

Standard M Pocket Guide

Edition 6.1 Update

Jon Diamond

Diamond Consulting Ltd

Electronic Edition

Copyright © **Diamond Consulting Ltd 2006**

Printed Edition

Copyright © **CAMTA** 2004

Caché and M Technology Association

<http://www.camta.net>

InterSystems Caché is a registered trademark of InterSystems Corporation.
All other marks, trademarks and product names are the property of their respective vendors.

Typographic Conventions

Convention	Meaning and Example
Arial type	<p>Syntactic definition. In general each element in a syntax definition can be an expression, except where specifically identified as not allowed. For example:</p> <pre>\$FIND(str1,str2,start)</pre> <p>can be coded as</p> <pre>\$FIND(a_" , "_b, c_d, e+f-1)</pre>
Arial type bold	<p>Approved abbreviation of command, variable or function name. For example:</p> <pre>CLOSE device:devparams</pre> <p>The command can be abbreviated to C.</p>
Optional syntax items	<p>These are not identified specifically in the syntax definition, but they are typically the rightmost items in a command/function. For example:</p> <pre>CLOSE device:devparams \$EXTRACT (expr, start, end)</pre> <p>means that the following are all valid:</p> <pre>CLOSE device \$EXTRACT (expr, start) \$E (expr)</pre>
....	Preceding syntax element is repeated an indefinite number of times.
<i>Times Roman Italic type</i>	Syntax elements used within a description section referring to the associated syntax definition.
Courier Bold	<p>Example code.</p> <p>Successive lines are understood to be executed sequentially.</p> <pre>D LINE1, LINE2^ROUTINE</pre>
⇒	<p>Shows a result in example code (spaces are significant)</p> <pre>x+y⇒ 123</pre> <p>A result of 123 preceded by a space.</p>
Shaded	<p>Extension to or difference from Standard M implementation-specific item.</p> <p>Note: The Standard defines all commands, functions and variables starting with Z as implementation-extensions.</p>
Shaded (darker)	Obsolete.

Foreword

This Guide aims to provide a quick, but comprehensive, reference for developers to the Standard M (MUMPS) programming language.

This sixth edition uses a slightly different format to the previous editions and also a different approach. It provides more specific information about the various implementations and their extensions to the Standard. However, in order to minimise the size, some infrequently used extensions and comprehensive function descriptions have had to be omitted.

Two areas specifically not covered by this Guide are that of program development and debugging; since these are not specified within the M Standard and also vary significantly between implementations and type of application.

This edition reflects the 1999 ISO M Standard (equivalent to the 1995 ANSI M Standard) and

- Fidelity Information Services, Inc - GT-M version 4.4
- InterSystems Corporation - Caché version 5.0
- M21 - M21 version 2.14

I would like to thank Paul Grabscheid from InterSystems, K.S Bhaskar from Fidelity Information Services and Keith Snell from M21 for their assistance in providing information and checking this document. Thanks also go to Ed de Moel and George James for reviewing drafts of this guide and providing significant input.

Special thanks go to the editors of previous editions: Rick Marshall (5th), Thomas Salander (4th), Joel Achtenberg (3rd and 2nd) and Joan Zimmerman who wrote the first edition

For a definitive reference to the Standard M language please refer to **ISO/IEC 11756:1999 – Information technology - Programming Languages - M** available from ISO (<http://www.iso.ch>) or your National Standards Body. For definitive information on a specific implementation please consult the vendor's reference manuals.

For more examples and an explanation as to how to use Standard M see **M[UMPS] By Example** by Ed de Moel (<http://www.jacquardsystems.com/Examples>).

All errors in this document are mine, but if you do find any please report them to CAMTA, so that they can be corrected in the next version of this Guide.

