Operational Differences between MultiValue and Caché

Version 2017.2
2020-06-25

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

Operational Differences between MultiValue and Caché
Caché Version 2017.2 2020-06-25

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

List of Tables

About This Book

In some cases, the features provided by Caché and those of MultiValue overlap. In these instances, InterSystems generally has chosen to use the native Caché facility rather than duplicate functionality. This document lists the places where those choices have been made, and gives advice to users moving their applications on what must be done to successfully exploit Caché facilities.

This book contains the following sections:

There is also a detailed Table of Contents.

For general information, see *Using InterSystems Documentation*.

1

MV Accounts and Caché Namespaces

Both MV and Caché have the concept of a logical space to hold groups of related programs and data. In MV, this space is called an ACCOUNT; Caché calls it a NAMESPACE. Because of this similarity, it is natural to consider mapping MV accounts to Caché namespaces since this will also provide the easiest access to all the other facilities that Caché provides for MV applications.

Similarity is not identity, however. The rules for forming MV account names differ from those for Caché namespaces. The following describes the differences between them and how those difference are resolved.

- An MV account name could contain any character from the extended ASCII character set.
- Caché namespace names are at least one character long, starting with an alphabetic character or a percent sign, and followed by an arbitrary number of alphanumerics, dashes or underscores.
- In the simplest case, a Caché namespace maps to a Caché database of the same name. Caché database names are between 1 and 30 characters long, can start with an alphabetic character or an underscore. The remaining characters can be alphanumeric, dash, or underscore.

1.1 Converting the Account Name

In transforming the account name for Caché, it is desirable to end up with a result that doesn't alter simple names, that transforms non-conforming account names in an obvious way, and that results in a string acceptable for both the namespace and the database name. To that end, the following algorithm is used to transform an MV account name into the required Caché namespace and database names. AName, NSName, and DBName stand for the MV account name, Caché namespace name and Caché database name, respectively.

1. Start with an empty NSName.
2. Scan AName from left-to-right finding the first alphanumeric character. Make this the first character of NSName.
3. Continuing from the character just found, append all following characters of AName to NSName in the order scanned as long as each is an alphanumeric, dash or underscore.
4. If the resulting NSName is “SYSPROG”, set NSName to “%SYS” (the Caché administrator namespace).
5. If NSName is empty (no suitable characters were found), set NSName to the string “ACCT_NIL”.
6. If NSName is longer than 27 characters, set NSName to the string "ACCT_TRUNC_nnn_1", where nnn is the length of the original account name.
7. Convert NSName to uppercase.

8. Set DBName equal to NSName.

1.2 Resolving Duplicate Namespace Names

When creating a new MV account within Caché, it is possible that the preceding algorithm will result in a namespace name that already exists. In this instance, additional processing is done to make the namespace name unique so that the creation of the new account succeeds.

1. If the name does not ends in an underscore followed by a string of digits, "_1" is appended to the name.
2. Repeatedly do the following until a unique name results:
   a. Extract the string of digits following the last underscore.
   b. Increment the integer formed by this string of digits by one.
   c. Replace the extracted string of digits by the new value.

1.3 Account Name Maps

When creating a new MV account names, for example when importing MV applications, Caché keeps track of the original account name and its resulting namespace name. This map is used to ensure that references in the MV application to other MV account are properly resolved.

When an account name is deleted, it is removed from this map.

1.4 Dictionary Items — Single Versus MultiValued

The basic rule for dictionary entries in Caché is this: unless you are certain that the field will always be single-valued, mark it as multivalued or just leave the M/S indicator blank. This is because setting an entry as single valued allows the query processor to optimize the generated code, and generally this optimized code will not work correctly on multivalues.

1.4.1 I-Types

If you have an I-type dictionary attribute, regardless of whether it is marked single or multivalued, the I-type expression processes the entire data record at once. In the I-type expression, you can choose to use single-valued functions like OCONV, or multivalued functions like OCONVS, so you can control whether multivalues are processed as one string (using OCONV), or as multivalues (using OCONVS).

Note: This processing is independent of whether the attribute is marked as single or multivalued.

After processing the I-type expression, the result is passed through the option conversion in attribute 3 of the dictionary item 1 value at a time, again regardless off single or multivalued identification.

The usage of the attribute varies with whether it is marked as single or multivalued. If the data is multivalued, but the attribute is marked as single-valued, then a comparison against a single value will likely fail. For example:

```
SELECT FILE WITH ATRB = "ABC"
```

where the actual data in ATRB is something like “ABC]DEF” will pass if ATRB is marked multivalued, but will fail if it is marked single-valued, because on a multivalued compare, “ABC” is compared separately against “ABC” and “DEF”, but on a single valued compare, “ABC” is being compared against the entire string “ABC]DEF” and is not equal.

If an attribute is marked as single-valued, then the results of an exploded select or WHEN clause will be different than if it were marked multivalued.

Note: This is true on Caché and all platforms that support I- and D-type dictionary attributes.

1.4.2 A-Types

For A-types, the rules are slightly different. No other platform besides Caché allows an A-type to be marked single-valued. On UniVerse ODBC, A-types can be identified as single-valued, but this has no effect on MultiValue query.

On any platform, when an A-type is multivalued, the correlative on attribute 8 of the dictionary is called repeatedly, once for each value in the data. This is in contrast to an I-type where the I-type expression is only run once. However, as with the I-type, the conversion (attribute 7 in an A-type) is applied to each value.

On Caché, when an A-type is single-valued, the results depend on the type of correlative. For a simple correlative like MCT, the entire data attribute, including all multivalues, is passed through the correlative as one string. If the attribute data is something like ABC]DEF, then the result of the MCT correlative will be Abc]def, as opposed to Abc]Def which would be produced by a multivalued attribute. Then the conversion is applied one value at a time. So a single-valued A-type with a simple correlative will process like a single-valued I-type.

1.4.3 Other Considerations

If the correlative is an A, F or C processing code, then the data will be passed through the correlative once, but each attribute reference in the correlative will only get the first value of the attribute it references. For example if the correlative is F1, and attribute 1 contains “ABC]DEF”, the result of the correlative will be just “ABC”. The second value never gets processed.

So, it is ill-advised to ever use an A, F or C processing code in a single-valued attribute if there's any chance the processed data might be multivalued. This applies not just to the data in the AMC (Attribute 2 of the dictionary item) but also if any of the other attributes the correlative references might be multivalued.

2

Startup Issues

Most MultiValue applications ported to Caché should consider including the following in their login procs.

2.1 Two-Digit Years Date Conversion

The command

```
CENTURY.PIVOT 1930
```

will set the conversion of two-digit years to four-digit years on Caché to be identical to that on most other MultiValue systems. The value 1930 sets the beginning of a hundred-year range (1930–2029) that is used for two-digit year conversion. The Caché default is 1900, which means a year of 08 will be interpreted as 1908.

CENTURY.PIVOT sets a hundred-year range for two-digit year to four-digit year conversion for the current process. To set the hundred-year range default system-wide, invoke one of the methods of the %SYS.Date.SlidingWindow class as part of the **^%ZSTART** Caché startup routine (for details, see the section on "Using the Caché ^%ZSTART and ^%ZSTOP Routines" in *Caché Specialized System Tools and Utilities*).

2.2 Emulation Definition

The emulation setting for an account is persistent. This means that if the emulation is changed via the CEMU command during a session the new setting will remain in effect until it is changed. To assure that sessions always start with a defined emulations, include the verb

```
CEMU CACHE
```

as part of the login prompt. The CEMU verb is described later in this document and in the *Caché MultiValue Commands Reference*. Replace "CACHE" as needed with one of the supported emulations.

2.3 Date Display Format

If the application is using a non-American date format, consider adding the verb, DATE.FORMAT, as in

```
DATE.FORMAT (I)
```

3

File-Related Issues

3.1 Name Mangling

Different operating and applications systems permit different characters to be used in file names. While they all agree on uppercase alphabetic characters and digits, they differ on the rest. The following are the rules Caché MultiValue uses for converting file names containing “special” characters into forms that are accepted across all the platforms it runs on.

Alphanumeric Characters

These become part of the name without conversion.

The Empty String

The resulting filename is “%”.

ANSI Punctuation Characters

Each character listed is converted to the sequence shown (chosen to be easily remembered). Punctuation characters not in the table become part of the name without conversion.

Character	Replacement	Mnemonic
%	%%	
*	%A	Asterisk
\	%B	Backslash
:	%C	Colon
"	%D	Double quote
>	%G	Greater than
<	%L	Less than
.	%P	Period (only used for the leading or trailing character of a file name)
?	%Q	Question mark
/	%S	Slash
|	%V	Vertical bar
(del)	%Z	

ANSI Control Characters

Each of the control characters is converted to its own sequence as shown here:

Character	Replacement
(nul)	%^0
Chars 1 to 26	%^A to %^Z, respectively
Chars 27 to 31	%^1 to %^5, respectively

Latin-1 Characters

The Latin-1 characters shown in the table are converted to their respective sequences. Characters above 159 are printable characters and are left unchanged.

Character	Replacement
128	%_0
Chars 129 to 154	%_A to %_Z, respectively
Chars 155 to 159	%_1 to %_5, respectively

Prohibited Names

Files whose names begin with any of the following sequences followed by a period will be prefixed with %M (Microsoft):

- COM1 through COM9
- LPT1 through LPT9
- CON
- PRN
- AUX
- CLOCK$
- NUL

3.2 ANODE Files and INODE Files

By default, the **CREATE.FILE** command creates INODE files, in which each global node represents one item. An INODE file stores each record in a global node. Therefore, in an INODE file *each record* cannot exceed 3.5 million characters in length. INODE files generally provide superior query and read/write performance.

