listing of data in vertical format. Listing remains in horizontal format, regardless of width. This is the opposite of VERT.
- `(H` or HDR-SUPP suppresses the default page header. It does not suppress a page header specified using the HEADING clause. HDR.SUP and SUPP are synonyms for HDR-SUPP.
- `(N` or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.

- (Z displays the CMQL Query Execution Plan before performing the **LISTF** operation.
- *field* COL.HDG *name* substitutes the specified *name* for the default *field* name. Delimit *name* with double quotes or backslash characters. DISPLAY.NAME is a synonym for COL.HDG.
- COL-SUPP suppresses the column headers. COL.SUP is a synonym for COL-SUPP.
- DBL-SPC displays data listed in horizontal format as double-spaced. The default is single-spaced. DBL.SPC is a synonym for DBL-SPC.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.
- VERT displays listed data in vertical format. The default is to list data in horizontal format when listing five or fewer *dict* items, and vertical format when listing more than five *dict* items. VERTICALLY is a synonym for VERT.

After listing each full page of items, **LISTF** issues a prompt to the user to input a character to display the next page, unless you specified (N. This prompted character is not echoed on the list display, regardless of the MultiValue emulation. To terminate a listing before reaching its end, specify Q at the display prompt.

You can use **SET.FILE** to create a Q-type file.

See Also: LIST

1.79 LISTME

The **LISTME** command displays a table listing the current MultiValue user processes.

```
LISTME
```

For each user process, **LISTME** lists the process ID (pid), the port number, the date and time of initialization of the MV Shell, and the username. The current process is indicated by an asterisk preceding the pid number.

This listing is initiated by displaying a header, and concludes with a count of the number of items listed.

After listing each full page of items, **LISTME** issues a prompt to the user to input a character to display the next page. To terminate listing, input the letter Q at the display prompt. This prompted character is not echoed on the list display, regardless of the MultiValue emulation.

The **LISTME** command returns the same information as the **LISTU** and **STATUS** commands. However, **LISTME** can only list processes when invoked from the USER account. **LISTU** and **STATUS** list all active processes when invoked from any account.

See Also: LISTU, LOGOFF, STATUS, WHERE

1.80 LISTPA

The **LISTPA** command lists the paragraphs in the VOC.

```
LISTPA [field [field2 ...]] [ALL] [query] [(CDEINPYZ]
```

LISTPA is supported for the D3, IN2, jBASE, MVBase, PICK, R83, POWER95, Reality, and Ultimate emulations. Caché MultiValue and the INFORMATION, PIOpen, Prime, UniData, and UniVerse emulations return no data from this command.

For each paragraph field in the VOC, **LISTPA** lists @ID, F1, and F2. F1 is always “PA”. If no paragraphs exist (or this command is not supported in the current emulation), it returns a [401] message.

The optional *field* arguments permit you to specify which paragraph fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. By default, fields are listed in ascending collation order. If you omit the *field* argument all paragraph fields are listed. If *field* does not exist, or is not a paragraph field, a “not found” message is returned for that item.

If you specify ALL, all the attribute values for each field are listed. This is similar to **LIST.ITEM**.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. Note that the first conditional clause cannot be specified as a WITH clause; it must be specified either as a WHEN clause, or with no conditional clause keyword. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- `(C` or COL-SUPP suppresses the column headers. COL.SUP is a synonym for COL-SUPP.
- `(D` or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- `(E` prevents the listing of data in vertical format. Listing remains in horizontal format, regardless of width. This is the opposite of VERT.
- `(I` or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- `(N` or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **LISTPA** operation.
- *field* COL.HDG *name* substitutes the specified *name* for the default *field* name. Delimit *name* with double quotes or backslash characters. DISPLAY.NAME is a synonym for COL.HDG.
- DBL-SPC displays data listed in horizontal format as double-spaced. The default is single-spaced. DBL.SPC is a synonym for DBL-SPC.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.
- VERT displays listed data in vertical format. The default is to list data in horizontal format when listing five or fewer *dict* items, and vertical format when listing more than five *dict* items. VERTICALLY is a synonym for VERT.

See Also: LISTPH, LISTS

1.81 LISTPEQS

The **LISTPEQS** command displays a table listing the printer queue elements and their status.

```
LISTPEQS ["account"] [jobno[-jobno]] [(ACF]
```

The optional *account* argument allows you limit display to only the elements assigned to a specific account. *account* is case-sensitive, and must be specified as a quoted string. The optional *jobno* argument allows you to specify a single print job number, or a range of print jobs.

The (ACF options consist of one or more of these letter codes in any order, preceded by an open parenthesis. The letter codes have the following meaning: A=display only jobs created by the current account. C=do not display detailed information, just return the total number of queue elements and pages in use. F=sort the display by form queue number.

See Also: LISTPTR, SPOOL, SP.DELETE, SP-EDIT

1.82 LISTPH

The **LISTPH** command lists the phrases in the VOC.

```
LISTPH [field [field2 ...]] [ALL] [query] [(CDEINPYZ]
```

LISTPH is supported for the D3, IN2, jBASE, MVBase, PICK, R83, POWER95, Reality, and Ultimate emulations. Caché MultiValue and the INFORMATION, PIOpen, Prime, UniData, and UniVerse emulations return no data from this command.

For each phrase field in the VOC, **LISTPH** lists @ID, F1, and F2. F1 is always “PH”. If no phrases exist (or this command is not supported in the current emulation), it returns a [401] message.

The optional *field* arguments permit you to specify which phrase fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. By default, fields are listed in ascending collation order. If you omit the *field* argument all phrase fields are listed. If *field* does not exist, or is not a phrase field, a “not found” message is returned for that item.

If you specify ALL, all the attribute values for each field are listed. This is similar to **LIST.ITEM**.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. Note that the first conditional clause cannot be specified as a WITH clause; it must be specified either as a WHEN clause, or with no conditional clause keyword. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- (C or COL-SUPP suppresses the column headers. COL.SUP is a synonym for COL-SUPP.
- (D or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- (E prevents the listing of data in vertical format. Listing remains in horizontal format, regardless of width. This is the opposite of VERT.
- (I or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- (N or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **LISTPH** operation.
- *field* COL.HDG *name* substitutes the specified *name* for the default *field* name. Delimit *name* with double quotes or backslash characters. DISPLAY.NAME is a synonym for COL.HDG.
- DBL-SPC displays data listed in horizontal format as double-spaced. The default is single-spaced. DBL.SPC is a synonym for DBL-SPC.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.

- VERT displays listed data in vertical format. The default is to list data in horizontal format when listing five or fewer *dict* items, and vertical format when listing more than five *dict* items. VERTICALLY is a synonym for VERT.

See Also: LISTPA, LISTS

1.83 LISTPTR

The **LISTPTR** command displays a table listing the current printer assignments.

```
LISTPTR  [jobno[-jobno]]
```

The optional *jobno* argument allows you to specify a single print job number, or a range of print jobs.

See Also: LISTPEQS

1.84 LISTS

The **LISTS** command lists the sentences in the VOC.

```
LISTS [field [field2 ...]] [ALL] [query] [(CDEINPYZ]
```

LISTS is supported for the D3, IN2, jBASE, MVBase, PICK, R83, POWER95, Reality, and Ultimate emulations. Caché MultiValue and the INFORMATION, PIOpen, Prime, UniData, and UniVerse emulations return no data from this command.

For each sentence field in the VOC, **LISTS** lists @ID, F1, and F2. F1 is always “S”. If no sentences exist (or this command is not supported in the current emulation), it returns a [401] message.

The optional *field* arguments permit you to specify which sentence fields to list. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. By default, fields are listed in ascending collation order. If you omit the *field* argument all sentence fields are listed. If *field* does not exist, or is not a sentence field, a “not found” message is returned for that item.

If you specify ALL, all the attribute values for each field are listed. This is similar to **LIST.ITEM**.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. Note that the first conditional clause cannot be specified as a WITH clause; it must be specified either as a WHEN clause, or with no conditional clause keyword. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- (C or COL-SUPP suppresses the column headers. COL.SUP is a synonym for COL-SUPP.
- (D or DET-SUPP suppresses detail listings. DET.SUP is a synonym for DET-SUPP.
- (E prevents the listing of data in vertical format. Listing remains in horizontal format, regardless of width. This is the opposite of VERT.
- (I or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- (N or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.

- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **LISTS** operation.
- *field* COL.HDG *name* substitutes the specified *name* for the default *field* name. Delimit *name* with double quotes or backslash characters. DISPLAY.NAME is a synonym for COL.HDG.
- DBL-SPC displays data listed in horizontal format as double-spaced. The default is single-spaced. DBL.SPC is a synonym for DBL-SPC.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.
- VERT displays listed data in vertical format. The default is to list data in horizontal format when listing five or fewer *dict* items, and vertical format when listing more than five *dict* items. VERTICALLY is a synonym for VERT.

See Also: LISTPA, LISTPH

1.85 LISTU

The **LISTU** command lists the current MultiValue user processes.

```
LISTU
```

LISTU lists the current MultiValue user processes in port number order. For each user process, **LISTU** lists the process ID (pid), the port number, the date and time of initialization of the MV Shell, and the username. The current process is indicated by an asterisk preceding the pid number.

This listing is initiated by displaying a header, and concludes with a count of the number of items listed.

After listing each full page of items, **LISTU** issues a prompt to the user to input a character to display the next page. To terminate listing, input the letter Q at the display prompt. This prompted character is not echoed on the list display, regardless of the MultiValue emulation.

The **LISTU** and **STATUS** commands are functionally identical.

See Also: LISTME, LOGOFF, STATUS, WHERE

1.86 LOGOFF

The **LOGOFF** command logs off a current MultiValue user process. You can log off your own current process, or another active process.

```
LOGOFF portno
```

Specifying **LOGOFF** with a port number logs off the MV Shell on the specified user terminal process. When you issue a log off, the terminal process immediately exits the MV Shell and returns to the Caché mode prompt. If you specify an invalid *portno*, the **LOGOFF** command completes successfully, performing no operation.

To determine the *portno* of current user processes, use the **WHO** command, or the **@PORTNO** MVBasic system variable.

The **LOGOFF** and **LOGOUT** commands are similar.

See Also: LOGOUT, Q, QUIT, WHO

1.87 LOGOUT

The **LOGOUT** command logs off a current MultiValue user process. You can log off your own current process, or another active process.

```
LOGOUT [pid]
```

Specifying **LOGOUT** with no operand logs off the MV Shell on the current terminal process. (You can use the **Q** or **QUIT** command to perform the same operation.) Specifying **LOGOUT pid** logs off the MV Shell on the specified user terminal process. Specifying **LOGOUT ALL** logs off the MV Shell on all user terminal processes except the current terminal process.

When you issue a log off, the terminal process immediately exits the MV Shell and returns to the Caché mode prompt. If you specify an invalid *pid*, the **LOGOUT** command completes successfully, performing no operation.

To determine the *pid* of current user processes, use the **LISTME** command.

The **LOGOUT** and **LOGOFF** commands are similar.

See Also: LISTME, LOGOFF, Q, QUIT

1.88 LOGTO

The **LOGTO** command changes your MV Shell login to another account (namespace).

```
LOGTO account
```

LOGTO changes your current environment to a different account environment. The *account* must be an existing account/namespace. Each account has its own files, local variables, and emulation setting. The current account is shown as the command line prompt.

If the specified *account* is not valid (does not exist), **LOGTO** generates a [229] error. An account may be a Caché namespace; it does not have to be an account created using **CREATE.ACCOUNT**. The SYSPROG account corresponds to the %SYS namespace. If you are already in the specified *account*, no operation is performed.

LOGTO causes the MultiValue Shell to search the target *account* VOC in the following order: an item with the same name as the user, an item with the same name as *account*, or an item named "LOGIN". The MultiValue Shell runs the first of these items that it encounters.

Changing to an *account* means entering the emulation environment established for that account. Refer to CEMU for details. If a &HOLD& file has been created in the old account, **LOGTO** create a &HOLD& file in the new account. For details on the relationship between accounts and namespaces, and the naming conventions used for each, refer to "MV Accounts and Caché Namespaces" in *Operational Differences between MultiValue and Caché*.

See Also: CEMU, CREATE.ACCOUNT, WHO

1.89 MESSAGE

The **MESSAGE** command sends a message text to the specified users or processes.

```
MESSAGE username [username]
MESSAGE pid [pid]
```

Specify one or more usernames or process IDs (pid). Multiple arguments are separated by blank spaces. The **MESSAGE** command prompts you for a message text. Type the text then press ENTER.

MESSAGE sends the message to all specified users that are currently logged on. It displays “message sent” with username and pid for each valid recipient. It displays “not logged on” for each invalid recipient. usernames are not case-sensitive. Duplicate values are ignored.

