MSM to Caché Conversion Guide

Version 2018.1
2020-11-13

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

MSM to Caché Conversion Guide
Caché Version 2018.1 2020-11-13

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

Whenever you migrate from one database environment to another, there are always many points to consider. The decisions you will have to make can be as simple as determining how much disk space you will need, or as complex as figuring out which implementation-specific language features you will need to change in your code in order for your application to work.

This document will attempt to address all of these concerns. We will cover a variety of topics ranging from hardware requirements to data migration to system management.

Because this conversion guide is meant to provide a path for you to follow, rather than actually teach you how to use the various Caché components themselves, you will want to familiarize yourself with the Caché documentation as much as possible. These references will be your most complete source for moving ahead with Caché. See the InterSystems web sites for the most up-to-date information on Caché (http://www.intersystems.com/cache/index.html) and MSM (http://mtechnology.intersys.com/mproducts/index.html).

You are of course invited to contact your InterSystems Account Manager and Sales Engineer for more detailed information as necessary.

Overview

Migrating to Caché from MSM involves several steps. Your success in this project depends greatly on carefully planning each aspect of the transition. Keep the following points in mind:

1. *Choosing the appropriate hardware and installing Caché*
    - Define a tier structure suitable for the application and user load
    - Choose the best disk configuration for optimal performance
2. *Distributing your databases on individual or networked servers*
    - Design data distribution as a means of load balancing
3. *Converting the MSM volume groups*
    - Convert globals by UCI and/or volume groups, or
    - Convert globals individually
4. *Converting the application*
    - Move routines over to Caché
    - Handle implementation-specific M language features and syntax
    - Connect the application to Caché
    - Call out to the operating system from Caché, and vice-versa
5. *Creating solid Caché system operations*
    - Choose the right backup method
    - Implementing system security

- Manage users

6. *Training your staff*
   - Learn systems operations for both the operating system and Caché
   - Enroll in InterSystems training programs
   - Teach staff internally

CAUTION: All information written in this document is believed to be accurate as of the date of publication. While most of the strategies behind migrating to Caché 5 will be applicable to migrating to future versions of Caché, there may be some differences. You should carefully consult future Caché documentation to learn how the strategies presented in this paper may need to be modified.

Any code presented is by example only, and is unsupported and without warranty. You are free to use any code you see here, but in doing so you assume all responsibility and risk associated with its use. We welcome any comments or suggestions that you might have regarding this guide.

1

Getting Started

1.1 Determining Hardware Requirements

Before beginning this project, you will need to make sure your hardware meets the requirements for the expected user load. Most of your hardware-related decision making will be based on total process count. First, you should check your operating system documentation for suggested hardware configurations based on this total process count. In particular, pay attention to requirements for CPU, memory, and swap space. In addition to these operating system requirements, Caché specifically requires shared memory to store various system structures such as global, routine, and network buffers.

See your platform's version of the *Caché Installation Guide* for an understanding of how Caché uses memory, and how to calculate the appropriate memory configuration.

In some cases, you may be changing hardware platforms as part of the migration process. Your Caché migration is a good opportunity to move to a more powerful hardware platform, as Caché makes optimal use of SMP and multiple-disk configurations. In these cases, you will need to carefully consider how your application will exist in the new hardware configuration.

For the most recent list of currently supported Caché platforms, see the Current and Planned Caché Products page on the InterSystems web site:

http://www.intersystems.com/cache/technology/product-tables/current-prodlist.html.

1.2 Determining Networking Requirements

Caché 5 networking supports the following message format protocols:

- Enterprise Cache Protocol (ECP)
- Distributed Cache Protocol (DCP)
- Distributed Data Processing (DDP)
- DTM-NetBIOS (For a DTM client/Caché server configuration only)

on top of one of these communication protocols:

- Raw Ethernet
- TCP/IP (both UDP and TCP)
- NetBIOS

- NetBEUI

You use the Management Portal to set up your network connections; see the *Caché System Administration Guide*. For more information on Caché's supported network protocols, see the *Distributed Data Management Guide*.

1.2.1 Mixed MSM/Caché Networks

Peer-to-peer networking between MSM and Caché can be achieved using DDP over Raw Ethernet. With this method, MSM can act as either a client or a server to Caché. At some stages of the conversion process, you may find it useful to have some machines running MSM and some running Caché, with Caché-level networking connecting your systems.