The **CREATE.FILE** command can also create ANODE files. In an ANODE file, each field of each record is stored as a global node. Therefore, in an ANODE file the limit for *each field in each record* can be 3.5 million characters. ANODE files can be extremely large. ANODE files are especially useful when writing directly to the global. However, maximum string length restrictions may limit the usage of extremely large ANODE files. For example, the MVBasic **READ** command may encounter a maximum string size when reading an extremely large ANODE file.

- CREATE.FILE and CREATE.BFILE can create an ANODE file by specifying the ANODE option. By default, these commands creates both DICT and DATA files as INODE files. If you specify the ANODE option, the DATA file is created as an ANODE file, but the DICT file is still created as an INODE file. You must explicitly specify DICT ANODE to create both DICT and DATA as ANODE files.

- Select lists are stored as ANODE files.
- &SAVELISTS& and &PH& files are created as ANODE files by default. The &HOLD& file can be created as an ANODE file.
- When CMQL is used on an ANODE file, only the first 100 attributes are available. This permits the listing of the files and a sample of the data but prevents CMQL from trying to read in very large items.
- An index *cannot* be an ANODE file.
- You *cannot* create a class on an ANODE file.

3.2.1 Example Showing the Difference Between INODE and ANODE Files

Create an INODE file with:

```
MV:CREATE.FILE CARI
```

Create an ANODE file with:

```
MV:CREATE.FILE CARA ANODE
```

Use ED to enter the same two records in each file, displayed below:

```
MV:CT CARI 1 2

1
0001 1965
0002 Ford
0003 Mustang
0004 Red

2
0001 2008
0002 Austin
0003 Mini Cooper
0004 RedýBlack

MV:CT CARA 1 2

1
0001 1965
0002 Ford
0003 Mustang
0004 Red

 2
0001 2008
0002 Austin
0003 Mini Cooper
0004 RedýBlack
```

If you examine the data on disk, you can see the difference.

For the INODE file, each record is stored in a global node, with fields delimited by field (attribute) marks.

```
MV:[zw ^CARI
^CARI=$lb(0)
^CARI(1)="1965þFordþMustangþRed"
^CARI(2)="2008þAustinþMini CooperþRedýBlack"
```

For the ANODE file, each field of each record is stored in a global node, where the second subscript is the field number.

```
MV:[zw ^CARA
^CARA=$lb(1)
^CARA(1,1)=1965
^CARA(1,2)="Ford"
^CARA(1,3)="Mustang"
^CARA(1,4)="Red"
^CARA(2,1)=2008
^CARA(2,2)="Austin"
^CARA(2,3)="Mini Cooper"
^CARA(2,4)="RedýBlack"
```

In the INODE file, MultiValue field values are delimited by fiedd marks. In the ANODE file, field four with Red and Black shows a value mark (see the section "Dynamic Arrays" in the *Caché Multivalue Basic Reference* for a table of dynamic array delimiter characters.

3.3 Locking Behavior

Caché associates locks with the file variable. This differs from most MultiValue implementations (except UniData) which associate the lock with the file. Locking is affected by the transaction state; see also the section "Transaction Lock Handling" in this book.

For example, consider the following sequence of events:

1. A main program opens a file and locks a record in it.
2. The main program calls a subroutine, passing the file name but not the file variable.
3. The subroutine uses the name to open the file using a local file variable, locks the same record in the file, performs some calculation, releases the lock, and returns to its caller.

When the subroutine returns control to the main program, the lock set by the main program is still in force. It is associated with the file variable in the main program.

In this sequence, however:

1. A main program opens a file.
2. The main program calls a subroutine, passing the file variable as an argument.
3. The subroutine uses the file variable argument to lock a record in the file, performs some calculation, and returns to its caller.

When control returns to the main program, the lock set by the subroutine is still in effect because it is associated with the file variable passed as the argument.

Note: The same principle applies to a single program which opens the same file multiple times using different file variables. Lock operations using one file variable do not affect locks taken using a different file variable. Thus, when a program locks the same item through two different file pointers, the second lock isn't blocked, because it is taken by the same process. (Some MultiValue implementations DO block such a lock.) However, each lock must be separately released.

4

Transaction Lock Handling

Transaction lock handling follows these rules:

1. Any operation that creates or frees a lock during a nest of transactions delays release of the lock until execution of the COMMIT or ROLLBACK that exits the outermost transaction in the nest. Furthermore, any lock that is promoted from SHARED lock type to EXCLUSIVE lock type during a nest of transactions delays any demotion of that lock until the exit from the outermost transaction in the nest.

   This behavior is followed without exception, even in preference to subsequent rules.

2. Numbered or named locks that are created and freed by the MVBasic LOCK and UNLOCK statements are not incremented. You can do any number of `LOCK 2` statements, but a single `UNLOCK 2` statement frees lock number 2.

   None of the file locks or the record locks are incremented.

3. If a FILE or RECORD is locked with either the SHARED or EXCLUSIVE lock type on entry to an outermost transaction, then on exit from the outermost transaction (with either a COMMIT or a ROLLBACK) that FILE or RECORD is still locked with the same lock type. Locks promoted from SHARED to EXCLUSIVE inside the transaction nest are demoted when the nest exits. Locks newly created inside the transaction nest are freed when the nest exits.

   The following exceptions apply to this rule:

   a. If a FILE was CLOSEd during the transaction, then all FILE and RECORD locks on the closed file are released on exit from the outermost transaction.

   b. Executing either `FILEUNLOCK fvar` or `RELEASE fvar` (but not `RELEASE fvar, recid`) frees the file lock AND all record locks on the file referenced by **fvar**. If this occurs during a transaction, the locks are released when the outermost transaction exits.

Note: Delaying unlocking until the exit of the outermost transaction is a feature of the Cache implementation. Most legacy MultiValue implementations delay unlocking only until the exit of the nested transaction level where the lock creation (or promotion) took place.

5

Notes on The MV Shell

This section provides additional information related to interacting with Caché from the MV shell.

5.1 Privilege Requirements

Running the MV Shell requires **%Developer** privileges. 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.

5.2 Issuing MVBasic Commands in the Shell

The MV shell provides the ability to execute MVBasic commands directly by preceding them with a semicolon. For example,

```
; A = 2
; B = 3
; PRINT  "The product is "
; PRINT A*B
```

will result in 6 being echoed to the terminal window.

5.3 Issuing ObjectScript Commands from MV

The MV shell provides a way to issue ObjectScript commands without having to terminate shell operation. The COS command sends the remainder of its line to Caché for execution. For example, at the MV prompt, the command

```
COS WRITE "Hello, World", !
```

will write the expected string to the output device followed by a newline. Since multiple Caché commands may be placed on a line (separated by single blank spaces), a single invocation of COS may execute multiple Caché commands.

The character "[" may be used in place of the "COS".

5.4 Items with Null ID

Items with a "null" id are only partially accessible from the shell. For TCL-II commands, the following will work:

```
CT FILENAME ''
CT FILENAME ' ' 'SOMEITEMID'
```

but the following will not:

```
CT FILENAME 'SOMEITEMID' ''
```

The null id has to be the first id in the list. In addition,

- For CMQL commands, like LIST-ITEM and LIST/SELECT/SORT/REFORMAT, the null item cannot be referenced at all.
- For commands that use the active select list, a null item in the list will not be found.

Note: InterSystems discourages users from creating items with a null id and recommends that the only command they use on such items is DELETE.

6

Verbs

6.1 New Verb — CEMU

A new verb, CEMU, has the syntax:

```
CEMU [<FLAVOR>]
```

where <FLAVOR> is an optional string. If present, must be one of the supported emulations. If absent, **CEMU** prints the current emulation setting.

Note: This permanently changes the mode of the account in which it is executed. Changing the mode of the SYSPROG account is not allowed as this must always be run in CACHE emulation mode.

6.2 Verb — CREATE.INDEX

The CREATE.INDEX verb is currently limited to creating indices on files in the context of the account where the file resides.

Moreover, **CREATE.INDEX** does not support the “NO.NULLS” option. If this option is specified, a warning message is issued. The NO.NULLS clause actually means "don't index empty strings, not NULL values ($c(128) in UniVerse) which makes it confusing.

The NO.NULLS clause is in fact just an attempt to speed up index building where the majority of values are the same. This is not an issue for Caché indices, nor are many duplicates of any single key.

WARNING! Application which depend on not having NULL as one of the values on an index will have to be modified to account for this situation.

6.3 TIME Command Extension

By default the TIME command outputs the current date and time, for example:

```
USER:time
14:02:09 19 OCT 2006
USER:
```

In the MV Shell. it has been enhanced to execute a statement and print the amount of time needed to execute the statement. This is very useful as an ad-hoc way of timing MV commands, particularly CMQL statements, and is similar to the time command on UNIX® shells. For example:

```
USER: time LIST EMPLOYEES WITH ACCENT LIKE ...NORTHERNER...

FIRSTNAME      MIDDLENAME    FAMILYNAME
MARTIN         JAMES         IDLE

Execution time 3.012333 seconds
```

