Caché MultiValue Terminal Independence

Version 2018.1
2024-05-02

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

Caché MultiValue Terminal Independence
Caché Version 2018.1 2024-05-02

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

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.

In the domain of this discussion, “terminal” refers to console windows such as Terminal sessions, and the Microsoft Windows command terminal, and to legacy “green screen” terminals such as the VT-100 or the Wyse 60.

Caché contains a set of terminal definitions suitable for MultiValue applications. For practical purposes, this is limited to the most popular definitions, but it is easy to add or modify existing definitions. These MultiValue definitions differ from the terminal definitions of standard Caché devices because the latter do not cover all the requirements of a MultiValue application. The MultiValue definitions pointed to by the TERMDEFS definition in the VOC. The layout of an entry in the TERMDEFS file is a simple text record which mirrors the terminal definition given by the "infocmp" command on most UNIX® systems.

When a user begins a MultiValue session, Caché defaults to use a terminal of type CACHE of fixed screen size. This can be modified by executing the **TERM** command, as detailed later.

The application can write control characteristics to the terminal in an independent way using the MultiValue Basic @ function.

There is no real MultiValue standard for reading control characteristics from the keyboard. Pressing the F1 key on a keyboard, for example, will simply return a stream of regular characters in the range $c(32) through $c(255). Examples of input processing are also available in the SAMPLES database.

Note: These facilities allow replacement of the jBASE CommandInit and CommandNext subroutines.

1

Terminal Definition

1.1 The TERMDEFS File

Each account (namespace) in MultiValue will have an F pointer in the VOC called TERMDEFS which points to a database-wide file, ^%MV.TERMDEFS. Each entry in this global contains a simple text file which defines the characteristics of a terminal (both output and input characteristics). The global node name should be uppercase.

The layout of these text files is one or more lines, delimited by CR, CR/LF or @AM ($C(254)). The lines have the same attributes as that presented by the "terminfo" command available on most UNIX® systems. Indeed many of the supplied terminal definitions come from UNIX® declarations. For example, the global ^%MV.TERMDEFS("VT100") might look like this:

```
#
vt100|vt100-s|dec vt100 (w/advanced video),
am, msgr, xenl, xon,
cols#80, it#8, lines#24, vt#3,
acsc=``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
bel=^G, blink=\E[5m$<2>, bold=\E[1m$<2>,
clear=\E[H\E[J$<50>, cr=^M, csr=\E[%i%p1%d;%p2%dr,
cub=\E[%p1%dD, cub1=^H, cud=\E[%p1%dB, cud1=^J,
cuf=\E[%p1%dC, cuf1=\E[C$<2>,
cup=\E[%i%p1%d;%p2%dH$<5>, cuu=\E[%p1%dA,
cuu1=\E[A$<2>, ed=\E[J$<50>, el=\E[K$<3>, el1=\E[1K$<3>,
enacs=\E(B\E)0, home=\E[H, ht=^I, hts=\EH, ind=^J, ka1=\EOq,
ka3=\EOs, kb2=\EOr, kbs=^H, kc1=\EOp, kc3=\EOn, kcub1=\EOD,
kcud1=\EOB, kcuf1=\EOC, kcuu1=\EOA, kent=\EOM, kf0=\EOy,
kf1=\EOP, kf10=\EOx, kf2=\EOQ, kf3=\EOR, kf4=\EOS, kf5=\EOt,
kf6=\EOu, kf7=\EOv, kf8=\EOl, kf9=\EOw, rc=\E8,
mc4=\E[4i,mc5=\E[5i,
rev=\E[7m$<2>, ri=\EM$<5>, rmacs=^O, rmam=\E[?7l,
rmkx=\E[?1l\E>, rmso=\E[m$<2>, rmul=\E[m$<2>,
rs2=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h, sc=\E7,
sgr=\E[0%?%p1%p6%|%t;1%;%?%p2%t;4%;%?%p1%p3%|%t;7%;%?%p4%t;5%;m%?%p9%t\016%e\017%;$<2>,
sgr0=\E[m\017$<2>, smacs=^N, smam=\E[?7h, smkx=\E[?1h\E=,
smso=\E[7m$<2>, smul=\E[4m$<2>, tbc=\E[3g,
```

This example was taken from a UNIX® definition of a vt100 terminal. It shows all the characteristics of the terminal. Not all of these are needed by Caché, and those that are not relevant are ignored. This makes importing terminal definitions relatively painless, since no editing is required.