See Also: LISTME, LISTU, STATUS, WHO

1.90 MVI

The **MVI** command locates the source code for a MVBasic routine.

```
MVI [MVB.]routine[.MVI] [linenumber]
```

MVI allows you to cross reference line numbers from MVBasic intermediate source code (MVI code) to the original MVBasic source code (MVB code). MVI searches for MVB.*routine*, where *routine* is a hexadecimal module number assigned to the MVBasic routine when it is compiled. The MVB. prefix and .MVI suffix are both optional. The *linenumber* corresponds to a line in the MVI code; if you omit *linenumber* it defaults to line 1. If the specified *routine* is not located, you may be searching in the wrong account. For further details, refer to the MVI: MVI-To-MVB Cross-Reference section of the *Operational Differences between MultiValue and Caché* manual.

See Also: BASIC

1.91 MVIMPORT

The **MVIMPORT** command imports accounts from other MultiValue implementations to Caché MultiValue. This command is described in the MVIMPORT chapter of this manual.

1.92 NSELECT

The **NSELECT** command generates a select list of items in the supplied (or default) select list that are *not* in the file.

```
NSELECT filename [FROM n] [TO n]
```

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **NSELECT** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the select list is to be compared with the contents of a named data section.

You can use the FROM clause to specify an existing numeric select list. By combining the FROM and TO clauses you can specify a range of numbered select lists. By default, **NSELECT** uses select list 0. If there is no active select list, **NSELECT** returns a [240] message.

See Also: BSELECT, CLEARSELECT, QSELECT, SEARCH, SELECT, SSELECT

1.93 OFF

The **OFF** command logs off your current MultiValue user process.

```
OFF
```

Specifying **OFF** logs off the MV Shell on your user terminal process. When you issue an **OFF**, the terminal process immediately exits the MV Shell and returns to the Caché mode prompt.

The **QUIT**, and **Q** commands are synonyms. The **OFF** command is functionally identical.

See Also: LOGOFF, LOGOUT, Q, QUIT, WHO

1.94 P

The **P** command suppresses terminal screen display. It suppresses all terminal display, including displaying the terminal prompt.

```
P [ON | OFF]
```

Specifying **P** with no operand toggles display suppression. **P ON** suppresses display. **P OFF** re-enables display.

The **KEYS** command temporarily overrides the **P**. However, when **KEYS** times out, it returns to the prior **P** mode. This may be mistaken for a hang state.

The **P** and **HUSH** commands are synonyms.

See Also: DISPLAY, HUSH

1.95 PAGE.MESSAGE

The **PAGE.MESSAGE** command displays an end-of-page message on the screen when displaying multi-page output.

```
PAGE.MESSAGE [ON | OFF]
```

Specifying **PAGE.MESSAGE** with no operand returns the current setting. **PAGE.MESSAGE ON** displays the message “Press any key to continue” at the end of the page and pauses awaiting user response. **PAGE.MESSAGE OFF** does not display a message; it simply pauses at the end of the page awaiting user response. **PAGE.MESSAGE ON** is the Caché default. The default may differ in other MultiValue emulations.

1.96 PHANTOM

The **PHANTOM** command starts a phantom process in which to run the specified MultiValue command.

```
PHANTOM [BRIEF | SQUAWK] command
```

PHANTOM initiates a phantom (background) process. It does not validate the user-specified command. **PHANTOM** returns the process ID (pid) assigned to the phantom process. Normal terminal output does not appear on the screen. By default, *command* output is stored as a record in the &PH& file for the current account. In D3, IN2, jBASE, MVBase, R83, POWER95, Reality, and Ultimate emulations, this output is instead directed to the Spooler file.

The optional SQUAWK keyword causes **PHANTOM** to also display the record number of the record created in the &PH& file. In D3, IN2, jBASE, MVBase, R83, POWER95, Reality, and Ultimate emulations, SQUAWK returns the spooler job number.

The optional BRIEF keyword starts a background process and returns the process ID (pid), but causes no output to be generated.

PHANTOM is similar to the **ZH** and **Z** commands. Unlike **PHANTOM**, **Z** and **ZH** prompt for the account, password, and command to execute. Like **PHANTOM**, **ZH** directs *command* output to either the &PH& file or the Spooler file, depending on emulation. **Z** does not retain *command* output.

See Also: JOBS, LIST.JOB, Z, ZH

1.97 PQ.SELECT

The **PQ.SELECT** command copies the default select list (select list 0) into the specified numbered select list.

```
PQ.SELECT n
```

The *n* argument is a select list number in the range 1 through 10 (inclusive). If *n* is out of range, **PQ.SELECT** returns a [819] message. If the default select list is not active, **PQ.SELECT** returns a [240] message.

PQ.SELECT is used to make the contents of the default select list available to a MultiValue PROC, which can only reference numbered select lists. For further details refer to *Caché MultiValue PROC Reference*.

1.98 PQ.RESELECT

The **PQ.RESELECT** and **PQ.SELECT** commands are functionally identical.

1.99 PRINT.CATALOG

The **PRINT.CATALOG** command displays details of catalog pointers and their program references.

```
PRINT.CATALOG filename [itemspec]
```

The *filename* argument can be a compiled and catalogued MVBasic program file, or the VOC. The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **PRINT.CATALOG** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the catalog information is stored in a named data section.

itemspec is the name of a item, or a list of items to display. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the file should be displayed. If *itemspec* is omitted, **PRINT.CATALOG** uses the active select list as the source of item names. If there is no active select list, **PRINT.CATALOG** issues a "Item Id:" user prompt.

PRINT.CATALOG displays a message for each item cataloged. A local catalog item generates a [232] message. A global catalog item generates a [233] message. If an item does not exist in the file or is not catalogued, **PRINT.CATALOG** returns without issuing an error message.

See Also: CATALOG, DECATALOG

1.100 PRINT.ERR

The **PRINT.ERR** command displays specified items from the ERRMSG file.

```
PRINT.ERR ERRMSG [itemspec]
```

PRINT.ERR returns the specified messages from the ERRMSG file. For example, `PRINT.ERR ERRMSG 201 210` returns the message texts for errors [201] and [210], one message per line.

itemspec specifies one or more numeric error codes used to retrieve and display the corresponding error messages. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all messages in the ERRMSG file should be displayed. If *itemspec* is omitted, **PRINT.ERR** uses the active select list as the source of error codes. If you do not specify a *msg* and there is no active select list, **PRINT.ERR** prompts you for an item ID. If *msg* is not found in ERRMSG, **PRINT.ERR** returns a [202] "not on file" error.

As each item is accessed, **PRINT.ERR** updates the @RECORD special variable with the retrieved error message.

1.101 PTERM

The **PTERM** command sets and displays terminal options.

```
PTERM [LPTR channel] [DEVICE name] [DISPLAY] [option value]
```

By default, the terminal is the user terminal. The *option* argument is the name of an option; the *value* argument is a keyword setting for that option. Available *option value* pairs are CASE INVERT and CASE NOINVERT, CRMODE INLCR and CRMODE NOINLCR.

1.102 Q

The **Q** command quits the MultiValue Shell.

```
Q
```

Specifying **Q** causes the current terminal process to immediately exit the MV Shell and returns to the Caché mode prompt. To perform the same operation on other terminal processes, use the **LOGOFF** or **LOGOUT** commands.

The **Q** and **QUIT** commands are synonyms.

See Also: LOGOFF, LOGOUT, OFF, QUIT

1.103 QSELECT

The **QSELECT** command generates a select list of the specified items.

```
QSELECT filename [itemspec] [slist]
```

The *filename* is the name of an existing file which is used as the source for the select list data. If the file doesn't exist **QSELECT** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the items to be selected are stored in a named data section.

itemspec is the name of a item, or a list of items to copy to the select list. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the file should be copied to the select list. If *itemspec* is omitted, **QSELECT** uses the active select list as the source of item names.

As each item is accessed, **QSELECT** updates the @RECORD special variable.

The *slist* argument allows you to specify a numeric select list. By default, **QSELECT** outputs to select list 0.

See Also: BSELECT, CLEARSELECT, NSELECT, SEARCH, SELECT, SSELECT

1.104 QUIT

The **QUIT** command quits the MultiValue Shell.

```
QUIT
```

Specifying **QUIT** causes the current terminal process to immediately exit the MV Shell and returns to the Caché mode prompt. To perform the same operation on other terminal processes, use the **LOGOFF** or **LOGOUT** commands.

The **QUIT** and **Q** commands are synonyms.

See Also: LOGOFF, LOGOUT, OFF, Q

1.105 REFORMAT

The **REFORMAT** command copies one or more DICT entries from a file and reformats them into an inverted file.

```
REFORMAT [USING dictname | DICT] filename field1 [field2 [...]] [query] [(PYZ]
```

REFORMAT takes an input *filename* and prompts you to specify an output file. You must specify the name of an existing file at the `File Name=` prompt. If the specified input *filename* is not an existing file, it generates a [200] error. If the specified output file is not an existing file, it generates a [201] error.

The optional DICT keyword specifies that *filename* is accessing a dictionary file; otherwise, the *filename* is assumed to be accessing a data file. If there are multiple defined data sections (data files), you can specify *filename* as `filename,datasection`.

The *field* arguments permit you to specify which DICT entries to reformat to the output file. You must specify at least one *field*. You can specify multiple *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. If any one of the *field* items is not a defined DICT entry, **REFORMAT** generates a [7011] error for each invalid entry and no DICT entries are processed. Upon successful completion, **REFORMAT** generates a [180] message specifying the number of items processed.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **REFORMAT** operation.

See Also: CREATE.FILE SREFORMAT

1.106 RUN

The **RUN** command runs an MVBasic program.

```
RUN filename item
```

The **RUN** command can execute an MVBasic program that has been compiled using the BASIC command. The MVBasic program does not need to be cataloged. Once an MVBasic program has been cataloged using CATALOG, it may be executed either by issuing a **RUN** command, or by simply invoking the *item* as a verb.

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **RUN** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *accountname* if the file is located in an account other than the current account. Specify *datasection* if the program to be run is stored in a named data section of the file.

Before you can run an MVBasic program, you must first compile it using the **BASIC** command. If the specified *item* doesn't exist **RUN** generates a [41] error message. If the specified *item* has not been compiled **RUN** generates a [40] error message. If no *item* is specified **RUN** generates a [203] error message.

See Also: ; (semicolon), BASIC, MVI

1.107 SAVE.LIST

The **SAVE.LIST** command copies the specified numbered select list into a named select list.

```
SAVE.LIST [listname] [FROM n]
```

Use the optional *listname* argument to specify a named select list; if omitted, the select list name is taken from the process ID (pid) or the UDT_SAVEDLIST environment variable. The optional FROM *n* argument is a select list number in the range 0 through 10 (inclusive); if omitted, select list 0 is the default. Upon successful completion, **SAVE.LIST** returns a message such as the following: `[247] 478 Items saved to list 'MYLIST'.`

When a select list is saved it becomes unavailable to the MVBasic READNEXT command. To make a select list available again, use **GET.LIST**.

If *n* is not an active select list, **SAVE.LIST** returns a [240] error message. If *n* is out of range, **SAVE.LIST** returns a [209] error message.

SAVE.LIST is the inverse of **GET.LIST**. To copy Select List 0 to another numbered select list, use the **PQ.SELECT** command.

See Also: GET.LIST, PQ.SELECT

1.108 SEARCH

The **SEARCH** command searches item(s) for string(s) and copies those that match to a select list.

```
SEARCH filename [itemspec]
```

SEARCH searches an item or multiple items for one or more strings. When it finds a match, it copies the item to select list 0.

The *filename* is the name of an existing file, which is created as a Caché global (^filename). If the file doesn't exist **SEARCH** returns a [201] error message. The *filename* can be specified using filespec syntax, as follows:

```
[DICT | DATA] [accountname,]filename[,datasection]
```

The optional DICT and DATA keywords enable you to specify a dictionary file or a data file. The default is a data file. Specify *datasection* if the items to be searched are stored in a named data section.

itemspec is the name of a item, or a list of items to search for a specified string. An *itemspec* can be specified using the following syntax:

```
item [item2 [...]] | *
```

You may specify a single item or multiple items separated by spaces. An asterisk (*) specifies that all items in the file should be searched. If *itemspec* is omitted, **SEARCH** uses the active select list as the source of one or more items to search. If *itemspec* is omitted and there is no active select list, **SEARCH** prompts you for an *item* value. If a specified *item* is not valid, **SEARCH** returns a [202] message for that item.

As each item is accessed, **SEARCH** updates the @RECORD special variable.

SEARCH prompts you for strings to search for. You can specify one or more strings at successive prompts, then press Enter at a prompt to conclude string input.