6.4 CHOOSE.TERM Command

The D3 command, CHOOSE.TERM, is available in all emulation modes. It provides the user with a full list of the compiled terminal types and allows one to be chosen as the current terminal. The choice may be by full name (IBM3151), the menu number (6), or the short name, (if it exists (M). or example,

```
USER:CHOOSE-TERM
Terminal Description                  Term Name           Short Name
--------------------                  ---------           ----------
1)   ansi/pc-term compatible with co  ANSI
2)   AT&T 605 80 column 102key keybo  ATT605              Z
3)   cache terminal based on vt220    CACHE
4)   Hazeltine Esprit I,              ESPRIT              E
5)   hewlett-packard generic termina  HP
6)   IBM 3151 display,                IBM3151             M
7)   IBM Personal Computer (no ANSI.  IBM5051
8)   IBM PC/XT running PC/IX,         IBMPC
9)   linux console,                   LINUX
10)  accuterm emulation of Pick PC C  MM-MON
11)  accuterm emulation of Pick PC C  PICKMON
12)  MDC Prism-8,                     PRISM8
13)  MDC Prism-9 in ANSII mode,       PRISM9
14)  adds viewpoint,                  VIEWPOINT           V
15)  dec vt100 (w/advanced video),    VT100
16)  dec vt100 with color             VT100-COLOR
17)  dec vt220,                       VT220
18)  dec vt220 with color,            VT220-COLOR
19)  dec vt320 7 bit terminal,        VT320
20)  dec vt320 7 bit terminal with c  VT320-COLOR
21)  dec vt52,                        VT52
22)  Wyse 120 and 150,                WY120
23)  Wyse 120 and 150,                WY150
24)  Wyse 30,                         WY30
25)  Wyse 50,                         WY50
26)  Wyse 60,                         WY60                W
27)  wyse 75,                         WY75
28)  Wyse 120 and 150,                WYSE120
29)  Wyse 120 and 150,                WYSE150
30)  Wyse 30,                         WYSE30
31)  Wyse 50,                         WYSE50
32)  Wyse 60,                         WYSE60              W
33)  wyse 75,                         WYSE75
34)  xterm terminal emulator (X Wind  XTERM
35)  ansi/pc-term compatible with co  ansi
36)  AT&T 605 80 column 102key keybo  att605              Z
37)  cache terminal based on vt220    cache
38)  Hazeltine Esprit I,              esprit              E
39)  hewlett-packard generic termina  hp
40)  IBM 3151 display,                ibm3151             M
41)  IBM Personal Computer (no ANSI.  ibm5051
42)  IBM PC/XT running PC/IX,         ibmpc
43)  linux console,                   linux
44)  accuterm emulation of Pick PC C  mm-mon
45)  accuterm emulation of Pick PC C  pickmon
46)  MDC Prism-8,                     prism8
47)  MDC Prism-9 in ANSII mode,       prism9
48)  adds viewpoint,                  viewpoint           V
49)  dec vt100 (w/advanced video),    vt100
50)  dec vt100 with color             vt100-color
51)  dec vt220,                       vt220
52)  dec vt220 with color,            vt220-color
53)  dec vt320 7 bit terminal,        vt320
54)  dec vt320 7 bit terminal with c  vt320-color
55)  dec vt52,                        vt52
```

```
56)  Wyse 120 and 150,                wy120
57)  Wyse 120 and 150,                wy150
58)  Wyse 30,                         wy30
59)  Wyse 50,                         wy50
60)  Wyse 60,                         wy60                W
61)  wyse 75,                         wy75
62)  Wyse 120 and 150,                wyse120
63)  Wyse 120 and 150,                wyse150
64)  Wyse 30,                         wyse30
65)  Wyse 50,                         wyse50
66)  Wyse 60,                         wyse60              W
67)  wyse 75,                         wyse75
68)  xterm terminal emulator (X Wind  xterm
Choose number, term name or short name : 1
Terminal type ANSI loaded.
USER:
```

6.5 DATE.FORMAT Command Changes

The DATE.FORMAT command already supports the ON and OFF options to change the format from international and US formats. The (I) and (D) options do the same and were added for compatibility with other MV platforms. The augmented syntax is:

```
DATE.FORMAT [ ON | OFF | INFORM ] {(ID)}
```

where:

- ON - display dates in international format. This is the default.
- OFF - display dates in U.S. format
- INFORM - set @SYSTEM.RETURN.CODE to 0, if dates are to be displayed in U.S. format, or 1, if they are displayed in international format.
- (I) - display dates in international format; same as “ON”.
- (D) - display dates in U.S. format; same as “OFF”.

6.6 SP.COPY

The SP.COPY command has been added to control the placement of spooler jobs in the queue.

6.7 MVI: MVI-to-MVB Cross-Reference

MVBasic programs (with suffix “.MVB”) are compiled into an intermediate form before execution. These intermediate files have the suffix, “.MVI”. Errors reported at runtime identify their locations in terms relative to these MVI files.

The MVI command allows programmers and support personnel to cross-reference line numbers given in terms of the MVI file back to the originating source code, if the source is available.

The syntax of the command is:

```
MVI {MVB.}XXX{.MVI} {lineno}
```

where

- “MVB.” is a literal text string, assumed if not present
- XXX is a hexadecimal module number assigned to the routine when it is compiled
- “.MVI” is a literal text string, assumed if not present
- lineno is the line number in the MVI code where the error was reported

For example, an error reported as

```
<INVALID FILE VARIABLE>+5^MVB.20.mvi
```

can be referenced with the command

```
MVI 20 5
```

and produces (in this example) the following output:

```
Inspecting 'MVB.20' around mvi line 5
================================================
Origin - File : BP' program : 'TEST'

MVI BASIC code
--------------
 00001 READ X FROM FDSC ELSE STOP
 00002 OPEN "MYTEST" TO FDSC ELSE
 00003   ABORT 201,"MYTEST"
 00004 END
>00005 WRITE "TESTER" ON FDSC,"JJJ" ON ERROR
 00006  CRT "IT IS IN ERROR"
 00007 END

Original BASIC code
-------------------
 00002 OPEN "MYTEST" TO FDSC ELSE
 00003   ABORT 201,"MYTEST"
 00004 END
 00005 *
>00006 WRITE "TESTER" ON FDSC,"JJJ" ON ERROR
 00007  CRT "IT IS IN ERROR"
 00008 END
```

6.8 LOCK Commands

6.8.1 LIST.LOCKS

The LIST.LOCKS command lists the system lock table. Locks created by the MVBasic LOCK command are shown as LOCK nnn. Locks created by opening a sequential file are shown as FILE fname. All other locks are shown in their native format, including MVBasic record locks, which are shown on the global that holds the file.

6.8.2 CLEAR.LOCKS

The CLEAR.LOCKS command allows the user to release locks that are held by this process as a result of an MVBasic LOCK command. With an argument nnn, it releases that single LOCK nnn. Without an argument, it release all LOCKS held by the process.

6.9 TANDEM Command

The TANDEM command allows one multi-value user to connect to the terminal of another multi-value user, sharing terminal input and output. It is usually used as a support tool so that support staff can connect to a remote user and view or run their application.

There are two types of TANDEM user, the master and the slave. The master is the user who initiates the request and chooses the user to connect with. The slave is the user who accepts the connection request and it is the slave's terminal input and output that is shared.

Before a TANDEM session can exist, the slave user must issue a **TANDEM ON** command. By running this command, the user is acknowledging that their session may be connected to by a TANDEM master. Because of the way TANDEM is used, it is anticipated the **TANDEM ON** command will usually be run automatically when the user logs on through a logon proc/paragraph.

For security purposes, the TANDEM master user can only be run from the %SYS namespace. In MultiValue terms, the %SYS namespace is usually the more familiar SYSPROG account.

Note: Both the master and slave users should be running the same terminal type. **TANDEM** does not translate terminal specific escape sequences. Hence if the master terminal was a Terminal (VT220) and the slave terminal was a WYSE 50, then none of the escape control sequences would be translated, and the screen of the master might look garbled.

The **TANDEM** command allows the linking of two separate terminals in such a way that any displayed to or entered from either terminal is echoed to the other. The syntax is:

```
TANDEM [ ON | OFF | <nnn>
```

where:

- ON - This is run from the tandem slave to acknowledge they can become a slave user. Without running this, a user cannot be connected to.
- OFF - This is run from the tandem slave to refuse tandem requests.
- nnn - By specifying a port number, you are asking to start a TANDEM session, as the master user. You must be in the %SYS namespace (SYSPROG account).

Note: For legacy compatibility, **TANDEM (N)** and **TANDEM (F)** are the equivalent of **TANDEM ON** and **TANDEM OFF** respectively.

TANDEM Master User

By executing the **TANDEM nnn** command, you are asking to become a TANDEM master user to connect to the TANDEM slave user on port number nnn. Port number nnn must be logged on, and must not already be in a TANDEM session in any way. The TANDEM slave must have previously executed a TANDEM ON command. For example;

```
SYSPROG:TANDEM 23

TANDEM to port 23 in VIEW ONLY mode.

To exit TANDEM enter ESC + "X"
```

Once a master connects to a TANDEM slave, the connection is in VIEW ONLY mode. This means that the master will see the output of the screen of the TANDEM slave, but the keyboards are not connected.

Once a session is created, the TANDEM master user can enter the following keystrokes to control the session:

- ESCAPE+F

This sequences enters the TANDEM master user into FEED mode. This means the keyboards are now connected and anything the master user types at the keyboard is passed to the TANDEM slave user. Hence, the master can directly control the terminal of the slave. While in this mode, all keyboard characters are sent to the slave except for the ESCAPE key sequences. If you want to send ESCAPE to the slave, press ESCAPE twice.

- ESCAPE+X

  This sequence exits the TANDEM session.

- ESCAPE+V

  select

  This sequence enters the TANDEM master into VIEW ONLY mode. The keyboards are disconnected. All keyboard input from the master user is ignored, except for these escape sequences. In VIEW mode the master user will simply view the output of the slave user.

- ESCAPE+M

  This sequences enters both the TANDEM master and the TANDEM slave into message mode. What each user types is sent as a message to the other. The following example shows an instance of this feature.

- ESCAPE+ESCAPE

  While in the FEED mode, if the master user wants to send an ESCAPE character to the slave you must enter the ESCAPE character twice.

- Q

  While in the VIEW ONLY mode, pressing the Q key will quit the session in the same manner as ESCAPE+X.