Jon Diamond
May 2004

This update (6.1 – January 2006) reflects errors in the 6th Edition and the releases of GT.M version 5.0 and Caché version 5.1. I

Commands

Command	Description
ZBREAK location:action:condition:code ZBREAK /command:action	Caché: Sets a breakpoint at <i>location</i> or a watchpoint if this is a variable. If <i>condition</i> is TRUE at execution then <i>code</i> is executed if provided otherwise <i>action</i> is taken. The second form governs all break/watchpoints.
ZNSPACE expr	Caché: Current namespace is <i>expr</i> . ZN "USER"
ZQUIT lev	Caché: Exits <i>lev</i> trap levels. If <i>lev</i> not specified exits all trap levels. M21: Exits to next highest execution level with a trap set. (No <i>lev</i> allowed) ZQ
ZSYNC	Caché: Completes all network transactions for current job.
ZTRAP error	Caché+M21: Forces error condition with error code = " <i>Z</i> "_error Caché: special syntax ZTRAP \$ZERROR pops stack until error trap found. ZT "user request"
ZWRITE ZWRITE variable	Caché: Writes all local variables preceded by the variable name to current device. The second form writes <i>variable</i> and all its descendants.
ZZDUMP expr	Caché: Writes <i>expr</i> in hex format.

Intrinsic Functions

Function	Description
\$CHAR (val,...)	ASCII (or Unicode) character for <i>val</i> . \$C (65) ⇒A \$C (66, 65, 67, 75) ⇒BACK
\$DATA (variable,target)	Status information about <i>variable</i> : 0 not defined and no descendants 1 defined and no descendants 10 not defined and descendants 11 defined and descendants Caché: <i>target</i> becomes the value if <i>variable</i> is defined K ^global \$D(^global(5)) ⇒0 S ^global(5)="" \$D(^global(5)) ⇒1 S ^global(5,1)=1 \$D(^global(5)) ⇒11 \$D(^global(5), var) ⇒11 var ⇒5
\$FACTOR (number,scale)	Caché: Bitstring value of <i>number</i> multiplied by 10 ^{<i>scale</i>} rounded to an integer. \$BIT(\$FACTOR(2.2,0),1) ⇒0 \$BIT(\$FACTOR(2,0),2) ⇒1
\$ISVALIDNUM (num,scale,min,max)	Caché: Checks if <i>num</i> valid as a number, within the range <i>min</i> to <i>max</i> and rounding to <i>scale</i> number of decimal digits. If <i>scale</i> =-1 then <i>num</i> is truncated to an integer value. \$ISVALIDNUM(4.6,0,4,5) ⇒0 \$ISVALIDNUM(4.6,-1,4,5) ⇒1
\$LISTFROMSTRING (string,delim) \$LFS (string,delim)	Caché: List created from <i>string</i> using <i>delim</i> as the element separator. If <i>delim</i> is not specified a comma is used as the delimiter. S a=\$LISTBUILD(1,2,,4) S b=\$LFS("1,2,,4") \$LS(a,b) ⇒1

Function	Description
<code>\$LISTNEXT(list,ptr,value)</code>	<p>Caché: True if there is a next item in the list and returns in <i>ptr</i> a pointer to it and in <i>value</i> its value. <i>ptr</i> must be 0 before the first invocation for a list.</p> <pre>S p=0 \$LISTNEXT(a,p,v)⇒1 v⇒1 \$LISTNEXT(a,p,v)⇒1 v⇒2 \$LISTNEXT(a,p,v)⇒1 v⇒ \$LISTNEXT(a,p,v)⇒1 v⇒4 \$LISTNEXT(a,p,v)⇒0</pre>
<code>\$LISTTOSTRING(list,delim)</code>	Caché: <i>list</i> converted to a string using <i>delim</i> as delimiter between elements.
<code>\$LTS(list,delim)</code>	<code>\$LTS(a,"")⇒1*2**4</code>
<code>\$LISTSAME(list1,list2)</code>	Caché: True if the lists are the same.
<code>\$LS(list1,list2)</code>	<pre>S c="1,2,,4" \$LS(a,b)⇒1 \$LS(a,c)⇒0</pre>
<code>\$NORMALIZE(num,scale)</code>	<p>Caché: The numeric part of <i>num</i> rounded to <i>scale</i> decimal places. If <i>scale</i>=-1 then <i>num</i> is truncated to an integer value.</p> <pre>\$NORMALIZE(4.567,2)⇒4.57 \$NORMALIZE("4a",0)⇒4</pre>
<code>\$ORDER(variable,dir,target)</code>	<p>Next/previous subscript of <i>variable</i>. If <i>dir</i> is -1 the search is backwards. Caché: <i>target</i> becomes the value of the result node if defined</p> <pre>S ^a(1)="11",^a(1,"a")=2 S ^a(1,"aa")=3 S ^a(1,"b")=4 \$O(^a(1,"aa"))⇒b \$O(^a(1,"aa"),-1)⇒a \$O(^a(1,"c"))⇒ ;null \$O(^a(""))⇒1</pre> <pre>\$O(^a(""),,var)⇒1 var⇒11</pre>
<code>\$QUERY(variable,dir,target)</code>	<p>Full reference of next node following <i>variable</i> containing data. Caché: If <i>dir</i> is -1 the search is backwards. <i>target</i> becomes the value of that node.</p> <pre>K a S a(1)=one,a(1,"last")=2 S b=\$Q(a(1)) b⇒a(1,"last") @b⇒2 \$Q(@b)⇒1</pre> <pre>\$Q(@b,-1,var)⇒a(1) var⇒one</pre>
<code>\$ZF(function,param,...)</code>	Caché: Calls out of Caché.
<code>\$ZF(type,command,...)</code>	<p>The first form executes the named (pre-linked) function with appropriate parameters. The second form calls any O/S <i>command</i>. <i>type</i> is:</p> <ul style="list-style-type: none"> -1 Execute expr as O/S command and returns result code -2 Executes expr as O/S command asynchronously -3 Load and execute DLL -4 Load DLL -5 Executes DLL -6 Executes DLL using index <pre>\$ZF(-1,"dir > tmp.txt")</pre>