See Also: BSELECT, CLEARSELECT, NSELECT, QSELECT, SELECT, SSELECT

1.109 SELECT

The **SELECT** command generates a select list of items that satisfy the query criteria.

```
SELECT [DICT] filename [field1 [field2 ...] [dict [dict2 ...] | ALL] [query] [TO
listnum] [(FPYZ]
```

SELECT copies items selected from *filename* to a select list. If *filename* is not an existing file, **SELECT** generates a [200] error. If *filename* is an empty file, **SELECT** generates a [401] error and no select list is returned.

The optional DICT keyword specifies that *filename* is accessing a dictionary file; otherwise, the *filename* is assumed to be accessing a data file. If there are multiple defined data sections (data files), you can specify *filename* as `filename,datasection`. **SELECT** can specify any valid Caché MultiValue SQL (CMQL) query.

The optional *field* arguments permit you to specify which DICT entries to select. You can specify one or more *field* arguments separated by blank spaces. If a *field* argument is an item ID it must be enclosed with single quote characters. If you omit the *field* argument, all DICT entries in *filename* are selected, or all DICT entries are selected that pass the condition tests in *query*. If no items pass the *query* condition test, **SELECT** generates a [401] error and no select list is returned.

The optional *dict* arguments permit you to specify which DICT entries to select for each *field*. You can specify one or more *dict* entry names separated by blank spaces. DICT entry names are *not* enclosed with quote characters. If you omit the *dict*

argument, only the @ID (VOC) dictionary entry for each *field* is selected. If you specify ALL, all the *dict* attribute values for each field are selected.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

You can use the TO clause to specify a numeric select list. Valid *listnum* values are 0 through 10. By default, **SELECT** uses select list 0.

The following are supported letter code options:

- `(F` suppresses "not on file" message generation. Because select lists are implemented as SQL joins that only return rows that are in both the select list and the file, Caché MultiValue compares each item in the list with the file; items that don't match are added to the error list, unless suppressed using this option.
- `(P` redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- `(Y` displays query metadata.
- `(Z` displays the CMQL Query Execution Plan before performing the **SELECT** operation.

Upon successful completion, **SELECT** returns a message such as the following: `2 Items selected to list #0`. The first successful **SELECT** to any select list sets the boolean flag **$MVV(210)** to 1. **$MVV(210)** remains set to 1 until explicitly reset. The $MVV special variable is described in the *Caché ObjectScript Reference*.

See Also: BSELECT, CLEARSELECT, LIST, NSELECT, QSELECT, SEARCH, SSELECT

1.110 SET.FILE

The **SET.FILE** command creates a type Q file.

```
SET.FILE [account [filename [qname]]
```

SET.FILE with no arguments creates a Q-pointer file in the VOC in the current account. By default, it leaves Line 2 of the Q-pointer blank, which indicates the current account. You can create multiple Q-pointer files in the VOC. To specify a Q-pointer file other than the default QFILE, you must specify the *account* and *filename*, as shown in the following example:

```
SET.FILE "USER" "VOC" "QFILETWO"
```

The optional *account* argument permits you to specify the name of an account (Line 2); the default is the current account. The optional *filename* argument permits you to specify the name of a file in *account* (Line 3); the default is VOC. The optional *qname* argument permits you to specify the name of the Q-pointer in *filename*; the default is QFILE.

You can use **LISTF** to list files with their file type (F or Q), the corresponding data file and dictionary file.

See Also: LISTF

1.111 SETPTR

The **SETPTR** command lists or sets the current printer settings.

```
SETPTR [chan,width,depth,topmargin,botmargin,mode,option[,option]]
```

SETPTR with no arguments lists the current printer settings. **SETPTR** with a comma-separated list of positional arguments is used to change one or more printer settings.

For further details, refer to SETPTR in the "Spooler Commands" chapter of The Caché MultiValue Spooler.

1.112 SETPTR.DEFAULT

The **SETPTR.DEFAULT** command takes the current print channel 0 settings and establishes them as the print channel 0 default settings.

```
SETPTR.DEFAULT [LIST] [DELETE] [(LD ]
```

SETPTR.DEFAULT must be run from the SYSPROG account. Before issuing **SETPTR.DEFAULT** you define the print channel 0 settings using **SETPTR**. **SETPTR.DEFAULT** makes these settings the print channel 0 defaults for all future **SETPTR** commands systemwide. If any printer settings have not been set, **SETPTR.DEFAULT** establishes a default **SETPTR** characteristic for that setting. **SETPTR.DEFAULT** has no effect on print channels other than print channel 0.

The `LIST` keyword or the `(L` letter code option displays the current defaults that have been set. You can run **SETPTR.DEFAULT LIST** or **SETPTR.DEFAULT (L** from any account.

The **SETPTR.DEFAULT** settings remain in effect across system reboots until you issue a **SETPTR.DEFAULT DELETE** or **SETPTR.DEFAULT (D** command. The `DELETE` keyword or the `(D` letter code option reverts all settings to the initial printer default settings.

For further details, refer to SETPTR in the "Spooler Commands" chapter of The Caché MultiValue Spooler.

1.113 SH

The **SH** command issues an operating system command.

```
SH [commandline]

SH -c "commandline"
```

The **SH** command issues an operating system command without exiting the MultiValue Shell. Results are displayed to the MultiValue Shell. No additional windows are opened.

SH with no argument opens an interactive command prompt from which you can issue multiple operating system commands. To exit this command prompt, specify the quit command for the operating system. On Windows systems, this is EXIT.

SH commandline issues an operating system command as a background process. The *commandline* can be any valid command line for the current operating system. A *commandline* cannot exceed 248 characters in length. Upon completion it returns to the MultiValue Shell prompt.

The **SH -c "commandline"** alternate syntax is equivalent to **SH commandline**. This syntax is provided for UniVerse compatibility. The double quotes enclosing the *commandline* are mandatory.

You can use **COS** to issue an ObjectScript command without exiting the MultiValue Shell.

The **SH** command is supported on multiple operating system platforms. The **DOS** command is specific to Windows platforms. On Windows systems, the **DOS** and **SH** commands are synonyms.

On Windows systems you can use **SH set** (or **DOS set**) to display a list of environment variables. To display individual environment variables, you can use the MVBasic GETENV() function.

See Also: COS, DOS

1.114 SLEEP

The **SLEEP** command suspends the process for the specified number of seconds, or until the specified time. It then returns to the MultiValue Shell prompt.

```
SLEEP seconds
SLEEP time
```

You can specify *seconds* as an integer or a fraction.

You can specify *time* as local time in either 24-hour or 12-hour format. A 24-hour time is specified as hh:mm[:ss]. A 12-hour time is specified as hh:mm[:ss]{AM | PM}. In both formats, spaces are not permitted, leading zeros may be omitted, and the seconds component of the time is optional. The following are all valid 24-hour format *time* values: 02:34, 2:34:00, 14:34, 14:34:00. The following are all valid 12-hour format *time* values: 2:34PM, 02:34PM, 2:34:00PM. 2:34AM. Midnight can be represented by 24:00, 00:00, 12:00PM, 00:00PM, or 00:00AM. An invalid *time* argument generates a [6193] error.

Caché MultiValue determines local time as follows:

- It determines the current Coordinated Universal Time (UTC) from the system clock.
- It adjusts UTC to the local time zone by using the value of the Caché special variable $ZTIMEZONE.
- It applies local time variant settings (such as Daylight Saving Time) for that time zone from the host operating system.

1.115 SORT

The **SORT** command generates a sorted list of fields that satisfy the query criteria.

```
SORT [DICT] filename [field1 [field2 ...]] [dict [dict2 ...] | ALL] [query]
     [BY field | BY-DSND field | BY-EXP field | BY-EXP-DSND field] [(CDEFHINPYZ]
```

SORT returns a sorted display of the fields specified in *field* and/or selected by *query*. **SORT** is otherwise identical to the **LIST** command. Refer to the **LIST** command for further details.

Caché MultiValue defaults to sorting using BY logic: sorting items by their @ID values as single-valued elements in ascending collation sequence. Other MultiValue emulations default to BY-EXP logic.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

You can use the CMQL BY clause to sort by a *field* other than @ID, and/or sort in descending order, rather than ascending order. The BY keyword sorts the *field* values in ascending order (the default). The BY-DSND keyword sorts the *field* values in descending order. The BY-EXP keyword explodes the multivalue levels of *field* into a data row and sorting these values in ascending order. The BY-EXP-DSND keyword explodes the multivalue levels of *field* and sorts the values in descending order.

See Also: COUNT, LIST, SELECT

1.116 SORT.ITEM

The **SORT.ITEM** command generates a sorted list of fields that satisfy the query criteria and lists their items.

```
LIST.ITEM [DICT] filename [field1 [field2 ...]] [query] [(DFHNPYZ]
```

SORT.ITEM is the same as **LIST.ITEM**, except that it sorts the field names as single values in ascending collation sequence before listing them. Caché MultiValue defaults to sorting using BY logic: sorting fields by their @ID values as single-valued elements in ascending collation sequence. Other MultiValue emulations default to BY-EXP logic. You can specify a different sort order using the CMQL BY clause. See **LIST.ITEM** for additional details.

See Also: LIST.ITEM

1.117 SORT.LABEL

The **SORT.LABEL** command list values returned from a file, sorted in alphabetical order, presented in a display format that you specify.

```
SORT.LABEL [DICT] filename [field1 [field2 ...]] [query] [(DFNPYZ]
```

SORT.LABEL is the same as **LIST.LABEL**, except that it sorts the values in ascending collation sequence before formatting them for display. See **LIST.LABEL** for details.

See Also: LIST.LABEL

1.118 SORT.LIST

The **SORT.LIST** command sorts a saved select list in the &SAVEDLISTS& file.

```
SORT.LIST [filename [listname] ]
```

The optional *filename* is the destination file where the sorted select list is to be stored. If you omit *filename*, **SORT.LIST** prompts you for the list ID. The optional *listname* is the name of an existing select list in &SAVEDLISTS&; the default is select list 0. Caché stores the &SAVEDLISTS& file using the ^SAVEDLISTS global.

See Also: COPY.LIST, EDIT.LIST

1.119 SP.x Commands

Caché MultiValue supports 39 commands that control the Spooler. The names of these commands begin with either "SP-" or "SP." Caché supports both variant forms: the hyphen form and the dot form. For example, **SP-ASSIGN** and **SP.ASSIGN** are different names for the same command. In most cases, these two forms are synonymous. The one exception is **SP-EDIT** and **SP.EDIT**, which provide different syntax options.

The following spooler commands are supported: SP.ASSIGN, SP.AUX, SP.CLEAR, SP.CLOSE, SP.CONDUCT, SP.CONTROL, SP.COPIES, SP.COPY, SP.CREATE, SP.DELETE, SP.DEVICE, SP.DISPLAY, SP-EDIT, SP.EDIT, SP.EJECT, SP.FORM, SP.FQDELETE, SP.GLOBAL, SP.JOBS, SP.KILL, SP.LOOK, SP.MOVEQ, SP.NEWTAB, SP.OPEN, SP.OPTS, SP.PAGESIZE, SP.POSTAMBLE, SP.PREAMBLE, SP.PURGEQ, SP.RESUME, SP.SHOW, SP.SKIP, SP.START, SP.STATUS, SP.STOP, SP.SUSPEND, SP.SWITCH, SP.TESTPAGE, SP.VERBOSE.

For further information on these commands, refer to the "Spooler Commands" chapter of *The Caché MultiValue Spooler*.

1.120 SPOOL

The **SPOOL** command controls the spooling of files for printing. It has three forms: send a file to the spooler for printing; list the files pending on the spooler queue; delete a print job from the spooler queue.

```
SPOOL filename itemID [-NOHEAD] [-O]
SPOOL -LIST [formname]
SPOOL -CANCEL joblist
```

`SPOOL filename itemID` takes a MultiValue item and prints it to the currently assigned printer (the default is print queue 0). The optional –NOHEAD keyword suppresses banners defined by **SETPTR** for the currently assigned printer.

`SPOOL -LIST` lists all the jobs on the spooler table. The optional *formname* argument allows you to filter to a single form queue name. The form queue can be specified either by name or by number. The default form queue has the name STANDARD, and a form queue number of 0. It can be specified as "STANDARD", "0", "F0", "FN0", or "FQ0".

`SPOOL -CANCEL` deletes one or more pending print jobs. The *joblist* argument allows you to specify any number of individual print job numbers (separated by blank spaces), as shown in the following example: `SPOOL -CANCEL 66 68 71`. You can also delete a range of print jobs, as shown in the following example: `SPOOL -CANCEL 66-70`.

The -O option performs no operation and is ignored. It is accepted in syntax for compatibility with UniData code.

For further details and examples, refer to The Caché MV Spooler manual.

See Also: SP.DELETE

1.121 SREFORMAT

The **SREFORMAT** command copies one or more fields from a file and reformats them into a sorted inverted file.

```
SREFORMAT [USING dictname | DICT] filename field1 [field2 [...]] [query] [(PYZ]
```

The **SREFORMAT** sorts the records returned in ascending collation sequence. In all other respects it is identical to the **REFORMAT** command.