When planning a mixed network, bear in mind the following limitations:

- MSM does not support DCP or ECP networking.
- Caché does not support the Open MUMPS Interconnect (OMI) protocol.
- You cannot share routines across a network between Caché and MSM
- Caché does not support RVGs (Remote Volume Groups). For Caché to Caché connectivity it is recommended that you use ECP, which can be likened to a high-performance equivalent to MSM's RVGs.
- You cannot share string collated globals over a DDP network between MSM and Caché (see the section on "Preserving MSM String Collation" for details).
- Accessing 8KB databases over DDP may cause problems because DDP does not support accessing globals that contain nodes with subscripts containing more than 255 characters.

1.2.2 Running Caché and MSM on the Same Machine

It is possible to have Caché and MSM installed and running on the same machine. If you plan on using DDP networking between Caché and MSM on the same machine, you will need two separate Ethernet cards (see "Configuring Multiple Caché Instances" in the *Caché System Administration Guide*).

If you want both Caché and MSM to run connectivity services simultaneously on Windows NT, you will need to do the following:

- For Telnet, change the Telnet port number for one of the systems. By default, both Caché and MSM use port 23.
- For LAT services, make sure that both Caché and MSM use unique service names.
- For Serial Port services, make sure MSM and Caché use unique ranges of COM ports.

For more information on Telnet, LAT, and Serial Port services, see "Terminal Servers".

1.2.3 Ensuring Unique $JOB Values Across the Network

While MSM networking could be set up to ensure unique $JOB values across a network, Caché cannot. Since Caché obtains a process ID (PID) from the operating system for each Caché process, it's possible for client processes from different machines to carry the same $JOB value. If your application sets globals indexed by $JOB from multiple clients to the same server, you can correct the situation with any of the following options:

- Combine $ZU(110), which returns the Caché node a process is running on, with $JOB to guarantee a unique identifier. This is functionally similar to MSM's $SYS.
- Store any globals indexed by $JOB locally on each client (this is generally a good practice on any system).
- Use subscript-level global mapping to distribute the global differently.

1.3 Installing Caché

The steps taken to install Caché depend upon the host operating system. The following outline will highlight some important information, but you should read your platform's version of the *Caché Installation Guide* for more details.

Before installing Caché make sure you that you have the following:

- System privileges for your machine (either Administrator, root, or system, depending on your operating system)
- Approximately 200MB of free hard disk space for the core product installation
- TCP/IP configured for your OS (Caché requires that you have the TCP/IP protocol installed on your machine, even for stand-alone configurations, because it is used to connect the Caché GUI utilities to the server.)
- The license key supplied to you by InterSystems to activate Caché's product features (recommended)

1.3.1 Installing the Caché License Key

If you have not configured your Caché license during initial installation of the product, or you can do so with the Management Portal, or via a host-based editor such as vi, emacs, or edit. The license information is stored in an ASCII file called cache.key, found in your Caché manager's directory. Caché license keys are case-sensitive. If the license is not properly entered, Caché will start in a single-user mode. All fields must be entered.

On Windows platforms, the installation program will ask you for your license key during installation if it has not already located one. In UNIX®, if the installation program does not prompt you for the license, you will need to enter the license manually, using either a host-based editor or the License wizard on a PC client running Caché. In either case, the information must be entered exactly as shown on your paper copy of the license.

If you purchased licenses for multiserver systems, you must also configure one or more license managers to allocate the Caché license units authorized by the key.

Unlike MSM, where licenses are purchased in blocks of concurrent users, Caché licenses are typically purchased on a simple per concurrent user basis. For example, you could purchase a 17 user Caché license, which is not possible under MSM. There are a number of different license types available with Caché, and it is recommended that you contact your InterSystems Account Manager for more detailed information on this topic.

For more information on how to install and configure your license, see your platform's version of the *Caché Installation Guide*.

2

Creating Caché Databases and Namespaces

To create a Caché database and access its data, use the Management Portal, which provides a convenient wizard that creates the physical volume and namespace, if needed.

2.1 Creating Database Files

A Caché database can be one of two formats — 2KB block size or 8KB block size. InterSystems strongly recommends (for performance and ECP networking reasons) that you convert your MSM databases to the 8KB database format.

A Caché database is consists of a file named CACHE.DAT, representing a logical database which can grow to 32 terabytes, assuming that the host operating system can support files large enough for this. The database is referenced by the directory path of the CACHE.DAT file.

The absolute limit on the number of databases per Caché instance is 15,998, but the practical limit is likely to be much lower. See Configuring Databases in the "Configuring Caché" chapter of the *Caché System Administration Guide* for more information about the maximum number of databases per instance.

Some key points to consider while creating databases are:

- Creating as large a database as possible will aid in combating fragmentation issues at the operating system level.
- You can reference a database that physically exists on another system. For this to happen, you must first have a working Caché network configured between the two systems.
- Currently, database names, which are mapped to physical locations, are internally translated to all uppercase and do not allow punctuation.