For example, the following illustrates the activity at the start of a TANDEM session:

Line	Master (Port 22)	Slave (Port 23)
1		Types: “TANDEM ON”
2	Types: TANDEM 23”	
3	Shows: “TANDEM to port 23 in VIEW ONLY mode. Shows: “To exit TANDEM enter ESC+X.	
4	Types: ESCAPE+M Shows: “TANDEM to port 23 in MESSAGE mode”	
5		Shows: “Entering MESSAGE mode from TANDEM user on port 22” Shows: “MSG:”
6	Shows: “MSG: Types: “Hello, how are you?”	

Line	Master (Port 22)	Slave (Port 23)
7		Shows: “From port 22 at Sep 26 2007 13:57:27 Hello, how are you? Shows: “MSG:”
8		Types: “I'm okay but busy”
9	Shows: “MSG:” Shows: “From port 23 at Sep 26 13:57:37” Shows: “I'm okay but busy”	
10	Shows: “MSG:” Types: “Okay, goodbye”	
11		Shows: “MSG:” Shows: “From port 22 at Sep 26 2007 13:57:51 Shows: “Okay, goodbye”
12	Types: ESCAPE+F Shows: “TANDEM to port 23 in FEED mode”	
13		Shows: “MSG:” Shows: “Exiting MESSAGE mode from TANDEM user 22”
—	...	...

Note: When a TANDEM master starts a session, the master must wait until the slave presses RETURN on their keyboard to finish their current terminal input before the session fully takes effect.

Note: The slave *must* have executed the TANDEM ON command prior to this. This has both security and performance implications. The requirement that the slave terminal must declare itself as such means that a master cannot connect to a terminal arbitrarily. Also, executing TANDEM ON will marginally slow all subsequent terminal input/output incurs an additional performance penalty since all i/o involving the terminal does additional checking for available connections.

6.10 Alternate Verb Forms

In Caché, if a user issues a verb that contains dashes in it, and the verb is not found in the VOC, the shell will replace the dashes with periods and attempt to find the transformed name in the VOC. If it is found, it will be used.

All standard verbs are represented in the VOC with periods in the name. This conversion allows them to be accessed in either form: with periods, or with dashes in the name.

7

Phrases

7.1 Attribute Usage By Phrases

In many MultiValue implementations, a phrase only uses attributes 1 and 2. All others are treated as comments. So the VOC entry

```
0001 PH
0002 F1 F2
0003 FN
```

will list only F1 and F2 on most implementations.

The exceptions, among the supported emulations, are Reality and jBASE. Since Caché strives to provide a superset of MultiValue functionality, it performs like this latter set. That is, a phrase may use any or all of its attributes.

Note: An attribute of a phrase whose first character is an asterisk is treated as a comment, that is, it is ignored.

8

MVBasic

There are several differences associated with MultiValue Basic on Caché. The status of individual items in MVBasic can be found in the Caché MultiValue Basic Reference manual.

8.1 Source Files And Studio

MultiValue files containing MVBasic programs will be automatically recognized by Studio, the Caché IDE, if they have a “B” character in attribute 6. This use reduces the amount of information needed in Studio dialog boxes for opening files and also affects the files that are searched when using the Studio Find-in-files feature.

This attribute can be set when the file is created, for example,

```
CREATE-FILE <filename> (B
```

or

```
CREATE-FILE <filename> DIR <dirpath> (B
```

For individual files, the attribute can also be set using ED.

8.1.1 Studio Debugging and MVBasic

The current version of Studio provides a fully-functioning debugger for use with ObjectScript, Caché Basic and MVBasic.

8.2 Duplicate Naming Conflicts

The name of a subroutine or the names of formal parameters in a SUBROUTINE declaration may not be the same as the names of variables declared in named common. for example, an application containing the lines

```
SUBROUTINE MYSUB (VARA, VARB)
COMMON /NCOM/ VARA, CNT, DATE
```

will produce an error message at compile time because the compiler cannot uniquely identify any later reference to *VARA*.

Also, the subroutine name itself (MYSUB) cannot be redefined in the named common. However, the subroutine name is not used internally and can be replaced with something else to remove the error.

8.3 Variables

Caché MultiValue and most other emulations support case-sensitive variable names. D3 emulation, by default, provides variable names that are not case-sensitive. Use of such variables is not advised when interacting with Caché CSP variables, ZEN, and other InterSystems software, all of which uses case-sensitive variables. To make D3 emulation use case-sensitive variables, specify the flag option `$OPTIONS -NO.CASE`.

8.3.1 @DATE And @TIME

The values of @DATE and @TIME are the date and time that the process started respectively, NOT the current date and time of the system.

8.4 Intrinsic Objects

8.4.1 $SYSTEM

MVBasic has an intrinsic object that may be used to obtain system information. The name is **$SYSTEM**. Basic information is available in the *Caché ObjectScript Reference*; further details are found in the %SYSTEM package in the *InterSystems Class Reference*.

8.5 Statements

8.5.1 FOR – NEXT Iteration Variable Restriction

Caché does not permit the iteration variable for a loop to be of the form "%var", "@var", or "^var", nor can it be a property of a Caché class.

8.5.2 RELEASE with no argument

A RELEASE statement with no argument releases all record locks held by the current process that were applied at the current @LEVEL execution level. This is true for all emulations. This differs from native UniData behavior, which releases all locks held by the current process on all levels.

8.6 Expressions

8.6.1 Interpretation Of Apparent Undimensioned Arrays

The compiler will not accept references of the form

```
name (expr)
```

Depending on the format mask, these must be manually converted to one of the two forms:

```
OCONV(name, expr)
FMT(name, expr)
```

8.6.2 Ambiguous References

The compiler will report ambiguous references as an error. For example, a statement of the form:

```
PRINT ON PTR ('ABC':'DEF')'R#20'
```

is ambiguous because it could be interpreted as

- a variable reference, name(expr) mask
- an array reference followed by mask
- a name (expr)mask local variable followed by masked expression

To be accepted it must be rewritten as, for example,

```
X=('ABC':'DEF')'R#20'
PRINT ON PTR X
```

8.6.3 Multiple Subscript Evaluation For Compound Assignment Operators

Caché does not guarantee that the left side of a compound assignment statement will be evaluated only once. For example, consider the sequence of statements

```
deffun otherfunc(a)
dim v(10)
v(otherfunc(1)) += 1
```

Because of the use of the compound assignment, "+=" in the last statement, it can be rewritten during compilation as

```
v(otherfunc(1)) = v(otherfunc(1)) + 1
```

In this rewritten form, the function **otherfunc** will be called twice.

If the expressions used for a subscript has no side effects, that is, its resulting value depends only on its argument values, then this will not be an issue. However, if the expression used for the subscript depends on other values in the environment beyond those passed in as arguments, then the statement may not accomplish its intended effect. For example, the statement sequence

```
dim temp(10)
temp(rnd(10)) += 2
```

will not add 2 to a random value in the array. Instead, in its rewritten form as

```
dim temp(10)
temp(rnd(10)) = temp(rnd(10)) + 2
```

will result in adding the value, 2, to a random element of the array and storing the sum n a second (perhaps different value of the array.

The compound assignment operators that provoke this behavior are: "+=", "–=", "*=", "/=", and ":=".

8.7 Line Continuation in MVBasic

Caché MVBasic does not support a specific line continuation character. You can, however, break a line of code that contains a comma right after the comma. The compiler will attempt to join a line that ends with a comma to the following line. For example, in the sequence

```
COMMON /BASICINFO/ MACHINE, IPADDR, USERNAME,
PASSWORD
```

the comma which ends the first line is sufficient indication that the line is continued. However,

```
LET x = 1 + 2 + 3 +
4 + 5 + 6
```

will result in a parse error.

8.8 Interaction with Caché Classes

8.8.1 Property Manipulation Restriction

In this release, a program cannot assign a value to a property of a Caché class reference using "<>" syntax. Instead of

```
obj -> Name<2,3> = 'abc'
```

use the statement

```
obj -> Name = REPLACE(obj->Name, 2, 3, 'abc')
```

Properties may also not be deleted with the **REMOVE** or **REVREMOVE** statements, or with the **REMOVE()** function.

8.9 $OPTIONS

Normally, the characteristics of the program, such as the emulation to be used, are taken from the account that contains the source file. The $OPTIONS statement is used to alter some of these settings and thereby change the interpretation of the source program.

CAUTION: The $OPTIONS statement should be used with caution. The preferred approach is to work as much as possible within a defined emulation and reserve the use of this statement for very specific needs. Internal documentation within the program justifying the use of the $OPTIONS is also recommended.

Note: The choice of actions selected by $OPTIONS is in effect at the time the program is compiled. For MVBasic statements such as **EXECUTE**, when the statement is processed at runtime the settings used are those of the account, not the $OPTIONS of the containing program.

The BNF for the statement is:

```
$OPTIONS EMU? (FLAG  |  -FLAG  | FLAG=VALUE )*
```

where:

- EMU is the name of a supported emulation.
- FLAG is the name of an emulation flag to alter. The three forms are:
  - "FLAG" turns the flag ON.
  - "–FLAG" turns the flag OFF.
  - "FLAG=VALUE" sets the value associated with the specified flag.

Table 8–1: Available $OPTIONS A — M

Option	Meaning
ACOR.INTSUMS	Indicates whether A correlative arithmetic is integer-only or not.
ACOR.NUMLIT	Indicates if unquoted numbers in A correlatives are treated as numeric values or as references to attributes. ON means they are numeric literals.
ADD.DELIMS	Indicates whether additional delimiters are added to delimited arrays if the last element is already a trailing delimiter. This affects the behavior of functions and statements that extend arrays such as X<-1> = "Attval" and so on. Set to ON to add delimiters anyway.
ARITH.DELIM	Indicates whether arithmetic is delimiter aware or not. When set to ON, Addition = "41":@AM:"3" will return "42":@AM:"3" If not delimiter aware, then a Non-Numeric Zero Used would ensue.
ARITH.NO.WS	Causes spaces and TABS to be ignored when converting string to numerics. Set to ON if leading whitespace should cause a string to remain a string and not be convertible to numeric.
ARRAY.IS.FMT	Indicates that name(expr) is treated as FMT(name,expr) if name is not a DIM'ed array.
ARRAY.RESELECT	Causes override of NO.RESELECT for dynamic arrays.
ASSUNASS.OK	Indicates that the assignment of a variable to another does NOT cause an error if the source variable is Unassigned (ON setting). If set to off, then the assignment of an unassigned variable to anything else causes a runtime error.
ATPTR	Support printer @)-) codes: @(-27 through -33, -47 through -55, -59 through -126, -220 through -239).