See Also: REFORMAT

1.122 SSELECT

The **SSELECT** command generates a sorted select list of items that satisfy the query criteria.

```
SSELECT [DICT] filename [field1 [field2 ...] [dict [dict2 ...] | ALL] [query] [TO
listnum] [(FPYZ]
```

The **SSELECT** sorts the records selected in ascending collation sequence. In all other respects it is identical to the **SELECT** command.

See Also: SELECT

1.123 STACK

The **STACK** command changes the behavior of the MultiValue Shell command line recall stack.

```
STACK [option [option]]
```

STACK with no argument returns the current MV Shell recall stack settings.

The following **STACK** options are supported. You can specify multiple options in any order, separating options with spaces.

MAX *nn* — Sets the maximum number of entries in the recall stack to *nn*. The default is 99.

DUP ON | OFF — Allows duplicate commands in the recall stack. The default is DUP OFF.

CLEAR — Clears the current recall stack in the current MV Shell.

RECALL START | END — Positions the cursor to either the start or the end of a recalled command. The default is RECALL END.

BY *type* — Specifies how the MV Shell provides persistence of command recall during logon, logoff, and logto. The available *type* values are AUTHORIZATION, COS, IP, LOGNAME, NAME, NONE, PORT, USER, and ROUTINE *name*. The default is BY COS.

When **STACK BY AUTHORIZATION** is specified, the commands are stored in the terminal command line stack indexed by the Caché username. This is the same value contained in the @AUTHORIZATION system variable. When **STACK BY USER** is specified, the commands are stored in the terminal command line stack indexed by the operating system username. This is the same value contained in the @USER system variable. **STACK BY LOGNAME** is a synonym for **STACK BY USER**.

When **STACK BY ROUTINE *name*** is specified, an ObjectScript routine or a MV subroutine is invoked at MV Shell logon and logoff. It is up to that routine to save and restore the stack in whatever way it sees fit. The syntax of *name* determines whether an ObjectScript routine or a MV subroutine is invoked. If *name* contains a ^ character, Caché assumes the routine is a standard ObjectScript routine. For example, `SHELL^MYFUNCS` would be procedure SHELL in the ObjectScript module MYFUNCS. If *name* does not contain a ^ character, Caché assumes the routine is a MV subroutine. For example `MYSHELL` indicates that Caché should call the MV subroutine MYSHELL.

1.124 STACKCOMMON

The **STACKCOMMON** command specifies whether the MVBasic PERFORM statement stacks unnamed COMMON variable areas.

```
STACKCOMMON
STACKCOMMON ON
STACKCOMMON OFF
```

STACKCOMMON with no argument returns the current setting. **STACKCOMMON ON** causes each PERFORM to NEW the unnamed COMMON variables area before calling a routine. **STACKCOMMON OFF** (the default) causes the unnamed COMMON variables area to be preserved across multiple PERFORM routine calls.

See Also: The MVBasic COMMON and PERFORM statements.

1.125 STAT

The **STAT** command returns the total, average, and count for a numeric attribute.

```
STAT [DICT] filename [dict [dict2 ...] | ALL] [query] [(CHINPYZ]
```

STAT with a *dict* argument returns the total of the values for that attribute, the average value for that attribute, and the count of values that fulfill the *query* criteria.

The optional DICT keyword causes the command to total, average, and count the specified DICT entries in the *filename* dictionary file; otherwise, *filename* is assumed to be a data file.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- (C or COL-HDR-SUPP suppresses both the default page header and the column headers. COL.HDR.SUPP (note two P's) is a synonym for COL-HDR-SUPP.
- (H or HDR-SUPP suppresses the default page header. It does not suppress a page header specified using the HEADING clause. HDR.SUP and SUPP are synonyms for HDR-SUPP.
- (I or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- NI-SUPP suppresses the total item count at the end of the listing. NI.SUP and COUNT.SUP are synonyms for NI-SUPP.
- (N or NOPAGE suppresses the page break prompt. NO.PAGE is a synonym for NOPAGE.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **STAT** operation.

The following example returns the total, average, and count (ENUM) for the specified attribute:

```
STAT VOC F5
```

The VOC contains 26 F5 values that begin with the number 2. **STAT** returns a F5 total of 52 (26 x 2), an AVG F5 of .108786611 (52 / 478), and an ENUM F5 of 478.

The following example returns the same statistics on F5 values that fulfill the*query* criteria, in this case, the criteria that every F5 has a non-null value:

```
STAT VOC F5 WITH F5
```

STAT returns a F5 total of 52 (26 x 2), an AVG F5 of 1.06122449 (52 / 49), and an ENUM F5 of 49.

STAT results can be returned individually using the **COUNT** and **SUM** commands or the AVG, ENUM, and TOTAL CMQL clauses.

See Also: COUNT, SUM

1.126 STATUS

The **STATUS** command displays a table listing the current MultiValue user processes.

```
STATUS
```

For each user process, **STATUS** lists the process ID (pid), the port number, the date and time of initialization of the MV Shell, and the username. The current process is indicated by an asterisk preceding the pid number. If a current terminal process is not running the MultiValue Shell, **STATUS** does not displayed it.

This listing is initiated by displaying a header, and concludes with a count of the number of items listed.

After listing each full page of items, **STATUS** issues a prompt to the user to input a character to display the next page. To terminate listing, input the letter Q at the display prompt. This prompted character is not echoed on the list display, regardless of the MultiValue emulation.

The **STATUS** and **LISTU** commands are functionally identical. The **LISTME** command returns identical information, but only lists processes when in the USER account. To list the current account name and currently executing command for each terminal process, use the **WHERE** command.

See Also: LISTME, LISTU, LOGOFF, WHERE

1.127 SUM

The **SUM** command sums the values of items that satisfy an SQL query.

```
SUM [DICT] filename {field1 [field2 ...] | ALL} [query] [(CHIPYZ]
```

SUM returns a sum for each specified *field* item in *filename*, or the items selected from *filename* by *query*. It returns a [423] message for each *field*: `Total of field is : n`. After returning one or more [423] messages, it returns a [438] message giving the count of the items summed. This is shown in the following example:

```
USER:SUM VOC F1 F5
[423] Total of F1 is : 0
[423] Total of F5 is : 54
[438] 366 Items summed.
```

You must either specify one or more *field* arguments separated by blank spaces, or specify ALL to return totals for all fields.

The optional DICT keyword causes the command to sum the DICT entries in the *filename* dictionary file; otherwise, *filename* is assumed to be a data file.

The optional *query* component can contain one or more Caché MultiValue SQL (CMQL) query clauses. These CMQL clauses can be specified in any order; the order of application of CMQL clauses is independent of the specified order. For further details on CMQL clauses, refer to Caché MultiValue Query Language (CMQL) Reference.

The following are supported letter code and keyword options:

- (C or COL-HDR-SUPP suppresses both the default page header and the column headers. COL.HDR.SUPP (note two P's) is a synonym for COL-HDR-SUPP.
- COUNT.SUP suppresses the `[438] x Items summed` message.
- (H or HDR-SUPP suppresses the default page header. It does not suppress a page header specified using the HEADING clause. HDR.SUP and SUPP are synonyms for HDR-SUPP.
- (I or ID-SUPP suppresses listing the @ID field. ID-SUP and ID.SUP are synonyms for ID-SUPP.
- NI.SUP suppresses the `[438] x Items summed` message.
- (P redirects all output to the STANDARD print queue. The LPTR clause performs the same operation. You can use **LISTPEQS** to view the print queue.
- (Y displays query metadata.
- (Z displays the CMQL Query Execution Plan before performing the **LIST** operation.

When adding *field* values, **SUM** uses the leading numeric portion of a string; for example, `2CPM` is processed as 2. If a *field* value does not begin with a number, it is treated as 0. **SUM** does not check for duplicate names; if you specify a *field* item twice, it will be summed twice.

See Also: COUNT, LIST, SELECT, STAT

1.128 TABS

The **TABS** command sets tab stops.

```
TABS n[,n[,n]]
```

TABS can be used to set any number of tab stops at specified character positions. Multiple *n* arguments can be separated by commas or spaces. **TABS** overwrites any previous tabs settings. To remove all tabs, specify **TABS** with no arguments. If no tabs are set, the tab key advances by a single space.

1.129 TANDEM

The **TANDEM** command allows one MultiValue user to connect to the terminal of another MultiValue user, sharing terminal input and output.

```
TANDEM [ON | OFF | SYSPROG] [(N | (F | (S]
TANDEM port
```

A tandem session consists of a master and a slave terminal. A tandem session begins with the slave terminal.

The slave terminal uses the ON keyword to specify availability to receive a tandem slave request. The slave terminal uses the OFF keyword to end availability to receive a tandem slave request or to end the current tandem session. The `(N` and `(F` letter code options are synonyms for ON and OFF; specify either ON or OFF or `(N` or `(F`, not both.

Once the slave terminal has specified **TANDEM ON** (or **TANDEM (N**), the master terminal can use the *port* argument to specify the terminal they wish to establish as a tandem slave. (You can use the **LISTME** or **LISTU** commands to list the port numbers of current terminal sessions.)

Once established, the tandem session initiates when the slave terminal next presses the ENTER key.

By default, the master terminal and slave terminal can be running in any MV account. You can, however, restrict use of **TANDEM** to a master terminal running in the %SYS namespace (the SYSPROG account). To do so, the slave terminal must specify its availability using either **TANDEM ON (S** or **TANDEM SYSPROG**. This restrict tandem requests to master terminal running in the SYSPROG account.

The master terminal initially enters the tandem session in view-only mode. To quit the session in view-only mode, type "Q". To change the mode, type one of the following: **Esc-F** puts the master in feed mode; **Esc-V** puts the master in view-only mode; **Esc-M** puts both master and slave in message mode; **Esc-X** causes the master to exit the tandem session.

For further details, refer to the TANDEM section of the *Operational Differences between MultiValue and Caché* manual.

See Also: LISTME, LISTU

1.130 TERM

The **TERM** command displays and sets terminal and printer display characteristics.

```
TERM [name]
TERM [#1, #2, #3, #4, #5, #6, #7, #8, #9, #10]
```

TERM with no argument returns the current terminal and printer settings.

TERM name sets the terminal type to *name*. The default terminal type is CACHE. You can display the current terminal type using the MVBasic SYSTEM (7) function. You can display the available terminal types using the CHOOSE.TERM command. **CHOOSE.TERM** can also be used to set the terminal type. You can specify *name* with any combination of letter case: if all letters are specified in lowercase the terminal type is set as specified; if one or more letters are specified in uppercase the terminal type is set in all uppercase. You can optionally specify *name* using a Short Name (a single uppercase letter), as shown in the **CHOOSE.TERM** listing. When setting just the *name* value, it may be preceded or followed by any number of commas; thus **TERM name** and **TERM ,,,,,,,name** are synonymous.

TERM with one or more numeric arguments sets terminal and printer display characteristics using positional argument values. You many specify any subset of settings by including leading commas for unspecified values. Trailing commas are not required. For example, `TERM ,,,,,,100` sets the seventh argument: Printer page width. The unspecified arguments retain their previous values.

The ten positional arguments are as follows:

1	Terminal page width	Maximum line length displayed on the terminal screen. Available values range from 11 to 32767; the default is 79.
2	Terminal page depth	Maximum number of lines displayed on the terminal screen. Available values range from 1 to 32767; the default is 24.
3	Terminal line skip	Number of lines skipped at the bottom of the terminal screen. The default is 0.
4	Line feed delay	Number of seconds to delay following a line feed. Available values range from 0 to 7. Default is no delay.
5	Form feed delay	For terminal: Number of seconds to delay following a form feed. Available values range from 0 to 7. Default is no delay. For printer: character sequence used to paginate to the next page: 0 = use CR/LF to pad to the printer; 1 or above = use FF to pad to the printer. Default is 1.
6	Alternate backspace	ASCII decimal code value corresponding to the backspace character. The default is 8.
7	Printer page width	Maximum number of characters per line for printer output. Available values range from 1 to 32767; the default is 132.
8	Printer page depth	Maximum number lines per page for printer output. Available values range from 1 to 32767; the default is 66.
9	Terminal type	The terminal type. The default is CACHE.
10	Printer line skip	Number of lines skipped at the bottom of the printer page. The default is 0.

Note that positional arguments 3, 4, 6, and 10 are currently not modifiable.

If you specify an out-of-range value, **TERM** returns a [290] message. If you specify an illegal value, **TERM** returns a [202] message.

For further details, refer to the Terminal Output chapter of the *Caché MV Terminal Independence* manual.