Function**Description****\$ZUTIL**(function,param)

Caché: Implementation and system functions. Some are:

- 4 terminates another process
- 5 returns current namespace, switches to *param* if this supplied
- 9 broadcast message
- 12 translates device name/file to canonic form
- 18 treatment of undefined variables
- 20 namespace for routines
- 22 # or <BS> control sequence
- 39 search path for % routines
- 49 database label
- 55 M language mode
- 58 privilege level for XECUTE
- 62 syntax check of code
- 67 process information
- 68 set/unset process-specific defaults
- 69 set/unset system-wide defaults
- 71 set date
- 78 journal information/action
- 82 *param=12* redirects I/O
- 86 configuration location
- 90 namespace functions
- 94 broadcast message
- 96 sets/retrieves process information
- 100 Shows running Windows O/S
- 110 system name
- 113 reclaims routine/directory blocks
- 114 Ethernet address
- 128 *param=1* last debugging address
- 130 domain id
- 131 system identifiers
- 132 sets \$PRINCIPAL
- 133 metrics
- 140 file attributes + file/directory maintenance
- 147 handles pathnames with spaces
- 158 installed printer information
- 168 working directory
- 186 sets programmer prompt options
- 188 current date/time with fractional seconds
- 189 checks if TCP device disconnected
- 193 converts time/date to/from UCT

Special Variables

Variable**Description****\$ROLES**

Caché: Comma separated string of roles for the current process

\$TLEVEL

Caché: current nesting level in transaction processing

\$USERNAME

Caché: Full user name for the current process

\$USERNAME⇒mary@jupiter.com**\$USERNAME⇒UnknownUser**

Caché Objects

In addition the following intrinsic functions are provided to allow access to classes whose name cannot be determined at compile time and therefore not accessible through the ##class syntax:

\$ZOBJCLASSMETHOD(class,method,arg...)

Caché: Object class method access

\$ZOBJMETHOD(inst,method,arg...)

Caché: Object instance method access

\$ZOBJPROPERTY(inst,property,index...)

Caché: Object instance property access

Implementation Versions

This section summarises the most significant changes as they affect this Guide (items are additions unless otherwise stated). Each release includes implementation and environment changes in addition to those mentioned here:

Caché

Version 5.1

\$FACTOR, \$LISTNEXT, \$LISTTOSTRING and \$LISTFROMSTRING functions
\$ROLES and \$USERNAME special variables
Additional \$ZUTIL options
ZTRAP \$ZERROR command
ZQUIT described as obsolete

Version 5.0

\$ISOBJECT and \$BIT functions
\$ZBIT functions described as obsolete
CONTINUE command
Procedure blocks

GT.M

Version 5.0

\$INCREMENT
Names can now be 31 characters

Version 4.4

No changes

M21

Version 2.14

No changes

Version 2.12

SET \$KEY

Version 2.08

]] operator