The pad characters, denoted by $<nn>, are for really really old terminals and hark from the days before flow control to serial devices. This information is ignored.

In the above example, \E denotes the escape character $c(27), non-displayable characters denoted by ^A through ^Z being $c(1) through $c(26). If a string is not a constant and requires parameters to determine the eventual string (for example moving the cursor to a particular row and column) these are denoted by %, as described in the **terminfo** manual pages on a UNIX® system.

The first line shown in the example is a comment. The second line is set of fields delimited by the | character, the last field simply being a description, the other fields being the name of the terminals this defines. This allows synonym terminals to be defined by the same source. Hence a single source in the TERMDEFS file may create one or more compiled entries in the TERMCMP file.

Note: If you upgrade your Caché installation, the TERMDEFS and the TERMCAP files are overwritten. Be sure to keep backups of these files.

1.2 Adding a New Terminal Definition

Create the source definition for the terminal. Caché uses a standard source as output by the "infocmp" command on UNIX® systems. Therefore, the usual way of adding a new terminal definition, assuming it already works on a UNIX® system, is as follows:

- Create a source file on a UNIX® system, for example with the command,

  infocmp wyse50 > /tmp/wyse50def
- Copy this source file to a folder/directory on your target system, for example, to C:\tmp\wyse50def.txt
- Copy this file into the TERMDEFS file, cut-and-paste with the ED command to add an entry into the TERMDEFS file
- Issue the COMPILE.TERM command.

Note: The item id of the item in the TERMDEFS file is uppercase and is the name of the terminal type. The name MUST be uppercase, although when you refer to the terminal type with the TERM command it is not case-sensitive.

1.3 Modifying an Existing Terminal Definition

A developer may use the ED command to edit an existing entry in the TERMDEFS file and then run the COMPILE.TERM command.

Note: You should save copies of any changes to TERMDEFS because they will be undone any time you upgrade your Caché version.

1.4 Deleting a Terminal Definition

COMPILE.TERM compiles the specified TERMDEFS and adds them to the existing TERMCMP file so they are available for use, but it does not clear prior TERMCMP entries. If you delete entries from TERMDEFS and don't want the terminal definitions to be available, particularly if you don't want to have them listed in CHOOSE.TERM, you must delete the specific terminal definitions that you don't want from TERMCMP *after* you have compiled TERMDEFS and know you have the working definitions available.

You can directly delete entries from TERMCMP. Individual TERMDEFS entries can generate multiple TERMCMP entries. For example, the second line of 'wy150' is: wy120|wyse120|wy150|wyse150|WY120|WYSE120|WY150|WYSE150|Wyse

120 and 150, This is a '|' delimited list of TERMCMP names to generate from this definition, except the last item, which is the Terminal Description shown by CHOOSE.TERM.

Some TERMDEFS contain an entry at the bottom. For example, 'wy60' line 36 is: .termtype=w. The 'w' is up-cased and becomes the Short Name option in CHOOSE.TERM.