See Also: CHOOSE.TERM, COMPILE.TERM, SETPTR

1.131 TERM-TYPE

The **TERM-TYPE** and **TERM** commands are synonyms. Note that only the hyphen form (TERM-TYPE) is valid; the dot form cannot be used for this command.

See Also: TERM

1.132 TIME

The **TIME** command returns the current time and date, or returns the elapsed execution time for a MultiValue command.

```
TIME [command]
```

TIME with no argument returns the current local time and date. For example: “13:23:26 24 MAR 2008”. This date and time format is *not* affected by the **DATE.FORMAT** command. You can use the **DATE** command to return the current date and time.

Caché MultiValue determines local time and date as follows:

- It determines the current Coordinated Universal Time (UTC) from the system clock.
- It adjusts UTC to the local time zone by using the value of the Caché special variable $ZTIMEZONE.
- It applies local time variant settings (such as Daylight Saving Time) for that time zone from the host operating system.

TIME command returns the elapsed execution time (in fractional seconds) for the specified MultiValue command. This is particularly valuable for timing CMQL queries. **TIME command** does not include time spent waiting for an inline prompt (<<...>>). **TIME command** does include time spent waiting for user response to display the next page, regardless of whether **PAGE.MESSAGE** is set to display or omit the “Press any key to continue” page prompt.

For further details, refer to the TIME section of the *Operational Differences between MultiValue and Caché* manual.

See Also: DATE

1.133 TRACE

The **TRACE** command establishes a trace mode that times the execution of each subsequent command.

```
TRACE [ON | OFF]
```

When **TRACE** is on, the invocation of a MultiValue command or a MVBasic statement returns the START date/time (in internal format) before execution and the END date/time upon completion, along with the elapsed execution time in fractional seconds.

To activate this debug behavior for subsequent commands, invoke **TRACE ON**. To inactivate this behavior, invoke **TRACE OFF**. **TRACE** with no argument returns the current trace status (on or off).

You can use **TIME command** to return the elapsed execution time (in fractional seconds) for a single specified MultiValue command.

See Also: TIME

1.134 TRAP-EXCEPTIONS

The **TRAP-EXCEPTIONS** command enables or disables writing of exceptions to the mv.log file.

```
TRAP-EXCEPTIONS [ON | OFF]
```

When **TRAP-EXCEPTIONS** is on, it causes uncaught exceptions to be written to the MultiValue log file, located at Cache/mgr/mv.log. Then the program ends with an MV [EXCEPTION] error message. If the mv.log file grows larger than 5MB (or the size limit of cconsole.log) Caché automatically renames it with the current date suffix as mv.old_YYYYMMDD and initiates a new mv.log file. If mv.old_YYYYMMDD for the current date already exists, Caché appends an integer count suffix mv.old_YYYYMMDD_n to create a unique file name. Refer to the **%SYS.Task.PurgeErrorsAndLogs()** method.

When **TRAP-EXCEPTIONS** is off (the default), an uncaught error writes a Caché error message and terminates the MultiValue execution stack.

TRAP-EXCEPTIONS with no argument returns its current status (on or off).

You can also use the **%SYS.System.WriteToMVLog()** method to write to the mv.log file.

1.135 UNASSIGN

The **UNASSIGN** command deletes the assignment of a form queue spool device to a LPTR device.

```
 UNASSIGN form-queue
```

The *form-queue* can be specified either by name or by number. The default form queue has the name STANDARD, and a form queue number of 0. It can be specified as “STANDARD”, “0”, “F0”, “FN0”, or “FQ0”.

See Also: ASSIGN

1.136 WHERE

The **WHERE** command displays a table listing the current MultiValue user processes, with their current account locations, and currently executing command.

```
WHERE [startval[-endval]] [(DV]
```

By default, **WHERE** lists all running MultiValue processes, including phantom processes. By using the optional *startval* and *endval* arguments you can restrict the list of processes returned to a single process or a specified range of processes (inclusive). No spaces are permitted in the *startval-endval* range; for example, `WHERE 1872-2049` is valid, but `WHERE 1872 - 2049` is not valid.

By default, **WHERE** sorts the list of running MultiValue processes by port number, using *startval* and *endval* to specify a range of port numbers. The `(D` letter code option specifies sorting by Caché process ID (pid) number, using *startval* and *endval* to specify a range of pid numbers.

The `(V` letter code option specifies verbose mode, supplying more information about each process. You can specify either or both letter code option in any order.

For each terminal process running the MultiValue Shell, **WHERE** lists the port number, the device ID (including the pid), the current account name and current namespace name (these are often the same), and the currently executing MultiValue command. The SYSPROG account corresponds to the %SYS namespace.

The active process is indicated by an asterisk preceding the port number. Its current command is always “WHERE”. Other terminal processes indicate the currently executing command (for example, **KEYS** or **SLEEP**); if no MultiValue command is executing, they return “MVShell” as the current command. If a current terminal process is not running the MultiValue Shell, **WHERE** does not displayed it.

See Also: LISTU, STATUS

1.137 WHO

The **WHO** command displays the terminal port number, the account name, and the username.

```
WHO [portnum]
```

WHO with no argument returns the string *portnum account* `from` *name* for the current process. **WHO** *portnum* returns the same string for the process corresponding to the specified port number. If no active MultiValue process corresponds to *portnum*, **WHO** returns the empty string. You can use LISTU to list all active MultiValue processes with their port and process ID (pid) numbers.

You can use **STATUS** to find the process ID (pid) corresponding to the terminal port number. The account name usually identical to the Caché namespace name. However, note that namespace names are not case-sensitive, but account names are case-sensitive. The SYSPROG account corresponds to the %SYS namespace. An account is created using the **CREATE.ACCOUNT** command.

See Also: LISTU, LOGTO, STATUS, WHERE

1.138 Z

The **Z** command prompts for a MultiValue command, then starts a phantom process in which to run the command, not retaining command output.

```
Z [command]
```

If you specify **Z** with no argument, it prompts you for account name, password, and command to execute. The account name is usually the same as the Caché namespace name. The password is ignored. The command is any valid MultiValue command.

The **Z** command does not retain output from the invoked MultiValue command.

Z initiates a phantom (background) process. It does not validate the user-specified command. **Z** returns to the MultiValue Shell with a message specifying the process ID (pid) used for the phantom process.

The **PHANTOM** and **ZH** commands perform similar operations. Both, however, retain and direct *command* output. **PHANTOM** is not interactive; it does not prompt for the command to execute. **ZH** prompts for account name, password, and command to execute.

See Also: PHANTOM, ZH

1.139 ZH

The **ZH** command prompts for a MultiValue command, then starts a phantom process in which to run the command, directing command output to a file.

```
ZH [command]
```

If you specify **ZH** with no argument, it prompts you for account name, password, and command to execute. The account name is usually the same as the Caché namespace name. The password is ignored. The command is any valid MultiValue command.

Normal terminal output does not appear on the screen. By default, *command* output is stored as a record in the &PH& file. In PICK and some other emulations, this output is instead directed to the Spooler file.

ZH initiates a phantom (background) process. It does not validate the user-specified command. If, for example, output from a CEMU command is directed to &PH&, **ZH** displays a message such as: `CEMU_39690_15409 record has been created in the '&PH&' file`. If **ZH** is unable to open the &PH& file (usually because the account name is invalid), it returns a [201] error, and no phantom process is initiated. If the current emulation directs output to Spooler, **ZH** displays the message `Output to spooler`. In all emulations, successful execution of **ZH** then displays the process ID (pid) assigned to the phantom process.

The **PHANTOM** and **Z** commands perform similar operations. **PHANTOM** is not interactive; it does not prompt for the command to execute. **Z** prompts for account name, password, and command to execute, but does not retain *command* output.

See Also: PHANTOM, Z

2

MVIMPORT

Imports accounts from other MultiValue implementations to Caché MultiValue.

```
MVIMPORT backupfile [dirpath [targetpath]] [(code)]
```

2.1 Arguments

backupfile	The file containing the information to restore. This is assumed to be in the native “save” format of the originating system (for example, a UniVerse BACKUP). It can contain data for one or more accounts (namespaces).
dirpath	*Optional* — The location in which to restore certain directory files from a jBASE or UniVerse backup (other MultiValue implementations do not have “stray files”). This path is used to store any directories and files that are otherwise skipped because they do not directly relate to a VOC (“stray files”). MVIMPORT examines the backup and determines the lowest point in the tree of directories (the root) for each group of files. MVIMPORT then imports these subtrees into *dirpath*. If this argument is omitted, the default location for these stray files is platform-dependent. On Windows, MVIMPORT attempts to restore to the original path, so if the backup contains a file K:\stray, MVIMPORT restores them into K:\stray; if there is no K drive, the file goes nowhere and no error is issued. Restoring on a OSX, the file goes into ./~/stray On Windows, when *dirpath* is included, if the original drive exists, MVIMPORT will use it and ignore the *dirpath*.
targetpath	*Optional* — The location to import the account to. If this argument is omitted, import into the MGR directory.
code	*Optional* — A letter code governing the operation of MVIMPORT. Available values are A, C, L, M, O, and R.

MVIMPORT ignores any command line arguments it does not understand and is not prepared to handle.

2.2 Description

The **MVIMPORT** command used to move accounts from other MultiValue implementations to Caché. It does this by taking a backup file created by another MultiValue implementation and restoring it to Caché MultiValue. **MVIMPORT** supports backups created by UniVerse, jBASE, D3, Reality, MVBase, OpenQM and Power95. The features of **MVIMPORT** vary according to the type of backup being restored.

- When importing a D3 backup, **MVIMPORT** uses the D3 file definition to determine the formatting for the @ID record created. This formatting determines whether files are listed left justified or right justified, and defines columnar widths.
- If an imported file is a single level file, **MVIMPORT** always imports it as a single level file, regardless of the emulation setting. See CREATE.FILE for information on single level files.
- Caché MultiValue does not support OpenQM. When importing an OpenQM backup, the Caché MultiValue emulation is automatically changed to UniVerse.

MVIMPORT is designed to be a restoration utility for migrating legacy MultiValue accounts to Caché. Users should become familiar with Caché administrator facilities for backup, restore, and other administrative functions.

Because **MVIMPORT** is designed as a migration aid, it only recognizes programs and data when performing its task. Any indices present in *backupfile* are ignored. Native Caché facilities for indexing need to be used to recreate indices after the import. Caché classes are not created. It is the responsibility of the application developer to create and use Caché classes as needed. You can use PROTOCLASS to create Caché classes corresponding to MultiValue data dictionary definitions.

Note: MultiValue account manipulation requires %Admin_Manage privileges. This is normally associated with the SYSPROG account. For information on the Caché security model, see the Caché Security Administration Guide. For specific information on roles and privileges, please consult the chapter on Roles and the chapter on Privileges and Permissions in the *Caché Security Administration Guide*.

2.2.1 Letter Code Options

The following *code* values are supported:

- A: (Alternate account name) Prompts for an alternate account name for restoration instead of using the account name that was originally backed up. This takes precedence over the R option.
- C: (Case) Forces the account name to upper case. D3 is not case-sensitive; all other MultiValue emulations are case-sensitive. For D3 backups, file names are also forced to upper case.
- L*nn*: (Label) Specifies the tape label size as *nn* bytes. This may be needed when the original backup was to tape which was then copied to an operating system file. Usually this value is “L80”.
- M: (Merge) If the account exists in Caché, and the file exists in the account, **MVIMPORT** overwrites existing records if they are in *backupfile*, and leaves alone existing records that are not in *backupfile*. Any new records and files in *backupfile* are also imported.
- O: (Overwrite) If the account exists in Caché, **MVIMPORT** replaces it. The old account will be deleted, then recreated, and its records imported from *backupfile*.
- R: (Rename) If the account exists in Caché, **MVIMPORT** does not replaces it. Instead, **MVIMPORT** imports a new account with a new name. The new name will be ExistingName_nn where *nn* is a sequence counter.

2.3 Determining the Account Name

The account name is found on the backup from D3, Reality, MVBase, OpenQM and Power95. For a UniVerse backup, the account name is the last directory name in the file path. jBASE does not provide the account name, so the user will be prompted for an account name at the start of the import.

If the account exists, and the M, O, or R options were not specified on the command line, **MVIMPORT** prompts for a response. You can respond with the M, O, or R letter code, which have the same effects as the (M), (O), or (R) command line *code* options. For example:

```
Account 'DBIHELPDESK' already exists.
Enter 'M'erge,'O'verwrite,'R'ename or 'Q'uit:?
```

MVIMPORT creates a new account, namespace, and database for each account in the backup that does not already exist in Caché. All hash files (for example, database files) are restored to Caché globals in the created account. All non-hash files (for example, OS files in a directory) are restored to the home directory of the account, or as specified by the *dirpath* argument. **MVIMPORT** imports all valid directories by default, including empty directories. For details on the relationship between accounts and namespaces, and the naming conventions used for each, refer to "MV Accounts and Caché Namespaces" in *Operational Differences between MultiValue and Caché*.