Option	Meaning
ATTR.0IS1	UniData treats attribute numbers < 1 in EXTRACT and <> differently than the other MultiValue implementations. Setting this option correctly handles that difference.
BTYPE.PRM	Indicates whether subroutines called from DICT entries pass a parameter which contains the current item being processed, or not.
BYDR.ISDN	Indicates whether the BY 'DR' clause of the LOCATE command should act the same as BY 'DN' (ON setting) or not (OFF setting).
CASE	Indicates that the MVBasic compiler should treat identifiers (variable names) in a case-sensitive fashion when set to ON. This means that a variable *Abc* is NOT considered to be the same variable as *ABC* and *abc*. This is the default mode for everything except D3.
CMD.IN.HDR	Indicates whether a default heading, being the command typed in should be used by the output processor in this emulation mode. That is, LIST MD will be displayed in the heading of the output for that command.
COM.EMPTYSTR	Indicates that unnamed commons should be initialized at program startup to the empty string.
COM.UNASSIGNED	Indicates that unnamed commons should be initialized at program startup to Undefined. Default: UNASSIGNED
COM.ZERO	Indicates that unnamed commons should be initialized at program startup to numeric zero.
CONV.DIV0IS0	Indicates the behavior of division by zero in conversions. If set to ON, division by zero returns zero. If OFF, then return the numerator unchanged.
CONV.ESZERO	Indicates whether conversions involving an empty string "" will treat the string as numeric 0 or an invalid input. For instance, Sequoia-based systems will take "" string as being 0.
COUNT.OVLP	Indicates that COUNT and INDEX should use overlapped counting when multiple characters are allowed for delimiters. So a string “CCCC” when counted for “CC”returns 3 and not 2.
D	Indicates that dates and times in headers or footers are in D2 (fixed length) format rather than variable (D) format.

Option	Meaning
DELIM.MATCH	Indicates whether a trailing delimiter on a MATCH will match the source or not. For example, "T]R]Y]" will match with "Y]" if this option is turned ON.
E	Indicates (ON) that STOP and ABORT statements use the ERRMSG file rather than just print the string they are given.
ELSE.ON.WRITE	Indicates that an ELSE clause is allowed on WRITE statements. This is set ON by default if you are in UniVerse mode, and OFF otherwise.
ENHANCED.MATCH	Indicates whether to use Prime/UniVerse enhanced pattern matching or not. Note that the MATCHFIELD function comes from UniVerse/Prime and always uses this type of pattern matching.
EXEC.AM	Indicates that dynamic arrays passed to EXECUTE/PERFORM will execute each attribute in the array in turn, rather than just the first attribute.
EXTRA.DELIM	Synonym for ADD.DELIMS
F	Indicates that the FOR loop should increment/decrement the index BEFORE instead of AFTER the bounds checking.
FCOR.REVERSE	Indicates whether F correlatives are true reverse Polish or not. ON means to operate the same way as Reality.
FM.DEC.OUTPUT	ON indicates that a format mask that does not specify decimal places will output decimal places anyway as needed.
FMT.HASLEN	Indicates whether the format codes are preceded with a length field or not.
FOLD.DELIM.VM	Changes the behavior of the FOLD function by setting the delimiter character to @VM rather than @FM.
FOLD.LEN	Changes the behavior of the FOLD function by setting a length of less than 1 equal to a length of 1.
FOR.INCR.BEF	Indicates that the FOR loop should increment/decrement the index BEFORE instead of AFTER the bounds checking.
FORMAT.OCONV	Indicates that FMT is allowed to process out conversion codes as format masks. This is here for compiler compatibility; CACHE mode always allows this.

Option	Meaning
FSELECT	Indicates that the BASIC SELECT statement should set the @SELECTED variable to whatever it selects from. This may be a performance issue on some emulations if the selected lists are large.
FULL.DELIM	Indicates whether functions like INDEX and FIELD will use the whole string supplied as the delimiter (ON), or just the first character (OFF).
FULL.LOGICAL.EVALUATION	Specifies that AND and OR operators evaluate both sides of the operation, even if the left hand side evaluates to False.
G	Indicates (ON) that ON GOTO and ON GOSUB statements that receive values that are out of range for the control variable just ignore the GOSUB or GOTO and move on to the next line. If this is set to OFF, then if the value is too small the first branch is taken, and if too big then the last branch is taken.
H	Indicates that a newly installed HEADER from BASIC is printed immediately, rather than just installed ready for the next page break.
HEAD.PRNNN	Indicates whether HEADINGs in queries suppress "nnn Records Listed." message. ON to causes them to be printed.
HEADER.BRK	Indicates that the PIOpen flavor of 'I' and 'P' options for HEADING statements is in use, if set to ON.
HEADER.DATE	Indicates that dates and times in headers or footers are in D2 (fixed length) format rather than variable (D) format.
HEADER.EJECT	Indicates that a newly installed HEADER from BASIC is printed immediately, rather than just installed ready for the next page break.
HUSH.IO	Indicates whether the HUSH command operates on both Input and Output (ON), or just INPUT echoing (OFF).
I	Indicates (ON) that the INPUT statement does not echo data that comes from a DATA statement.
ICONV.BDATE	Indicates whether the conversion processor should try to make some sense of questionable dates such as 31 FEB nnnn (ON), or should treat them as invalid input.
ICONV.RET.ES	Indicates that ICONV should return the empty string and not the original value if the input string was invalid.
IGNORE.EXTRA.LINES	When set, directs the compiler to ignore any lines that occurs after the logical end of the program. Normally, in this situation the compiler will report an error.

Option	Meaning
IM1.TA	Indicates that INPUT X,-1 checks for type ahead (ON) or NOT (OFF).
IMTS.HOURS	Indicates whether ICONV MTS returns hours or minutes when ambiguous. If ON then ICONV(1, "MTS") returns 3600 assuming 1 hour, and OFF returns 60 assuming 1 minute.
IN.FMT.MASK	Indicates that a string that is specified as a format mask in an input: INPUT A,len {,} Fmt is really a format mask with which to format input for display and validate input as it goes rather than a 3 character sequence as per Reality/jBASE.
IN2.SUBSTR	Indicate that a single parameter substring extract [nn] assumes a length of 1 character rather than that position to the end of the string.
INFO.LOCATE	Indicates (ON) that LOCATE uses the PRIME / INFORMATION style syntax instead of the Reality syntax. This is turned on by defaults for INFO, PIOpen and UniData emulation modes.
INFO.MARKS	Indicate that RAISE and LOWER use a limited range of delimiters in the same way as PI/Open does. When set to ON, this flag restricts raise and lower characters to CHAR(252) to CHAR(255). When OFF, the character range translated is CHAR(248) to CHAR(255).
INSERT.0IS1	Indicates (ON) that values supplied to the INSERT function (not the INS BEFORE...) should treat a parameter that is 0 as if it were 1. This is quite a subtle difference as it is only the INSERT function and only on Prime and UniVerse emulations.
ITYPE	Indicates (ON) that we are compiling I-types.
K	Indicates (ON) that the emulations supports negative subscripts in string extractions, as per Reality/ROS.
KEEP.ECOMMON	Indicates whether ENTER "PROG (I" keeps the global common intact or not.
L	Indicates (ON) that LOCATE uses the PRIME / INFORMATION style syntax instead of the Reality syntax.
LENCNV.ONEMAX	Indicates (ON) that the length conversion with a single parameter should treat that as a max length rather that a length to equal.

Option	Meaning
LINE.CONT.BSLASH	Indicate that the backslash character (\) can be used as the line continuation character. Refer to Line Continuation in *Caché MVBasic Reference* for details.
LINE.CONT.VBAR	Indicate that the vertical bar character (|) can be used as the line continuation character. Refer to Line Continuation in *Caché MVBasic Reference* for details.
LIST.KEYS	Indicate whether the COPY command should list the keys it is copying or not.
LOCATE.R83	Synonym for BYDR.ISDN
MASK.DESCALE	Indicates whether output formatting masks should descale or not. When this flag is ON, for say Reality, then a format such as: FMT("987654", "R26") yields 0.98 but when this flag is OFF, it will yield 9876.54.
MATBUILD.UNASSIGNED.ERROR	When this flag is ON, MATBUILD generates an error when it encounters an unassigned array node. When this flag is OFF, MATBUILD treats an unassigned array node as a null string, and returns a dynamic array with the same number of elements as the array dimension. Refer to MATBUILD statement in *Caché MVBasic Reference* for details.
MATREAD.EMPTY	Synonym for PIOPEN.MATREAD
MX.IS.MCDX	Indicates (ON) that MX conversions should be treated as MCDX rather than an ASCII HEX conversion.

Table 8–2: Available $OPTIONS N — Z

Option	Meaning
NCOM.EMPTYSTR	Indicates that named common should be initialized to an empty string when first loaded by a program.
NCOM.UNASSIGNED	Indicates that named common should be initialized to Undefined when first loaded by a program.
NCOM.ZERO	Indicates that named common should be initialized to numeric zero when first loaded by a program.
NO.CASE	Indicates that the MVBasic compiler should treat identifiers (variable names) in a not case-sensitive fashion when set to ON. This means that a variable *Abc* is considered to be the same variable as *ABC* and *abc*. This is the default mode for D3.