1.5 The COMPILE.TERM Command

The data for terminal definitions exists in two globals. The %MV.TERMDEFS global contains the source for the definition. The %MV.TERMCMP global contains the compiled definition used by the runtime. To compile the sources from %MV.TERMDEFS to the compiled version in %MV.TERMCMP issue the COMPILE.TERM command. **COMPILE.TERM** no arguments compiles all the definitions in the %MV.TERMDEFS file. For example,

```
USER: COMPILE.TERM
123 terminal definitions compiled
USER:
```

1.6 The Default Terminal Type

By default, the type of a terminal is CACHE. Users can change current terminal type using the TERM command, for example,

```
USER: TERM vt100
USER: ; PRINT SYSTEM(7)
vt100
USER: TERM VT100
USER: ; PRINT SYSTEM(7)
VT100
```

1.7 The TERM Environment Variable

The **TERM** command also sets the TERM environment variable. When MV starts, if the terminal specified cannot be loaded, the MultiValue shell will

1. Display an error message to that effect;
2. Attempt to load the terminal type, CACHE;
3. If successful, display a message that the terminal type is set to CACHE. If unsuccessful (because, for example, it has been removed from the TERMDEFS file), the terminal type will be set to UNKNOWN and an error message indicating this will be displayed instead.

1.8 Supported TERMINFO Manipulation Codes

Cursor addressing and other strings requiring parameters are described by a parameterized string capability. For example, to address the cursor, the “cup” capability is supported. It expects using two parameters: the row and column to move the cursor to. (For more info, see the UNIX terminfo(5) man page.

This parameter mechanism uses a stack and special % codes to manipulate it. Typically, a sequence may push one or more parameters onto the stack and manipulate them in some way. The Caché termdefs file supports the following subset of the UNIX capability:

Code	Description
One or more digits	Push the integer onto the stack
%%	Add “%” to the output string
%p[1-9]	Push the designated parameter onto the stack
%c	Pop an integer from the stack and append it to the output string as a character
%d	Pop an integer from the stack and print its decimal value to the output string
%i	Add 1 to the first two parameters (p1 and p2). This is used for cursor positioning; it allows for some terminals that start their definitions for column and row with position 0, and other terminals that start with position 1.
\E	Append the ESCape character ($CHAR(27)) to the output string
%F	Caché extension: equivalent to “%p1%d”
%G	Caché extension: equivalent to “%p2%d”
%H	Caché extension: equivalent to “%p1%d;%p2%d”
%I	Caché extension: equivalent to “"%p1%{64}%+%c”
%J	Caché extension: equivalent to “"%p2%{10}%/%{6}%*%p2%+%c”
|	Append the newline character ($CHAR(10)) to the output string
\'x'	Push the character represented by the escape sequence onto the stack (the representation is given in octal)
%+, %-, %/ , %* , %m	Arithmetic operations on the top two positions of the stack: addition, subtraction, division, multiplication, and modulo, respectively.
%&, %|, %^	Bitwise operations: AND, OR, and negation, respectively
%=, %>, %<	Comparison operations: equal to, greater than, and less than, respectively
%A, %O	Logical operations: AND and OR
%? … %t … %e …	Conditional: If … then … else …

Let us assume we want to move the cursor to column 3, row 6 (column and row numbers start at 0, so column 3 is actually the 4th column and row 6 is the 7th row). The cursor function will be called with two parameters, 3 and 6, and the string

```
cup=\E[%i%p2%d;%p1%dH$<5>
```

is interpreted as follows:

Code	Interpretation
\E	Append the ESCape character ($CHAR(27)) to the output string
[	Append the character, “[”, to the output string
%i	Add 1 to the first two parameters; 3 and 6 become 4 and 7, respectively
%p2	Push parameter 2 (a 7) onto the stack
%d	Print the top of the stack (the integer 7) to the output string
;	Append the character, “;”, to the output string
%p1	Push parameter 1 (a 4) onto the stack
%d	Print the top of the stack (the integer 4) to the output string
H	Append the character, “H”, to the output string
$<5>	Add 5 units of pad character. Caché ignores this sequence. It was used for very old character-oriented terminals running over an asynchronous link without flow control.

So the resulting control sequence sent to the terminal to move the cursor to column 3, row 6 (zero-based) consists of the ESCape character followed by the characters “[7;4H”.