If **MVIMPORT** is merging accounts using the M option, when duplicate record keys exist in Caché and on the import data stream, the Caché values are overwritten.

MVIMPORT always creates a new account in the default location, as a sub-directory of the Intersystems\Cache\Mgr directory. If you want to create the account in any other location (for example, on another disk), then the procedure is as follows:

1. Prior to the **MVIMPORT**, create a blank MultiValue account with **CREATE-ACCOUNT**, specifying an alternate location. For example:

   ```
   SYSPROG: CREATE-ACCOUNT MYACCOUNT UV E:\data\MYACCOUNT
   ```

2. Run **MVIMPORT** with the (M) option. This enables you to merge the account on the backup with the already created account.

2.4 Errors and Log Files

At the conclusion of a **MVIMPORT** operation (for all backups except UniVerse):

- Caché displays a summary of the number of errors and warnings. **MVIMPORT** saves the details of these errors and warnings and general restore statistics to an external file. It displays the filename of this error log file to the user.
- When **MVIMPORT** fails to create a namespace and/or a database, it displays an informative error message.
- All items found in *backupfile* are restored to the VOC, unless they already existed in the VOC.
- The file IMPORTED.VOC contains all the original VOC contents, regardless of whether or not **MVIMPORT** restored the items to the VOC. This allows you to see what the VOC looked like before **MVIMPORT** made decisions on whether or not to apply items to the VOC.
- **MVIMPORT** looks for object code in D3 backups. While this is discarded on Caché because it cannot be used, its presence indicates which files contain compiled code. This fact is used to mark the file as BASIC source for use with Studio.

3

PROTOCLASS

The **PROTOCLASS** command maps MultiValue dictionary definitions to corresponding Caché class definitions. It is used to import a MultiValue file's data dictionary to Caché. **PROTOCLASS** creates a Caché class for the file, providing the corresponding object representation. **PROTOCLASS** creates and compiles the class definition, It generate both SQL and Object projections of MultiValue data. The dictionary items (or a specified subset of them) are mapped to corresponding Caché class properties.

CAUTION: When creating a Caché class definition for a MultiValue file, it is strongly recommended that every unique attribute have a corresponding class property. Any field/attribute that does not have a corresponding class property will become empty when the **%Save()** method is invoked. (Attributes that are mere synonyms do not require a corresponding class property.)

Caché MVBasic object syntax can be used with an imported MultiValue file *after* the **PROTOCLASS** utility has been run on its dictionary.

PROTOCLASS provides the following types of access to MultiValue data:

- Server access through Caché object syntax: MVBasic, ObjectScript, and Caché Basic.
- Server access through Caché SQL via either Embedded SQL or Dynamic SQL.
- Client access through projections: Java, .NET, C++, Perl, Python, and others.
- Client access through JDBC and ODBC: standard tools, applications.

Note: The name *PROTOCLASS* is meant to emphasize that typically the resulting class definition is a prototype or starting point. Frequently, you will have to run the utility repeatedly, each time making adjustments to the MV dictionaries, until you get your desired result. In addition, you may have to edit the class definition manually. You are free to make copies of **PROTOCLASS** (an MVBasic program) and modify it to suit your own application requirements.

In order to use **PROTOCLASS** you must perform the following steps:

1. Load **PROTOCLASS** into the desired account (namespace).
2. Configure **PROTOCLASS** to specify how the classes should be named.
3. Check the data dictionary with **CHECK.DICT**.
4. Run **PROTOCLASS** against the data dictionary.

3.1 Loading PROTOCLASS

The source code for **PROTOCLASS** is located in <cachesys>\Dev\mv\samples. (In a standard Windows Caché installation, <cachesys> is C:\InterSystems\Cache.) The code is fully customizable and contains extensive documentation describing how to customize it.

Use the Caché MultiValue shell to load **PROTOCLASS** into the current account (namespace).

The following Windows example uses the MV Shell to load PROTOCLASS into the USER account.

```
USER:;file = "C:\InterSystems\Cache\dev\mv\samples\PROTOCLASS.xml"
USER:;stat = "%SYSTEM.OBJ"->Load(file,"c")

Load started on 03/03/2011 12:51:06
Loading file C:\InterSystems\Cache\dev\mv\samples\PROTOCLASS.xml as xml
Creating file BP
Imported document: PROTOCLASS.mvb
Compiling BP PROTOCLASS

Load finished successfully.
USER:
```

After loading, PROTOCLASS appears as a verb in the VOC for the account.

```
USER:LIST.ITEM VOC "PROTOCLASS"


LIST.ITEM VOC "PROTOCLASS"                 01:00:18pm  03 Mar 2011  PAGE    1


     PROTOCLASS
0001 V
0002 PROTOCLASSýUSER
0003 B
0004
0005
0006
0007 L
0008 PROTOCLASS
0009 BP
0010 USER

 One item listed.
USER:
```

3.1.1 Setting Attribute 5

Before using PROTOCLASS, you must set the PROTOCLASS verb attribute 5 to “2D”. You can use the MultiValue ED line editor to set attribute 5, as shown in the following example:

```
USER:ED VOC PROTOCLASS
PROTOCLASS
10 lines long.
----:5
0005:
----:R 2D
0005: 2D
----:FI
"PROTOCLASS" filed in file "VOC".
```

When completed, the VOC entry for PROTOCLASS should look like this:

```
     PROTOCLASS
0001 V
0002 PROTOCLASSýUSER
0003 B
0004
0005 2D
0006
0007 L
0008 PROTOCLASS
0009 BP
0010 USER
```

3.2 Package and Class Naming

Before using PROTOCLASS, you must specify how it is to handle package and class name assignment. The default package and class is MVFILE.dictfilename. Using this default, PROTOCLASS creates classes in package MVFILE and assigns the MultiValue dictionary file name as the class name. For example, MVFILE.PERSON.

You can override this default by setting the dictionary file attribute 5 setting. When setting a package and class, a name to the left of a period is treated as a package name. A name without a period, or a name to the right of a period is treated as a class name. If either the package name or the class name is not specified, PROTOCLASS uses the default for that name.

3.3 Checking the Dictionary with CHECK.DICT

Before running **PROTOCLASS** on your file's dictionary, use the CHECK.DICT command to validate the dictionary entries that you intend to represent in your Caché class. **CHECK.DICT** verifies the following:

- The validity of conversion and correlative codes. Caché is somewhat stricter than many MultiValue platforms with regard to validity of correlative and conversion codes. In addition, there may be some codes that are not supported by Caché. Any such codes that **CHECK.DICT** finds invalid need to be removed or fixed prior to running **PROTOCLASS**.
- The existence of dictionary items referenced by itypes and correlatives in your dictionary. If such items do not exist, then **PROTOCLASS** cannot generate a valid Caché class.

3.4 Running PROTOCLASS

You run **PROTOCLASS** against a MultiValue file using the following syntax:

```
PROTOCLASS filename {item1 [item2 [itemn]] | *} [(D]
```

3.4.1 Arguments

filename	The MultiValue dictionary file to be mapped to a corresponding class.
item	One or more dictionary items to be mapped to class properties. Multiple items are separated by blank spaces.
*	A wildcard specifying that all dictionary items be mapped to class properties.
(D	The optional (D letter code causes **PROTOCLASS** to delete an existing class. The default is to update an existing class.

By default, **PROTOCLASS** creates a class named MVFILE.*filename*, then creates a class property for each specified *item*. If you specified asterisk (*), **PROTOCLASS** creates a class property for each of the items in *filename*. **PROTOCLASS** then compiles the class it has just created. Only in the case of extremely clean dictionaries will **PROTOCLASS** generate properties for all items. Ensuring the dictionary entries are properly marked as Single or Multivalued prevents extraneous "list of" properties.

The resulting class definition includes parameter, property, and index declarations. The property definitions correspond to the attributes defined in the original MultiValue file. Single-value attributes are commonly assigned the Caché %String data type and MultiValue attributes are commonly assigned a list of %String data type strings. A property may be assigned a %MV.Date, %Time, or %MV.Numeric data type, based on the conversion codes in the attribute definition.

3.4.2 Run PROTOCLASS Example

The generation of Caché class definitions using **PROTOCLASS** is best done as an iterative process, rather than as a single bulk conversion of the entire MultiValue file. First run **PROTOCLASS** on the simple attributes (D-type attributes or A and S types without A or F correlatives) that you want represented in Caché. Use Studio to view the resulting Caché property declarations and make any needed modifications. Then run **PROTOCLASS** on a few of the complex attributes. View and modify the results. Repeat until you are done.

The following example runs **PROTOCLASS** against simple attributes of the PERSON file:

```
MyAccount:PROTOCLASS PERSON NAME AGE HAIR PHONE

Processing simple attribute definitions
Creating property called Name from NAME
Creating property called Age from AGE
Creating property called Hair from HAIR
Creating property called Phone from PHONE
Processing computed attribute definitions
Saving the generated class...
Compiling the generated class...

Compilation started on 11/11/2011 09:38:47 with qualifiers 'cfvko3'
Compiling class MVFILE.PERSON
Compiling table MVFILE.PERSON
Compiling routine MVFILE.PERSON.1
Processing MV projection...
MV file name is 'PERSON'
MVREPOPULATE is False, skipping DICT update

Compilation finished successfully in 1.228s.
Class generation and compilation was successful!
```

Which yields the following class definition:

```
Class MVFILE.PERSON Extends (%Persistent, %MV.Adaptor, %XML.Adaptor)
[ ClassType = persistent, Inheritance = right, ProcedureBlock, SqlRowIdPrivate ]
{
Parameter MVAUTOLOCK = 0;
Parameter MVCLEARDICT = 0;
Parameter MVCREATE As BOOLEAN = 0;
Parameter MVFILENAME As STRING = "PERSON";
Parameter MVREPOPULATE = 0;

Property Age As %String(COLLATION = "MVR", MVATTRIBUTE = 1, MVAUTO ="P", MVNAME = "AGE",
     MVPROJECTED = 0, MVTYPE = "D");
Property Hair As %String(COLLATION = "Space", MVATTRIBUTE = 2,MVAUTO = "P", MVNAME = "HAIR",
     MVPROJECTED = 0, MVTYPE = "D");
Property ItemId As %String;
Property Name As %String(COLLATION = "Space", MVATTRIBUTE = 3,MVAUTO = "P", MVNAME = "NAME",
    MVPROJECTED = 0, MVTYPE = "D");
Property Phone As list Of %String(COLLATION = "Space", MVATTRIBUTE = 4, MVAUTO = "P", MVNAME = "PHONE",

    MVPROJECTED = 0, MVTYPE = "D");

Index indexItemId On ItemId [ IdKey, PrimaryKey ];
}
```

Once your Caché class contains properties that adequately represent your simple attributes, you can continue by running PROTOCLASS on your complex attributes (I-types and A or S types with A or F correlatives in attribute 8). For a complex attribute, PROTOCLASS generates two different Caché class members:

- A calculated property. Properties of this type have no stored or in-memory values. Their values are calculated based on the values of other properties. One of the parameters of the property declaration is SqlComputeCode. This parameter identifies code (a Caché method) which calculates the value for the property.
- A method containing the code that calculates a value for the property.