Option	Meaning
NO.IMPLICIT.FMT	Indicates (ON) that implicit FMT operator is not allowed, that is expr1 expr2 will give an error, not act like FMT(expr1,expr2)
NO.RESELECT	Indicates (ON) that if select list 0 (the default select list) is already active (has values), then a subsequent SELECT is not performed and select list 0 remains active.
NULL.SORT.ZERO	Indicate whether Empty String "" sorts equally to Zero. This is primarily for UniVerse/Prime.
NUM.SYMS	Indicate whether the numeric symbols . + and - are treated as non-numeric values if they are the only thing in a string. Set ON to reject this interpretation.
O	Indicates that COUNT and INDEX should use overlapped counting when multiple characters are allowed for delimiters. So a string “CCCC” when counted for “CC” returns 3 and not 2.
OCONV.ES	Indicate if the OCONV code should run against empty strings (ON) or not.
OCONV.SCALE	Indicate whether OCONV formatting uses the scale factor or not by default. ON means it does.
ONGO.RANGE	Indicates (ON) that ON GOTO and ON GOSUB statements that receive values that are out of range for the control variable just ignore the GOSUB or GOTO and move on to the next line. If this is set to OFF, then if the value is too small the first branch is taken, and if too big then the last branch is taken.
OP.LSUP	Indicate whether the 'L' option in a BREAK ON clause means that a blank line before the data line is skipped (ON setting) or it means to skip the break line.
OP.PICK.STYLE	Indicate the general style of query output. If set to ON then the Olde Worlde Pick/ROS style output is sent out, otherwise (OFF) the output is like the U2 style.
OP.PAUSE.NULL	Indicates whether the output processor should pause at the end of a page of output if the HEADING statement in BASIC was set to "". ON to pause the output regardless.
OSEQ.CREATE	ON indicates that OPENSEQ should create the target file if it does not already exist.
PAGE0.CHDR	ON indicate that "PAGE 0" clears any existing HEADINGS or FOOTINGS or not.

Option	Meaning
PARA.DATA	ON indicates that DATA stacked for a paragraph is only used by the paragraph and not passed on to any program that the paragraph subsequently executes.
PCLOSE.ALL	Indicates (ON) that the PRINTER close statement closes ALL print channels and not just channel 1.
PHANTOM.TO.SPOOLER	Indicates (ON) that the terminal output from the PHANTOM command, or similar commands (ZH, PH-START) goes to the spooler. If set OFF, then the terminal output goes to the &PH& file.
PICK.CONVERT	Indicates (ON) that conversions from string to numeric, integer, or boolean use the MV/Pick rules like UniVerse and UniData, namely: A) An empty string will convert to 0 in all cases. B) A simple numeric string will convert to number in all cases (that is, with a single + or minus initial character, any number of leading zeros, optionally followed by digits, optionally a decimal point, and optionally more digits) If PICK.CONVERT is OFF, Caché conversion rules are in effect. Strings that have a leading number return that number, and the boolean value is simply whether the resulting number is 0 or non-zero. Multiple leading +/- characters are allowed, and the final sign is based on the # of - characters before the number. If PICK.CONVERT mode is ON, strings that are not a simple numeric string return TRUE if converted to boolean, and give an error message and return 0 if converted to integer or numeric.
PICK.SELECT	Indicates that the default (internal) list is list # 10, not # 0.
PIOPEN.ENTER	Indicates (ON) that ENTER is a synonym for CALL.
PIOPEN.MATREAD	Indicates (ON) that the MATREAD statement will sets all values of the target array to empty strings "" if the record ID is not found.
PROMPT.4IDS	Indicates whether commands assume that not specifying a list of IDs or a '*' for all, assume '*' rather than prompt for a list of IDs.
Q	Indicates whether a READ that fails should set the target variable to the empty string or leave it untouched.
QRY.AND.WITH	Indicates whether queries of the form: SELECT FILE WITH <clause> WITH <clause> are assumed to be AND WITH (ON) or OR WITH (OFF).
QRY.MSUBCALL	Indicate whether a BASIC subroutine called from a DICT should be called for each value and subvalue, or just once.

Option	Meaning
RADIANS	Indicates (ON) that trigonometric function arguments are in radians instead of degrees.
RAW.OUTPUT	Indicates (ON) that terminal output is not automatically translated into printable characters which happens in some emulations.
READ.RETAIN	Indicates whether a READ that fails should set the target variable to the empty string (ON) or leave it untouched.
REAL.SUBSTR	Indicates (ON) that the emulations supports negative subscripts in string extractions, as per Reality/ROS.
REFORMAT.OVR	ON indicates that REFORMAT overwrites existing attributes or not.
RETURNING.AM	Indicates whether a RETURNING clause wants the return codes in attribute delimited form (ON) or not.
RETURNING.CODE	Indicates (ON) that the returning clause on execute returns the value of SYSTEM.RETURN.CODE (UniVerse) rather than the error string.
RJUST.WRAP	Indicate whether right-justified data should wrap in query output. Set to ON it will operate like Prime/UniVerse, OFF like everything else.
RNEXT.EXPL	Indicates (ON) that READNEXT returns exploded values in the ID.
ROS.VIDEO	Indicates whether the @(-) function should support Reality video mode or not (where the codes -128 through -191 are indicative of effects such as BOLD or REVERSE and so on).
ROUND.UP.NEG	Indicates whether negative rounding is up or down. If turned ON, then a number such as -0.5 will round to 0 rather than -1.
RV0.DCOUNT	Indicates that READV on attribute 0 should return the DCOUNT() of the record.
RV0.EXISTS	Indicates that READV on attribute 0 should return 1 if the record exists, and 0 otherwise.
RV0.KEY	Indicates that READV on attribute 0 should return the record key.
S	Indicates (ON) that the SELECT statement behaves as SELECTV.
SELECT.ANY	Indicates (ON) that the SELECT statement behaves as SELECTV.
SMALL.RAISE	Synonym for INFO.MARKS.
SPASS.ALL	Indicate whether SP-ASSIGN assigns to all channels when no specific channel number is indicated.

Option	Meaning
SS.0IS1	Indicates whether zero is treated as 1 within substring specifications. When this option is ON, X[0,6] = "string" means the same things as X[1,6] = "string" When OFF, it means that the "string" is inserted before the first character of X.
SSEL.2ALIST	Indicates if the BASIC SSELECT statement should install the results as the active select list or not (for return from a BASIC program). You have the BASIC default select list and the system active select list. They can be the same on some systems and this flag indicates whether BASIC SSELECT affects the active select list (ON) or not.
STACK.GLOBAL	Indicates that stacked data acts as in UniVerse and Prime and is global to everything (ON). If set to OFF, then the DATA stack acts like Pick/Reality. The default is on for Cache, UniVerse (all flavors), and UniData; and off for all others such as jBASE, D3, MVBase, Ultimate, Reality.
STATIC.DIM	Indicate whether REDIM of arrays at runtime is supported or not.
STOP.MSG	Indicates (ON) that STOP and ABORT statements use the ERRMSG file rather than just printing the string they are given.
STR.ONEISONE	Synonym for IN2.SUBSTR
SUPP.DATA.ECHO	Indicates (ON) that the INPUT statement does not echo data that comes from a DATA statement.
SYS11.BOOL	ON indicates that SYSTEM(11) returns a binary 1 or 0 (ON setting) if a select list active or not. OFF indicates that SYSTEM(11) returns the number of elements selected to the active list or 0 if no list is active.
T	Indicates that a single parameter substring extract [nn] assumes a length of 1 character rather than that position to the end of the string.
TCONV.REPSYS	Indicates whether Tfile Conversions replace system delimiters. When ON, system delimiters are replaced by the blank character. The default is ON for Caché and all emulations except UniData, which defaults to OFF.
TD.2SPC	OFF indicates this system only returns a result separated by a single space character, or uses the more common 2 spaces (ON).
TRANS.REPSYS	On indicates that TRANS() function in ITYPES replaces delimiters with spaces or not.

Option	Meaning
U	Indicates (ON) that if select list 0 (default select list) is already active (has values), then a subsequent SELECT is not performed and select list 0 remains active.
UCASE.DATE	Indicates whether date conversions force the output text to uppercase or not: For example, if this flag is ON. July 14 1964 will be JULY 14 1964.
V	Indicates (ON) that the operators “*”, “/”, “+”, “-” should be compiled as vector arithmetic.
VAR.SELECT	Indicates (ON) that the SELECT statement behaves as SELECTV.
VEC.MATH	Indicates (ON) that the operators “+”, “-”, “*”, “/”, and “**” should be compiled as vector arithmetic. Refer to Arithmetic Operators in *Caché MVBasic Reference* for details.
W	Synonym for ADD.DELIMS
WHO1	Indicates the style of output for WHO and related user exits should be: PID/PORT ACCOUNT.
WHO2	Indicates the style of output for WHO and related user exits should be: PID/PORT ACCOUNT From systemloginname.
WHO3	Indicates the style of output for WHO and related user exits should be: PID/PORT ACCOUNT (systemloginname).
WRITE.LOCK.WAIT	Indicates that this emulation requires WRITE to wait on a lock held by someone else. ON means to wait for the lock.
X	Indicates (ON) that READNEXT returns exploded values in the ID.
Z	Indicates (ON) that the PRINTER close statement closes ALL print channels and not just channel 1.

8.10 PI/Open Compatibility Subroutines

Caché MVBasic does not include the subroutines for PI/Open compatibility (for example, **CALL !ADDS** and following) that are listed in the UniVerse Basic manual.

8.11 Soundex Differences

Given the same input, the SOUNDEX() function occasionally yields different results among legacy MultiValue platforms. This is also true of Caché. Any data fields whose values are based on a **SOUNDEX()** computed value should be regenerated on Caché as part of the migration of the application.