2

Terminal Output

2.1 TERM Command

The **TERM** command allows a user to query or establish certain characteristics of the terminal. When used from the shell as an interactive command:

```
TERM
```

the command displays the terminal characteristics. The terminal "type" can also be set from the command line, for example,

```
TERM VT220
```

sets the terminal to have the characteristics of a DEC VT220.

The characteristics of the terminal are usually set from a program, however, where the command format is

```
TERM swidth,sdepth,,,,,pwidth,pdepth,termtype
```

where:

- swidth and sdepth are the screen width (in characters) and depth (in lines)
- pwidth and pdepth are the printer width and depth (also in characters and lines, respectively)
- termtype is the terminal type identifier and can appear anywhere in the command as a non-numeric argument

Note: Arguments 3, 4, 6, and 10 of the command are no longer used; if present, they will be ignored.

The **TERM** command also supports single character terminal names which are synonyms for other terminal names. For example,

```
TERM W
```

selects the Wyse 60 terminal. To list the supported terminals and their single character synonym (if any), use the **CHOOSE.TERM** command.

2.2 The CHOOSE.TERM Command

The CHOOSE.TERM command allows you to see what terminals are supported on your system and then allows you to select a terminal type by entering a number, the name, or the short name. For 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
...
...
...
31)  xterm terminal emulator (X Wind   XTERM
Choose number, term name or short name :
```

The Short Name field is where the terminal definition has a “termtype=Xentry” and so allows the terminal to be selected via the **TERM** command using a single character synonym.

2.3 The SYSTEM() Function

The MVBasic SYSTEM () function is used to return a limited set of information about the terminal in use. The format is