In the following example, PERSON contains an I-type property named ITEST. To add a property to your Caché class to represent ITEST invoke the following command:

```
MyAccount: PROTOCLASS PERSON ITEST
```

PROTOCLASS adds the following property declaration to the class definition. Note that the MVITYPE parameter contains the actual MultiValue code that defines the property. Note also the SqlComputeCode parameter. This parameter contains code that invokes a Caché method that calculates the value of the property.

```
Property Itest As %String(COLLATION = "Space", MVAUTO = "P",MVHEADING = "TYPE-I",MVITYPE = "TYPE;
@1[1,1];
IF @2=""F"" THEN ""FILE""
ELSE IF @2=""V"" THEN ""VERB""
ELSE IF @2=""K"" THEN ""KEYWORD""
ELSE IF @2=""S"" THEN ""MACRO""
ELSE IF @1=""PA"" THEN ""PARAGRAPH""
ELSE IF @1=""PH"" THEN ""PHRASE""
ELSE ""OTHER"";
TYPE:'-':@3",MVNAME = "ITEST", MVPROJECTED = 0, MVTYPE= "I", MVWIDTH = 20)
[ Calculated, SqlComputeCode = {Set {Itest}=##class(MVFILE.PERSON).calcItest({%%ID},{%RECORD})},
  SqlComputed ];
```

Here is the method that PROTOCLASS generates to calculate the property's value:

```
/// Calculates property Itest from the raw ref, using the itype routine
/// IMPORTANT!!! This routine should be recoded to use explicit properties rather than @Record.
ClassMethod calcItest(ItemID As %String, Item As %String) As %String
[ Language = mvbasic ]
  {
  @ID =ItemID
  @RECORD=Item
  MV$ITYPE$1 = @RECORD<1>
  MV$ITYPE$2 = MV$ITYPE$1[1,1]
  MV$ITYPE$3 = IF MV$ITYPE$2 = "F" THEN "FILE"
    ELSE IF MV$ITYPE$2 = "V" THEN "VERB"
    ELSE IF MV$ITYPE$2 = "K" THEN "KEYWORD"
    ELSE IF MV$ITYPE$2 = "S" THEN "MACRO"
    ELSE IF MV$ITYPE$1 = "PA" THEN "PARAGRAPH"
    ELSE IF MV$ITYPE$1 = "PH" THEN "PHRASE"
    ELSE "OTHER"
  MV$ITYPE$4 = @RECORD<1>:'-': MV$ITYPE$3
  RETURN MV$ITYPE$4
  }
```

@RECORD is a system variable. In the above method, it contains the current record as a delimited string. Notice that the automatically generated comments that precede the method definition suggest that you recode the method to use actual property names rather than the @RECORD syntax. This can dramatically simplify the method. In the example, the I-type property ITEST is calculated based on the value of TYPE. The method can be recoded as follows:

```
ClassMethod calcItest(TYPE As %String) As %String
[ Language = mvbasic ]
  {
  IF TYPE = "F" THEN RETURN "FILE"
    ELSE IF TYPE = "V" THEN RETURN "VERB"
    ELSE IF TYPE = "K" THEN RETURN "KEYWORD"
    ELSE IF TYPE = "S" THEN RETURN "MACRO"
    ELSE IF TYPE = "PA" THEN RETURN "PARAGRAPH"
    ELSE IF TYPE = "PH" THEN RETURN "PHRASE"
    ELSE "OTHER"
  }
```

You would then also need to change the SqlComputeCode in the property declaration to the following:

```
SqlComputeCode = {Set {Itest}=##class(MVFILE.PERSON).calcItest({TYPE})}
```

3.4.3 Property Naming

PROTOCLASS creates a class property that corresponds to each specified *item*. Because of the different naming conventions in MultiValue and Caché, the following naming conversions are performed:

- If the first character of *item* is a number, prefix the property name with the letter “P”.
- If the first character of *item* is a letter, capitalize that letter.
- If the first character of *item* is a non-alphanumeric character, delete that character, then follow the naming rules above.
- If *item* contains a non-alphanumeric character, delete that character and capitalize the letter that follows it. Lowercase all other letters except the first letter.
- If *item* does not contains non-alphanumeric characters, retain the letter case as specified in *item*.

3.4.4 MVAUTO Parameter

Each resulting class property contains an MVAUTO parameter. MVAUTO contains a string of one or more letter codes indicating the source of the property definition and whether another property relies on it. MVAUTO prevents **PROTOCLASS** or **CREATE.INDEX** from overwriting any manual modifications that you may have made to a class.

PROTOCLASS sets MVAUTO to one or more of the following letter codes:

Value	Meaning
P	Property defined by PROTOCLASS.
I	Property is defined by or is being used by an index.
R	Another property references this property.
M	The property has been defined through manual intervention. None of the automatic tools will modify it. You must add the M manually.

For further details on index properties, refer to the CREATE.INDEX command.

3.4.5 ItemId Property

The ItemId property of the generated Caché class represents the original MultiValue file’s *item-id* field. The ItemId property has an index defined on it. This index gives ItemId the following characteristics:

- ItemId is an IDKEY and serves as the Caché objects ObjectID. Use ItemId to open objects. When you create a new object, you must assign a unique value to its ItemId property before saving the object.
- ItemId is also the SQL primary key, so each record must have a unique value. Note that the SQL representation also includes the RowID field (named ID). Its values are always identical to the ItemID values.

PROTOCLASS creates a class with a property named ItemId, which describes the item id of the original MultiValue file. You can change the names of other properties in the generated class (assuming that you also change the name anywhere that the property is referenced by other properties, indices, or methods) but the ItemId property must be named ItemId. Otherwise subsequent **CREATE.INDEX** commands will fail and leave the class in an uncompileable state.

3.4.6 MVSVASSOCIATION Parameters

If the MultiValue dictionary definition contains at least 10 lines, **PROTOCLASS** generates MVSVASSOCIATION parameters for the properties when the controlling/dependent attribute is multivalued. This permits subvalue selection. For further details refer to Subvalue Considerations.

3.4.7 dummyAttribute Property

If the dictionary items to be mapped to a class consist entirely of virtual fields, **PROTOCLASS** automatically creates an additional property called dummyAttribute. Virtual fields include calculated fields and transient fields.

The dummyAttribute property is not created if the class contains one or more real (storage) attributes. If PROTOCLASS later adds a real attribute to the class, the dummyAttribute is automatically deleted. If you manually add a real attribute to a class that contains the dummyAttribute property, you should manually delete the dummyAttribute property and associated storage.

4

MultiValue Command Stack Commands and Keystrokes

This chapter provides an alphabetical listing of the command stack commands supported by the Caché MultiValue Shell. Command stack commands begin with a period (.) character. These allow you to edit and execute the command lines on the command line stack. These command names are not case-sensitive.

This chapter also describes the keyboard keystrokes supported by the Caché MultiValue Shell.

4.1 .A

The **.A** command appends text to a command line.

```
.A[n] text
```

The command **.A** (or **.A1**) appends text to the current command line. The command **.An** appends text to a command line earlier in the command stack, with the integer *n* specifying how many previous line to count back. Thus, .A4 appends text to the command line issued four lines ago.

The *text* is appended exactly as specified. Thus a quoted string is appended with the quote characters.

4.2 .C

The **.C** command changes command line text.

```
.C[n] /oldtext/newtext
```

The command **.C** (or **.C1**) replaces the first instance of *oldtext* with *newtext* in the current command line. The command **.Cn** replaces text in a command line earlier in the command stack, with the integer *n* specifying how many previous line to count back. Thus, .C4 replaces *oldtext* with *newtext* in the command line issued four lines ago.

Text search is case-sensitive. To delete *oldtext*, specify `.C /oldtext/`. To replace *oldtext* with a blank space, specify `.C /oldtext/ /`.

4.3 .D

The **.D** command deletes one or more command lines from the command stack.

```
.D[n[-m]]
```

The command **.D** (or **.D1**) deletes the most recent command line from the command stack. The command **.Dn**, with *n* specified as an integer, deletes the *n* most recent command line from the command stack. The command **.Dn-m**, with *n* and *m* specified as integers, deletes the range of lines between *n* and *m* (inclusive).

4.4 .L

The **.L** command lists one or more lines from the command line stack.

```
.L[n]
```

The command **.L** lists all of the command lines in the command line stack. The command **.L1** lists the most recent command line in the command line stack. The command **.Ln**, with *n* specified as an integer, lists the *n* most recent command lines in the command line stack. If *n* is larger than the number of lines on the stack, all lines on the stack are returned.

4.5 .U

The **.U** command converts one or more command lines to uppercase letters.

```
.U[n[-m]]
```

The command **.U** (or **.U1**) converts the most recent command line to uppercase letters. The command **.Un**, with *n* specified as an integer, converts the *n* most recent command line to uppercase. The command **.Un-m**, with *n* and *m* specified as integers, converts the range of lines between *n* and *m* (inclusive) to uppercase letters. This range begins with the *n* command line and ends with the *m* command line. Therefore, to convert the last three command line commands and return them in the original order, you would specify `.U3-1`.

4.6 .X

The **.X** command executes one or more command lines.

```
.X[n[-m]]
```

The command **.X** (or **.X1**) retrieves and executes the most recent command line. The command **.Xn** retrieves and executes a command line earlier in the command stack, with the integer *n* specifying how many previous line to count back. Thus, .X4 executes the command line issued four lines ago. Note that issuing **.X** or **.X1** does not increment the command line count, whereas issuing **.Xn** (where n is >= 2) does increment the command line count by 1. The command **.Xn-m**, with *n* and *m* specified as integers, executes the range of lines between *n* and *m* (inclusive), beginning with the *n* command line and ending with the *m* command line. Therefore, to execute the last three command line commands in the original order, you would specify `.X3-1`.

4.7 .?

The **.?** command displays help text for the command stack commands.

```
.?
```

4.8 Keystrokes

The Caché MultiValue Shell supports the following terminal keystrokes:

- Left Arrow / Right Arrow keys: enable you to move left or right in the current command line.
- Home / End keys: enable you to move to the beginning or the end of the current command line.
- Up Arrow / Down Arrow keys: enable you to move backwards and forwards through the command line history in the command line stack.
- Page Up / Page Down keys: same as Up Arrow / Down Arrow keys.
- Insert key / Ctrl-O: enable you to toggle between text insert mode (the default) and text overwrite mode in the current command line. You can use either the Insert key or Ctrl-O to toggle between these modes within a command line. The MultiValue Shell defaults to insert mode for each new command line.
- Tab key: inserts a single space.
- Escape characters are not included in a command line.

5

Error Messages

This chapter lists the error messages supported by Caché MultiValue.

5.1 Error Codes and Error Messages

Caché MultiValue supports error codes and corresponding error messages. Most error codes are positive integer values, but there are also a small number of alphanumeric string error codes. Error codes are usually displayed enclosed in square brackets. You can use the MVBasic ERRMSG command to display the error message for a given error code. The MVBasic ABORTE and STOPE commands also return these error messages. Square brackets are not used when specifying an error code to these functions.

5.2 Numeric Error Codes

Error messages 1 through 99:

[1] Indexes purged.
[2] Missing a terminating string delimiter (' " \).
[3] " is not a verb.
[5] The dictionary entry for "" has an invalid format.
[11] No Dictionary Attributes Specified.
[12] File " is update protected.
[16] Syntax error in statement ".
[19] Value without an attribute name is illegal.
[20] Maximum number of new context levels exceeded.
[24] The word "" cannot be identified.
[27] Item id " is illegal.
[30] Format error in VOC entry defining verb.
[31] Basic program pointer " missing target routine name.
[32] The keyword definition " is recursive.
[39] An item " already exists in the VOC.
[40] Program " has not been compiled.
[41] Program " cannot be found (Source or bytecode).
[64] Only one item id is permitted in a "WITHIN" type statement.
[71] An illegal connective modifies the word ".
[82] Your system privilege level is not sufficient for this statement.
[86] File reference attempted on file not previously opened.
[87] File i/o via PROC file buffer failed: file not previously opened.
[88] PROC statement has attempted a divide by zero.
[89] File i/o via PROC file buffer failed: binary item encountered.
[93] TAPE NOT ATTACHED!
[94] End of file.
[96] Bot.
[97] Eot.
[99] " Tape records written.

Error messages 100 through 199:

[117] A delete statement must contain either item-ids or selection criteria.
[120] " negative balance not permitted.
[150] Two digit years default to the years 1900 to 1999.
[151] Two digit years are set at the system level to the
dates between: " and ".
[152] Two digit years are set in this process to apply to
dates between: " and ".
[154] The options specified, " serve no purpose alone.
[155] The options specified, " express conflicting intent.
[156] Invalid mask specification.
[157] Account " will be imported into namespace " due to
existing use of default namespace.
[158] An illegal connective of the form " modifies ".
[159] Account " will be imported into existing empty namespace ".
[160] Account " will be restored into namespace " which will be created.
[161] Account " already exists and will not be imported.
[163] Attribute for sort-key missing.
[166] Illegal attribute name ".
[167] Non-numeric amc.
[169] Invalid syntax.
[170] Missing left parenthesis.
[171] Missing right parenthesis.
[172] Missing THEN.
[173] Missing semicolon.
[174] Missing comma.
[175] The Itype code in " has not been compiled.
[176] EVAL must be followed by an expression enclosed in double quotes.
[177] AS must be followed by a new column name.
[178] The EVAL expression " failed to compile.
[179] The keyword " must be followed by an integer.
[180] " Items processed.
[181] " Message(s) sent.
[183] Expected a file path.
[184] Unable to open filepath "
[185] Expected a number.
[186] The DICT item referenced in the N() clause is invalid.
[187] The DICT item referenced by N() not found.
[195] " is not a list.
[196] The SYSPROG account cannot be deleted.
[197] Full file retrieval cannot be specified when using this verb.
[198] Account name?