8.12 Conversion Of UniData SUBR Usage

When any of the subroutines of the form

```
SUBR('-XXX',  args)
```

is used in I-types instead of the built-in names for the corresponding MultiValue function, that is,

```
XXX(args)
```

then it is converted to the built-in form at I-type compilation time. The functions for which this is done are:

- ADDS, ANDS
- CATS, CHARS, COUNTS
- DIVS
- EQS
- FIELDS, FMTS
- GES, GTS
- ICONVS, IFS, INDEXS
- LENS, LES, LTS
- MODS, MULS
- NES, NOTS, NUMS
- OCONVS, ORS
- SEQS, SPACES, SPLICE, STRS, SUBS, SUBSTRINGS

8.13 SYSTEM(100)

The **SYSTEM()** function provides numerous configuration and status values, many of which are standard to MultiValue implementations, a few of which are specific to Caché MultiValue. The SYSTEM() function, with its various code values, is described in the *Caché MultiValue Basic Reference*.

The **SYSTEM(100)** code returns the system ID. The format of the returned information varies according to the platform and the emulation, as shown in the following table:

Field	Windows Platform	Non-Windows Platform
1	System name	System name
2	Name of configuration file: the contents of the “name” statement in the configuration file	Operating system name, for example, “AIX®”
3	Current release level	Name of configuration file: the contents of the “name” statement in the configuration file
4	Current version level	UNIX® system release
5	Type of hardware	UNIX® system version
6	The date of the release	Machine hardware name, or serial number
7	Not used	Once, the release level of the monitor code; now not used; value: “ ”
8	Not used	Once, the date of the boot monitor release; now not used; value: “ ”
9 †	Caché configuration name	Caché configuration name
10 †	Operating system name	Operating system name
11 †	Hardware serial number	Hardware serial number

† This field is provided by Caché for cross-platform consistency.

9

Query Language Differences

There are several differences in how queries are interpreted and executed in Caché. They are detailed in this section.

9.1 Print Limiter Flags

Caché MV supports all the variants of MV query syntax and auto configures itself for the syntax implied by the emulation of the account from whence the query is issued. The emulation flag NO_PRINT_LIMITERS is predefined for all the emulation styles available to Caché MV-enabled accounts/namespaces (via the CEMU command) and affects whether Pick style print limiters are allowed in queries.

A print limiter does not affect the records selected from a file, only whether the value of a particular output column is displayed or not. In traditional Pick mode, this means that a listing such as:

```
LIST MD F1="K"
```

means list all the items in the MD file, displaying the DICT entry called F1, but only print the values of F1 if they are equal to the literal "K".

In Prime/UniVerse accounts however, this same query means something completely different because print limit is not a supported concept. In accounts requiring Prime/UniVerse style queries, the same command means:

List the item in the file MD whose item ID is "K", list the value(s) of DICT entry F1.

9.2 Default Assumptions For AND {WITH} And OR {WITH}

CMQL statements default the AND or OR operator according to the settings of the emulation for the current account. Accounts operating in UniVerse or PRIME modes will cause CMQL statements that omit AND or OR to default to AND. Accounts operating in PICK mode will cause AND or OR omissions to default to OR. Thus the statement:

```
LIST VOC WITH F1 = "K" F1 WITH F3 = "C"
```

will be interpreted in UniVerse or PRIME modes as:

```
LIST VOC WITH F1 = "K" AND F1 WITH F3 = "C"
```

while in a Pick style account, it will be understood as:

```
LIST VOC WITH F1 = "K" OR F1 WITH F3 = "C"
```

Because of the emulation-dependent nature of the interpretation, InterSystems recommends that developers be explicit with the query syntax and put in the AND and OR everywhere, rather than default them.

Value selections, as in:

```
WITH F1 = "K" "L" "N"
```

are ALWAYS assumed to have OR clauses in ALL emulation modes.

9.3 Use Of PICK Wildcard Characters

The use of the traditional wild card characters in a CMQL query is dependent on the emulation in effect at the time the query is issued.

If an emulation such as PICK, jBASE, R83, and so on is in effect, then the use of [ ] and ^ in a comparison string is allowed. However in Prime/UniVerse style emulations, they are not allowed and have no significant meaning.

The LIKE and UNLIKE clauses are always allowed in all emulations.

9.4 USING Clause Position In Query

When issuing a CMQL query, some emulations permitted the USING clause before the filename, for example:

```
LIST USING DICT ANOTHER.FILE CUSTOMER NAME, ADDRESS
```

In Caché, all emulations require any USING clause to follow the data file name, as in

```
LIST CUSTOMER USING DICT ANOTHER.FILE NAME, ADDRESS
```

9.5 CMQL Output Redirection

CMQL statements can be directed to other than the default printer channel using the syntax:

```
LPTR nnn
```

where nnn is optional and 0 or absence indicates the default channel.

10

Terminal Independence

Terminal and keyboard independence refers to the ability of an application to communicate with a user's output device (terminal) and input device (keyboard) without regard to the manufacturer of the device. The control sequences required to manipulate various aspects of the output device (e.g. move cursor, add bold, underscore etc.) varies among manufacturers. Similarly, the sequence of characters generated by a keyboard when certain non-alpha keys are depressed (for example. the function keys, cursor up key etc.) also varies. For MultiValue applications, help exists from the language to provide independence of this variance among devices. These mechanisms, through the use of classes, are also available to other Caché applications.

For further information, please see the Terminal Independence information.

11

Spooling

This version of MultiValue includes an implementation of a spooler that interfaces with the native Caché spooler for interoperability. A description of the spooler and its use by programs and administrators is found in The Caché MultiValue Spooler manual.

12

MVIMPORT

The **MVIMPORT** is the preferred mechanism for moving data from other MultiValue implementations to Caché. At present, **MVIMPORT** accepts supports backups created by UniVerse, jBASE, D3, Reality and MVBase. The features of MVIMPORT vary according to the type of backup being restoreD. The detailed description of MVIMPORT is located in the *MultiValue Commands Reference*.

MVIMPORT is designed as a migration aid. Thus,

- It will only recognize programs and data when performing its task. Any indices present in the import file will be ignored; native Caché facilities for indexing will be used instead.
- It will not create any of the underlying Caché classes needed by running programs. These will be created as needed when the programs are prepared for execution.
- It is not designed for general-purpose backup; only for restore (migration to Caché). Once the information has been migrated, administrators should use the Caché operator commands to save and restore data.

At the end of **MVIMPORT** (for all backups except UniVerse),

1. Caché displays a summary of the number of errors and warnings. The detail of these errors and warnings and general restore statistics are saved to an external file and this filename is displayed to the user.
2. When **MVIMPORT** fails to create a namespace and/or a database, it now displays an informative error message.
3. All items found in the backup are now restored to the VOC unless they already exist in the VOC.
4. The file IMPORTED.VOC contains all the original VOC, regardless of whether or not the items were restored to the VOC. This allows a customer to see what the VOC looked like on their original system before MVIMPORT made decisions on whether or not to apply the items to the VOC.
5. **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.

13

Port Numbers

For compatibility with legacy MultiValue systems, Caché labels each process with two separate identifiers: the process number, and the port number.

The process number is usually a 32-bit integer assigned by the operating system which cannot be changed by the user. Legacy MultiValue applications often expect a process identifier in a smaller range (for example, from 0 to the slightly above the number of licensed processes) and will have problems using a process id that varies over a 32-bit range. Additionally, MultiValue applications often expect a "nailed port number" where connections from a particular terminal or device always get the same identifier, or where the same program will operate differently based on what port or range of ports it runs on. So Caché provides both a process number which is fixed by the operating system, and a port number which can be set by the application.

Process Number

The process number is the operating system process id. It can be retrieved using the ObjectScript $J system variable $JOB. In MVBasic, the @USERNO system variable (@USER.NO is a synonym for @USERNO) returns this value. The process number never changes during the lifetime of the process.

Port Number

The port number is initially the position of the process's entry in the system process table, but can be changed. The port number can be retrieved using the MVBasic system variable @PORTNO (or synonym @UDTNO). The port number is also the first part of the strings returned by the MV shell WHO command, and by the user exit U50BB.

The port number can be changed using the setPortNumber() method in the %SYSTEM.MV class. This changes the value of @PORTNO and the results from WHO and user exit U50BB.

The MV shell commands LISTU, JOBS, LIST-JOB, LISTME, and WHERE all display both the port and the process number.

13.1 UniData

Under this emulation, @USERNO returns the operating system process id, @UDTNO returns the process table number, and the U50BB user exit returns the same as @UDTNO. No changes should be necessary when porting programs from UniData to Caché. The UniData WHERE and WHO commands are fundamentally different from those in Caché, so some attention to detail is needed there.

13.2 UniVerse

In UniVerse, @USERNO returns the operating system process id, as do the WHO command and U50BB. There is no equivalent for @PORTNO in UniVerse. In Caché, @USERNO is the process id, but the WHO command returns the port number, so ported UniVerse applications may want to set the port number to match the process number at login.

13.3 jBASE

In jBASE, @USERNO, @UDTNO and WHO all return the port number. When porting to Caché, references to @USERNO should be changed to @PORTNO.

13.4 D3 And Other PICK Versions

In D3, WHO, and U50BB return the port number as in Caché, but @USERNO returns the port number instead of the process number. When porting D3 programs to Caché, references to @USERNO should be changed to @PORTNO.

14

Other Compatibility Issues

14.1 General

14.1.1 Dates with Caché and MultiValue

The Caché epoch is January 1, 1840, which differs from the typical MultiValue epoch. Caché comes with a complete set of functions for handling dates. These include $ZDate, $ZDateTime, $ZDateH, and $ZDateTimeH.

14.1.2 Locales Other Than Latin-1

MultiValue assumes a character set where the numeric values of the characters range from 0 to 255, inclusive. This is defined in ISO-8859–1 and is often referred to as “Latin-1”.