```
SYSTEM(code)
```

where *code* is an integer that specifies the information desired. The following *code* values are used for terminal information:

- 1: return a non-zero value if output is not directed to a terminal
- 2: returns the screen width
- 3: returns the screen depth
- 7: returns the type name of this terminal

2.4 The @() Function

MultiValue Basic provides terminal output independence using the @ function call, as follows:

- @(m,n) where *m* is a positive integer

  Moves the cursor to column *m* and row *n*, where @(0,0) is the top left hand side of a terminal.
- @(m) where *m* is a positive integer

  Moves the cursor to column *m* on the current row.
- @(m) where *m* is negative integer

Generates video effect character strings. The value of *m* and the effect intended are emulation dependent and given by the table below.

- @(m,n) where *m* is a negative integer

  The same as @(m), except some video effects take an optional argument *n*, as shown by (nn) in the table below.

2.5 Predefined Names

The table below shows all the values we use from a terminfo definition. Caché does not use them all; it only use the ones necessary to support MultiValue Basic. In this table, there are some Caché defined names. These are shown by the name starting with an 'x' and the word "NEW" in the description field.

In the table below, the @(-nn) values vary between emulation. The values shown are for Cache Ideal (aka UniVerse) emulation, unless otherwise noted. The expression 'video attributes' refer to effects such as bold, dim, reverse and so on.

Name	Description	Usage
bel	Bell sounds	@(-108) or @(-19) in Unidata emulation
blink	Start blinking video attribute	@(-5)
bold	Start bold video attribute	@(-58)
civis	Make cursor invisible	@(-31)
clear	Clear screen and move cursor home	@(-1)
cnorm	Turn cursor back on	@(-32)
cub	Move the cursor left a number of columns	@(-9,nn). @(-9) when cub1 not available
cub1	Move the cursor left a single column	@(-9). @(-9,nn) when cub not available
cud	Move the cursor down a number of rows	@(-33,nn). @(-33) when cud1 not available
cud1	Move the cursor down a single row	@(-33). @(-33,nn) when cud not available
cuf	Move the cursor right a number of columns	@(-nn) and @(-34,nn). @(-34) when cuf1 not available
cuf1	Move the cursor right a single column	@(-34). @(n) and @(-34,nn) when cuf not available.
cup	Position the cursor at a specific column and row	@(col,row) cursor function call
cuu	Move the cursor up a number of rows	@(-10,nn). @(-10) when cuu1 not available
cuu1	Move the cursor up a single row	@(-10). @(-10,nn) when cuu not available
dch	Delete a number of characters	@(-22,nn). @(-22) when dch1 not available
dch1	Delete one character	@(-22) . @(-22,nn) when dch not available
dim	Start dimmed mode attribute	@(-11)
dl	Delete a number of lines	@(-18,nn). @(-18) when dl1 not available

Name	Description	Usage
dl1	Delete one line	@(-18) . @(-18,nn) when dl not available
ech	Erase a number of characters	@(-71,nn)
ed	Clear screen from cursor to end of screen	@(-3)
el	Clear screen from cursor to end of line	@(-4)
home	Move cursor to home position	@(-2)
ich	Insert a number of characters	@(-19,nn). @(-19) when ich1 not available
ich1	Insert one character	@(-19) . @(-19,nn) when ich not available
il	Insert a number of lines	@(-17,nn). @(-17) when il1 not available
il1	Insert one line	@(-17) . @(-17,nn) when il not available
ind	Scroll up a single lines	@(-48). @(-48,nn) when indn not available
indn	Scroll up a number of lines	@(-48,nn). @(-48) when ind not available
mc0	Dump contents of screen to printer	@(-28)
mc4	Reset auxiliary printing	@(-24). @(-26) if xmc4i not available
mc5	Start auxiliary printing to screen and printer	@(-23). @(-25) if xmc5i not available
prot	Start protected field attribute	@(-7)
rev	Start reverse video attribute	@(-13)
ri	Scroll up a single line	@(-49). @(-49,nn) when rin not available
rin	Scroll up a number of lines	@(-49,nn). @(-49) when ri not available
rmir	Reset insert mode	@(-21)
rmso	Exit standout attribute	@(-59)
rmul	Reset underscore attribute	@(-16)
setab	Set a background color	@(-38) and @(-38,nn)
setaf	Set a foreground color	@(-37) and @(-37,nn)
sgr	Reset specific video attributes Bit mapped video attribute calls. sgr0 = Reset all video attributes	@(-128) function call. When other reset attributes not found
smir	Start insert mode	@(-20)
smul	Start underscore attribute	@(-15)
xblink	NEW Reset blinking video attribute	@(-6)
xbold	NEW Reset bold video attribute	@(-59) if rmso is not available
xdim	NEW Reset dimmed mode attribute	@(-12)
xdisableprotect	Disable protected field handling	@(-63) or @(-12) in D3 mode
xdmi	NEW Disable manual input	@(-42)

Name	Description	Usage
xemi	NEW Enable manual input	@(-43)
xenableprotect	Enable protected field handling	@(-62) or @(-11) in D3 mode
xenter132	NEW Enter 132 column mode	@(-30)
xenter80	NEW Enter 80 column mode	@(-29)
xhpa	NEW Position the cursor at a specific column (This is used in preference to the hpa value which not all terminals work successfully with)	@(col) cursor function calls
xhpa1	NEW Set column position	@(-70). @(-70,nn) if hpa not available
xmc4i	NEW Reset auxiliary printing to printer only	@(-26)
xmc5i	NEW Start auxiliary printing to printer only	@(-25)
xprot	NEW Reset protected field attribute	@(-8)
xrev	NEW Reset reverse video attribute	@(-14)
xrnumeric	NEW Reset numeric keypad	@(-57)
xrtrunc	NEW Reset line truncate	@(-55)
xscl	NEW Scroll left a single column	@(-51). @(-51,nn) when xscl1 not available
xscl1	NEW Scroll left a number of columns	@(-51,nn). @(-51) when xscl not available
xscr	NEW Scroll right a single column	@(-50). @(-50,nn) when xscr1 not available
xscr1	NEW Scroll right a number of columns	@(-50,nn). @(-50) when xscr not available
xsetafdim	NEW Set foreground color dimmed	@(-109 through @(-116) or @(-57) to @(-64) in D3 emulation
xsmul	NEW Reset underscore attribute	@(-16) if rmul is not available
xsnumeric	NEW Set numeric keypad	@(-56)
xstrunc	NEW Set line truncate	@(-54)

2.6 jBASE- And Reality-specific Codes

Values -128 through -191 are for Reality video effects and are a bit map of video effects as shown in the list below. Caché supports these values for ALL emulations as there is no conflict. These values are additive. The only values Caché supports are:

- 0x01 – Half intensity
- 0x02 – Blink
- 0x04 – Reverse
- 0x08 – Blanked

- 0x10 – Underscore
- 0x20 – Bold
- 0x40 – (not defined)
- 0x80 – Always 1

Note: Value @(-128), which turns off ALL video effects, is available for all emulations.

2.7 MVBASE-Specific Codes

The color sequences @(-57) through @(-64) are used in the MVBASE emulation to produce dimmed foreground colors. The dimmed colors are white, yellow, magenta, red, cyan, green, blue, and black, respectively.

3

Terminal Input

3.1 The KEYS command

The KEYS command allows the user to see what characters are returned from the keyboard after depressing a particular key. The input will therefore be 100% binary and all characters will be interpreted by the **KEYS** command. Because of the manner of executing this command, there cannot be a "quit program" keystroke. Instead, the command will terminate after 10 seconds of inactivity.

This **KEYS** command is very useful when "debugging" a terminal definition. For example, to see what keystroke function key 3 generates, issue the KEYS command, depress function key 3, and then wait 10 seconds for the KEYS program to terminate. The following example shows that, on this terminal, pressing F3 generates an escape character followed by ASCII "O" and then ASCII "R".