Error messages 200 through 299:

[200] File name missing.
[201] Unable to open file ".
[202] " not on file.
[203] Item name?
[204] File definition " is missing.
[206] Data section " not found.
[207] FROM or TO clause missing listnumber.
[208] Account " deleted.
[209] List number must be between 0 and 10.
[210] File " is access protected.
[211] Index name missing.
[212] There is no index on "
[213] Backup format not recognized.
[214] MVIMPORT is already running on another port.
[215] Account " not deleted because multiple namespaces are mapped
into database ":
[220] " exited from file ".
[221] " filed in file ".
[222] " deleted from file ".
[223] " exists on file.
[227] The catalog entry points to object from a different file.
Account: , File: , Item:
[228] The target routine " in account " is missing.
[229] " is not a valid account name.
[231] Program " is a Normal catalog entry.
[232] Program " is a Local catalog Entry.
[233] Program " is Globally cataloged.
[234] The routine " is defined as being in namespace "
but there is no account associated with it.
[235] The target routine " is missing.
[238] File " is update protected.
[240] No select list active.
[241] " Cataloged.
[242] " decataloged.
[243] " Cataloged Local.
[244] " Cataloged Global.
[247] Items saved to list ".

[256] Execution time Seconds.
[258] source files failed to compile.
[267] PROC transfer to " cannot be completed.
[268] PROC error, destination of GO not found, at
line: offset: in the statement:
[269] Invalid operand or value used in the PROC statement:
[270] PROC format error at line: offset: in the statement:
[271] PROC error, No target for GO command,
at line: offset: in the statement:
[272] [Account: File: Proc:] PROC compile failed.
[273] PROC error, Invalid operator for pattern match (), at
line: offset: in the statement:
[274] PROC Unsupported command at line: offset: in the statement:
[275] PROC error, Unknown user exit, at line: offset: in the statement:
[276] PROC error, too many labels, at line: offset: in the statement:
[277] PROC error, Duplicate label, at line: offset: in the statement:
[278] PROC Error: Failed to open file containing PROC.
Account: File: Item:
[279] PROC Error: Item containing PROC not found.
Account: File: Item:
[290] The value "" of parameter number "" is not acceptable.
[298] Format error in specifications.

Error messages 300 through 399:

[310] Item is locked by port .
[351] Multiple using clauses in the query.
[352] Multiple using clauses within the macro " in the file "
[353] USING keyword must be followed by a valid filename.
[354] Error in Phrase or Macro "

Error messages 400 through 499:

[401] No items present.
[402] File-definition item " cannot be deleted or overwritten.
[409] The data section " already exists.
[410] A synonym (Q type) file cannot be specified in this statement.
[413] The file name already exists in the master dictionary.
[414] Data section " set to use directory ".
[415] " exists on file.
[417] CreateFile Completed.
[418] Default data section for file " created. Type =
[419] The specified file cannot be cleared or deleted!
[420] Dictionary file deletion cannot be done without
deletion of data first.
[421] DICT for file " created. Type =
[422] Data section " created. Type =
[423] Total of is :
[424] DICT with a data section name is illegal.
[425] USING must be followed by DICT <filename>.
[426] DICT of file " set to use DICT of file ".
[427] Unable to create directory " for file ".
[428] DICT " set to use directory ".
[429] Default Data Section of " set to use directory ".
[430] No items deleted.
[431] One item deleted.
[432] Items deleted.
[433] File " has been cleared.
[434] Expected a path to a directory in the host file system.
[435] The account " in namespace " cannot be opened.
[436] The file " in account " in namespace " cannot be opened.
[437] Added default record '@ID' to 'DICT '.
[438] Items summed.
[440] DICT " Deleted.
[441] Default Data Section " deleted.
[442] Data section " deleted.
[443] VOC entry for file " deleted.
[444] VOC entry for file " updated.
[445] Catalog pointer " needs to be modified to use TCL2.
Edit the VOC item and add '2' to attribute 5.
[446] Class cannot be generated for system file ".
[447] Single Level File " cannot delete the DATA section.
[448] The global " is already in use, operation failed.

Error messages 500 through 699:

None.

Error messages 700 through 799:

[701] Invalid function correlative definition.
[702] missing right bracket (]).
[703] Invalid C conversion/correlative definition :
[704] Invalid P conversion definition :
[706] The file specified in the translate '' does not exist.
[707] Unable to open target file of translate conversion.
[708] Value '' Cannot be translated.
[713] Function correlative data stack empty.
[714] Function correlative data stack overflow.
[723] Format mask exceeds maximum length of 256 characters.
[724] Length conversion exceeds maximum value of 32767.
[725] F correlative exceeds maximum of 200 opcodes.
[726] Invalid substitute code.
[727] Invalid character position in text extract.
[728] Invalid code in format.
[729] Too many ranges.
[730] Trailing characters not recognized as valid in the conversion code.
[731] Attempted to compile A correlative but the DICT was not opened.
[732] Error in conversion code on attribute 7 or 3 of ''.
[733] Error in conversion code on attribute 8 of ''.

Error messages 800 through 899:

[800] Conversion code exceeds size of internal buffer.
[801] Invalid year digits field.
[804] Missing separator character.
[806] Conversion Feature '' is not currently implemented.
[807] Parameters missing.
[808] Invalid Translate code.
[809] Error in conversion code ''
[810] Account '' already exists.
[811] Create account '' failed.
[812] Invalid emulation specified.
[814] Account '' created.
[815] Changing the emulation setting of account SYSPROG is forbidden.
[819] SELECT list number out of range (0-10).
[820] Invalid key , SELECTINFO().
[821] SELECT list parameter is invalid.
[822] Account '' in namespace '' attached.
[823] Long Strings Not Enabled
To run MV applications you must enable long strings.
In the Management Portal:
Select Configuration, Memory and Startup
Check the "Enable Long Strings" box
Apply the changes and RESTART your session.
[824] Trigger event '' is not a valid event name.
Valid event names are:
POSTOPEN, and PRE|POST: READ,INSERT,UPDATE,WRITE,DELETE,CLEAR
[825] Warning - Trigger subroutine '' is not yet CATALOGed.
[826] Trigger routine already installed for event '' on file ''
Use (O option to override existing routine.
[827] Trigger routine '' success.
Installed on file '' for event ''.
[828] event trigger removed from file ''.
[829] No port number specified.
[830] Port is not logged on.
[831] You do not have the authorization to perform this operation.
[832] Port logged off.
[833] A correlative 'N()' failed.
[834] @FILENAME null.
[835] Create of Directory '' failed with error code .
[836] Invalid package name ''.
[837] Invalid class name ''.
[838] Date segment widths are limited to 15 chars.
[839] Date segment text strings are limited to 15 chars.
[840] Open transaction(s) were rolled back.
[841] The DICT does not contain any attribute definitions.
[842] No class or indexes defined for this file.
[843] You do not have the required permission to access the MV shell.
[844] You do not have the permissions needed to access SYSPROG
Use of resource:%Admin_Manage.
[845] You do not have '' permission for required resource ''
to perform this operation.
[846] You do not have the required database access privileges to logon to .
[847] F pointer item '' not copied.
[848] Invalid routine name ''.
[849] Unknown User Exit ''.
[850] Error in conversion code '' at line of .
[851] Source Id: File: Line:
[852] Source Id: File: Line: Account:
[855] Conversion Feature '' is not currently implemented.
[856] Class '' has multiple properties with the MVNAME parameter ''.
[857] Index '' already exists.
[858] CREATE.INDEX is not supported on directory or anode file types.
[859] Index '' is already defined for ''.
[860] Index '' deleted.
[861] No indexes deleted.
[862] The attribute number in property '' does not match
the attribute number in dictionary ''.
[863] The Itype code in property '' does not match the code in dictionary ''.
[864] The data type in property '' does not match the type in dictionary ''.
[865] The correlative code in property '' does not match the code in dictionary ''.
[866] Index(es) created.
[867] Index Build Completed.
[868] Build Index Failed:
[869] The multivalue indicator in property '' does not match
the indicator in dictionary ''.
[870] The multivalue indicator in property '' does not match
the indicator in dictionary ''.
[871] There is no class associated with file ''.
[872] Unable to open compiled class ''.
[873] Index has unknown collation type.
[874] Index type not supported by MVBasic.
[875] The property name '' is already in use.
[876] Itype '' contains system delimiters.
[877] Printer assignment for channel '' cleared.
[878] All printer assignments cleared.

[879] Class '' has the MVAUTOLOCK parameter set.
This prevents this verb from modifying the class.
[880] Printer channel number '' is out of range.

Error messages 900 through 4999:

None.

Error messages 5000 through 5029:

[5000] Item '' in file '' Locked.
[5001] <MVFIO> Directory/File permission error or failure.
Account: File: Item:
[5002] <MVFIO> Failed to open file.
Account: File: Item:
[5003] <MVFIO> Invalid file reference object.
[5004] <MVFIO> Bad VOC entry.
Account: File: Item:
[5005] <MVFIO> End of Select List.
[5006] <MVFIO> Buffer Realloc.
Account: File: Item:
[5007] <MVFIO> Buffer Alloc.
Account: File: Item:
[5008] <MVFIO> Unable to allocate buffer space for item.
Account: File: Item:
[5009] <MVFIO> Item already exists in file.
Account: File: Item:
[5010] <MVFIO> Item not found
Account: File: Item:
[5013] <MVFIO> Unable to access S.ACCOUNT or an entry in it.
[5014] <MVFIO> Unable to create object reference.
[5015] <MVFIO> Attr/Value/Subval not found.
Account: File: Item:
[5016] <MVFIO> Format error in Qpointer.
Account: File: Item:
[5017] <MVFIO> Global for file is missing or corrupted.
Account: File: Item:
[5018] <MVFIO> The file type is not supported.
Account: File: Item:
[5019] <MVFIO> Object is not a selectlist object.
[5020] <MVFIO> Item to be updated not found
Account: File: Item:
[5021] <MVFIO> Global for file has no organization indicator
in the default subscript.
[5022] <MVFIO> Qpointers form an endless loop.
Account: File: Item:
[5025] <MVFIO> Index routine not found
[5026] <MVFIO> SQL error while updating indices
[5028] <MVFIO> Current namespace is not MV enabled
[5029] <MVFIO> Trigger routine not found

Error messages 6000 through 6219 correspond to ObjectScript error codes. See General System Error Messages in *Caché Error Reference* for descriptions of these errors.

Error messages 7000 through 7017:

[7000] The dynamic library '' failed to load with error:
[7001] Failed to resolve function ''. Error:
[7002] CMQL: Internal Error, contact Intersystems support
with the following info:
[7003] CMQL: Premature end of command or syntax error at command end:
[7004] CMQL: Syntax error near '' :
[7005] CMQL: Syntax error at end of command, while looking for '' :
[7006] CMQL: Syntax error near '', while looking for '' :
[7007] CMQL: Syntax error near command end
[7008] CMQL: Malformed query near '' :
[7009] CMQL: Malformed query at command end
[7010] CMQL: Syntax error near '' while looking for one of :
[7011] CMQL: The DICT entry is not defined.
[7012] CMQL: Fatal internal error - invalid query tree.
Please prepare the following information and contact Intersystems support:
o As much detail as possible regarding the CMQL statement
o As much detail as possible regarding the DICT of the file(s) in question
o Provide the information between the == lines below
==========
Exception message :
Recognizer line :
Tree line :
Node description :
==========

[7013] CMQL: Query specified REQUIRE.SELECT but no SELECT list was active.
[7014] CMQL: The query:
Caused an internal code generation error.
Please report this error, providing as much detail as possible,
to Intersystems Support.
[7015] CMQL: Cannot default WITH clause to AND or OR. Use AND WITH or OR WITH.
[7016] CMQL: Keyword " defined in VOC but not valid in CMQL:
[7017] CMQL Internal error: Unknown type code.
Column " Routine type code =

Error messages 7100 through 7140:

[7100] Internal error in Itype tree parser. "
Please report details to Intersystems.
[7103] Unexpected " in itype ".
[7106] Itype syntax error in Dictionary item "
[7107] Itype syntax error in Dictionary item "
[7114] Unexpected " in itype ".
[7115] Missing " in itype ".
[7120] Itype reference to invalid function " in item ".
[7121] Itype incorrect number of parameters to function " in dict item ".
[7122] Reference to undefined attribute " in itype ".
[7123] Recursive loop in itype definitions:
[7124] Class reference " needs to be in quotes in itype ".
[7125] Self reference in itype ".
[7130] Unable to open Dict of TRANS target file ".
[7131] Dict item " not found in TRANS file ".
[7140] itypes compiled.

5.3 Alphanumeric String Error Codes

These error codes must be specified as delimited strings. They are case-sensitive.

[417NS] CreateFile Completed in namespace ".
[B0] Compilation completed.
[B100] Compilation failed.
[I1] I-type compilation failed.
[I2] I-types failed to compile in file .
[I3] Referencing compound I-type that has not yet been compiled in I-type
[I4] Unable to open VOC
[I5] No source for I-type
[I6] I-type function too large
[I7] TOTAL() not yet implemented: used in I-type
[MVENOIQPTR] CREATE-INDEX on Q pointer to account " unsupported.
Use CREATE-INDEX in target account.
[NYI] The feature " in subsystem " is not yet implemented.