Caché supports ISO-8859–1 directly, but it also supports a number of other locales. These include single-byte character sets like French, Czech and Finnish, and multi-byte sets like Japanese, Korean, and Thai.

If you install Caché and choose a locale other than Latin-1, you are inherently accepting the Caché rules for that locale, for example, which character is used for the decimal point and the names of months. Refer to “Supported Languages” in the online *InterSystems Supported Platforms* document for this release for a list of supported languages. If you wish to establish a new locale, see the section on “System Classes for National Language Support” in *Caché Specialized System Tools and Utilities*.

14.1.3 Right-Justified Sort Ordering

The various MultiValue systems sort right-justified fields differently from one another. For example, consider the following set of three-character strings: “0A”, “1”, “01”, “1A”, “1.2”, “01.2”, “2”, “02”, “12A”. When sorted as right-justified values, the various platform results are:

- UniVerse Ideal

  0A, 01, 1, 01.2, 1.2, 1A, 2, 02, 12A

- Reality

  0A, 01, 1, 1A, 01.2, 1.2, 02, 2, 12A

- D3, jBASE

  0A, 01, 1, 01.2, 1.2, 1A, 02, 2, 12A

- UniVerse PICK

0A, 1, 01, 1.2, 01.2, 1A, 02, 2, 12A

- Caché

  0A, 1, 01, 1A, 1.2, 01.2, 2, 02, 12A

14.1.4 Limitations on Strings, Arrays and Dates

The current implementation sets these limits:

- The maximum number of characters allowed in a string is 3,641,144.
- The maximum number of elements allowed in an array is 16,381 regardless of the number of dimensions.
- Valid dates must be in the range 12/31/1840 to 12/31/9999, inclusive. These are day numbers -46,385 to 2,933,628) respectively.
- MultiValue programs should only use default date formats that display months as numbers (2– or 4–digits), and not as month names, even though Caché allows the latter.

14.1.5 Case-Sensitivity

Caché is case-sensitive. D3 is not case-sensitive. MultiValue applications using D3 emulation, or those that have turned off case sensitivity via by the $OPTIONS -CASE statement, will need to make sure that all such variables in Caché are declared as uppercase. Otherwise, they will be inaccessible by MultiValue applications. Some of the possible situations where case may be important are:

- The names of globals are always case-sensitive regardless of the setting of $OPTIONS -CASE.
- Basic command syntax such as CRT is not case-sensitive in all emulations.
- Variable names are not case-sensitive in D3, and if you use $OPTIONS -CASE in other emulations.
- Strings are always case-sensitive in the Caché environment. In the D3 emulation mode (and possibly others depending on $OPTIONS CASE), they are not case-sensitive.
- Executed commands may or may not be case-sensitive depending on environment settings, but literals in queries are always sensitive.
- Called subroutine names are case-sensitive.

Note: The last item may change is the near future. That is, called subroutine names may be treated as not case-sensitive.

14.1.6 Literals In PQN PROCs

Caché will report an error if a literal is not enclosed in quotes. This behavior is not uniform across all MultiValue platforms: UniVerse, for example, requires the quotes while Reality does not. Therefore, if you are importing PROCS from Reality, Caché will report an error when attempting to compile the PROC. To correct the error, simply change the code to add quotes around the literal.

However, if when moving procs from jBASE, and the compiler reports this sort of error, additional analysis is needed. If the PROCs originally came from Reality, there may be a silent bug in the code when running on jBASE, because jBASE ignores the literal, and actually uses the empty string. If your procs were written for jBASE, or you expect that behavior, then you need to change the literal to “”.

For example:

```
MV %1 0
```

should be changed to either:

```
MV %1 "0"
```

or

```
MV %1 ""
```

depending on what platform you are porting from, and the desired behavior.

Note: This applies only to those systems (beside Caché) that support PQN PROC: UniVerse, jBASE, Reality, and MVBase.

14.1.7 Execution Levels

Caché allows 30 levels of nested execution. Attempts to exceed this limit cause Caché to report an error and return to the previous level.

14.1.8 The Caché Debugger

When using the Caché to debug MultiValue programs, quitting from the debugger exits the MV shell. In doing so, it does not run any associated ON.ABORT or ON.ERROR exits.

14.1.9 Failures During CALL or ENTER

A failure during a CALL or ENTER aborts back to the TCL with no recovery. On some MultiValue systems, it may or not be possible to gain control (for example, in the debugger) when this situation occurs.

14.1.10 Verb Differences

14.1.10.1 CENTURY.PIVOT

Caché MultiValue supports the verb, but does not provide an equivalent MVBasic function. If an application needs this, the same functionality can be had by,

```
EXECUTE "CENTURY.PIVOT"
```

CENTURY.PIVOT is used to guide 2–digit to 4–digit year conversions. The command

```
CENTURY.PIVOT 1950
```

means that any two-digit year equal to or greater than 50 should be assumed to be prefixed with 19, and any two digit year less than 50 should be assumed to lie in the next century. Most MultiValue applications should use the command

```
CENTURY.PIVOT 1930
```

to duplicate under Caché the conversion of two-digit years they find on other MultiValue systems.

14.1.10.2 CREATE-FILE

On other systems, the parameters “mod” and “sep” are used to size the file being created. Caché stores files in globals and these parameters are not needed. Ignoring them, rather than indicating an error or warning, removes the need to change every CREATE-FILE during conversion to Caché.

14.1.10.3 DELETE-ACOUNT

Caché restricts the DELETE-ACCOUNT verb to be run from the SYSPROG account. Therefore, accounts may only be deleted by those having administrative privilege.

14.1.10.4 SH And DOS

In UniVerse, SH and DOS are VOC entries wrapping the OS commands "sh" and "cmd.exe", respectively. All the command line parameters for those commands (and other OS commands entered this way) are available. This is very system-specific.

In Caché, SH and DOS run an internal program with limited option passing, which also handles terminal I/O in such a way that interactive programs can't be shelled. For example, the command:

```
SH -c "vi .profile"
```

does not function in Caché the way UniVerse users on UNIX® platforms would expect. The equivalent for Caché is

```
$ZF(-1,"vi .profile")
```

See the Caché ObjectScript Reference for more detail.

14.1.10.5 SUM And STAT

The output formats for these verbs differ among the various implementations of MultiValue. Caché produces the output for these verbs in LIST format.

14.1.11 Select Lists

14.1.11.1 Maximum Size

There is no maximum size for a select list. However, operations that involve conversions between a select list and a string are constrained by the Caché maximum string size. It is strongly recommended that you configure long strings when using Caché MultiValue. The present implementation of long strings limits a string to 3.6 million characters. Select list operations that are constrained or unconstrained by this maximum string size are as follows:

- Unconstrained: SELECT, SAVE.LIST, GET.LIST commands. The MVBasic OPEN, SELECT, and READNEXT statements.
- Constrained: EDIT.LIST command. The MVBasic READLIST and WRITELIST statements.

Commands that treat the &SAVEDLISTS& file as a normal file are also constrained by this maximum string size. For example, `SELECT &SAVEDLISTS& WITH @ID LIKE "FP..."` is not constrained, because it only uses the item id, but `LIST.ITEM &SAVEDLISTS& LISTNAME` is constrained. If `LISTNAME` is more than 3.6 million characters, LIST.ITEM returns a truncated listing.

14.1.11.2 Representation

Select lists are stored as ANODES in order to support large list sizes.

14.1.12 MultiValue Attribute Defaults For Dictionary Items

For UniVerse-style dictionaries (also used by UniData and Prime Information Systems), "D" type items (attribute 1) use attribute 6 as a MultiValue indicator. The possible values are S(ingle value) or M(ultiValue). If attribute 6 is unspecified, it defaults to S.

For PICK-style dictionaries where attribute 1 is an "A" or "S", attribute 5 is the MultiValue indicator with the same allowed defaults: S or M. In this case, however, if attribute 5 is not specified, it defaults to M.

Note: Traditionally, PICK dictionaries did not have a MultiValue indicator attribute. Caché MultiValue added one.

14.1.13 MultiValue File Pointers

For those porting MultiValue applications to Caché, please note the following:

- Only one F-Pointer in any Caché system is allowed per MultiValue file. The target of the reference can be either a Caché an operating system file.
- All other references to files named in F-Pointers must be entered as Q-Pointers.

Caché uses the F-Pointers to maintain indices. If there is more than one F-Pointer to the same MultiValue file, indices may not be updated correctly.

14.1.14 MultiValue Function Defaults For Queries

If attributes of the form, Fnnn (where "nnn" are digits) occur in an I-type and are not found first in the dictionary or the VOC, they are now handled by default.

Note: UniVerse does not process defaults like this; the VOC file contains entries for F1..F10.

14.1.15 Default Terminal Type

By default, the terminal type is assumed to be CACHE, but may be changed with the TERM command.

14.1.16 Inheritance of MVENABLED Classes

An MVENABLED class needs to be the only persistent class in its hierarchy. You cannot have subclasses of MVENABLED class and you cannot have a persistent superclass of an MVENABLED class.

- If you extend a class that extends %MV.Adaptor
- Or if you extend a class from both %MV.Adaptor and another class that extends %Persistent

You see this error message:

```
ERROR #5135: An MVENABLED persistent class does not support polymorphic dispatch
so you can not create a subclass 'MVFILE.L2' to the extent root class 'MVFILE.L1'.
```

Examples:

This fails:

```
Class MVFILE.L1 Extends (%Persistent,%MV.Adaptor)
Class MVFILE.L2 Extends MVFILE.L1
```

This fails:

```
Class MVFILE.L1 Extends %Persistent
Class MVFILE.L2 Extends (MVFILE.L1,%MV.Adaptor)
```

This compiles:

```
Class MVFILE.L1 Extends %Persistent
Class MVFILE.L2 Extends (MVFILE.L1,%MV.Adaptor)
```