```
USER:KEYS
This program terminates after 10 seconds of inactivity.
Key: ESC           0x1B  27
Key: O             0x4F  79
Key: R             0x52  82
Key: Timeout
USER:
```

3.2 Keyboard Independence

jBASE provides a pair of supplied subroutines called **CommandInit** and **CommandNext**. These subroutines are supplied in the dev/mv/samples directory supplied with Caché. Also included in the dev/mv/samples directory is the CommandInclude file, which is a replacement for the jCmdKeys.h file used in jBASE keyboard independence. This source defines values such as cmd_cursor_up which a jBASE application will probably be using. The values specified in Caché are different from jBASE, but the end result is the same. This mechanism is available for all emulations, not just jBASE.

Note: For jBASE releases 4.1 and higher, these subroutines were renamed JBASECommandInit and JBASECommand-Next, so depending upon which release of jBASE you are porting from, you might need to rename these subroutines in our dev/mv/samples directory.

To use these samples, you can copy them to your own source file and compile and catalog them. The following selection shows an example of creating a MultiValue file called COMMANDBP which points directly to the samples themselves. This is used to compile and catalog the source code directly from the dev/mv/samples directory, and then run a small example program to show the installation has worked. The example can also be copied to a local file and run from there.

```
USER:ED VOC COMMANDBP
COMMANDBP
New record.
----:I
0001= F
0002= C:\InterSystems\CacheSysBuild282\dev\mv\samples
0003= C:\InterSystems\CacheSysBuild282\dev\mv\samples
0004=
Bottom at line 3.
----:FI
"COMMANDBP" filed in file "VOC".
USER:BASIC COMMANDBP CommandInit CommandNext CommandExample
CommandInit
[B0] Compilation completed.
CommandNext
[B0] Compilation completed.
CommandExample
[B0] Compilation completed.
USER:CATALOG COMMANDBP CommandInit CommandNext CommandExample
[243] 'CommandInit' Cataloged Local.
[243] 'CommandNext' Cataloged Local.
[243] 'CommandExample' Cataloged Local.
USER:CommandExample
Function key 2 pressed
Function key 3 pressed
cmd returned was '2'
Timeout after 5 seconds
Alpha character returned, value = 'x'
USER:
```