See the *Caché System Administration Guide* for detailed instructions on creating, editing, and deleting databases.

2.2 Creating a Namespace Configuration

You do not typically work within a database in Caché, but rather in a *namespace*, which is a virtual environment.

It is important to understand the differences between the concept of a Caché namespace and an MSM UCI. Simply put, a namespace is roughly equivalent to an MSM UCI plus its relevant translation table. This translation table also implicitly

includes locations of both % globals and % routines. By default, all % globals and % routines exist in either the CACHESYS, CACHELIB or CACHETEMP databases, but they can in fact live in any database that you choose, as long as the namespace mapping is first set correctly.

CAUTION: During a Caché upgrade, the CACHESYS, CACHELIB and CACHETEMP databases can be purged of globals and/or routines, or even completely replaced by a new version.

Thus, it can be considered dangerous to simply load any user written % routines into the Caché manager. Unless you completely understand the ramifications of using the Caché manager databases, it is much safer to define the namespace mappings to store your % routines and globals elsewhere, or modify your application to use non-% names for application specific globals and routines.

In Caché, unlike MSM, the concept of % globals and routines is a naming convention only, and has no hard "mapping" implications. See the chapter on "Converting MSM Globals" for tips on how to handle extended global references from your MSM application.

Once your databases are defined, you will add them to your namespace configuration. Through namespace mapping you will be able to access data from all of your databases from one logical location. To create Caché namespaces, you use the Management Portal.

Some useful facts on namespaces:

- Namespaces allow you to break the old 16-gigabyte physical limitation of a single database for your system.
- Mapping globals to different databases is an effective way to balance the load on your machines. You should always consider different database layouts for optimal I/O performance.
- The ^%CD utility allows you to list and change namespaces (enter a question mark at the prompt to list all available namespaces). See the "Changing Namespaces" section in the "Introduction to the Terminal" chapter of *Using Terminal* for more detailed information on navigating Caché namespaces.
- You can see the active namespace mappings by running either ^%NSP or ^%SYS.GXLINFO.
- As with databases, namespace names currently are translated internally to all uppercase.

2.2.1 The CACHETEMP Database

This database has special characteristics that are designed to give higher performance and make it easier to clean up temporary globals.

The CACHETEMP database is purged automatically by the system on restart. Because the database is primarily used for temporary working files, the system will attempt to retain in memory any database blocks that are scheduled to be written to it for as long as possible. Data blocks are only flushed to disk in cases of clean buffer starvation in the buffer pool. This feature can relieve the workload on the system and lead to superior performance and scalability.

To make use of CACHETEMP and the benefits that it brings, simply configure global mapping for the relevant globals (using wildcards where possible) to point to the CACHETEMP database.

3

Converting MSM Globals

Caché provides several mechanisms for converting your MSM data.

3.1 Converting Selected Globals

Caché offers the following methods for converting individual globals or groups of globals, rather than an entire database:

- Set up a network between MSM and Caché. Use either the MERGE command or ^%GCOPY to transfer globals from the MSM database to the Caché database.
- Use %GS or %FGS to save individual or groups of MSM globals in MSM format.
  - For %GS-format global saves, use ^%GI to import the data. It is more reliable to use the ANSI ^%GS entry point, especially if the globals can contain control characters.
  - For %FGS-format global saves, use ^%SYS.GIFMSM to import the data.

3.2 Preserving MSM String Collation

By default, Caché uses the Latin-1 subset of Unicode collation. The Latin-1 subset follows the same collation rules as MSM's default numeric collating sequence. You do not need to do any preliminary work to convert numerically collated MSM globals since Caché's Unicode collation will be backward compatible with this format. However, since you cannot reference string collated globals across DDP from Caché, you will need to prepare Caché in order to convert MSM globals that use a string collation.

For example, suppose that you need to convert a string collated global called ^XYZ. There are two ways you can do this conversion:

1. Create the ^XYZ global using the ^%SYS.GCREATE utility. This option is recommended unless you want to convert all MSM globals with a string collation type.
   - Change to the appropriate namespace.
   - Invoke ^%SYS.GCREATE, and select type129 (string collation) when prompted for the globals collation sequence.
   - Convert as described previously in "Converting Selected Globals".
2. Change the characteristics of the process performing the conversion to use string collation.

- Issue the command:

  ```
  Set x=$ZU(23,1,129)
  ```

- Convert as described previously in "Converting Selected Globals".

3.3 Enabling Null Subscript Support in Caché

By default, Caché does not allow the use of null subscripts in your global data. It is strongly recommended that you modify your application and data so that null subscripts are not used.

There are two ways you can convert MSM globals that contain null subscripts.

- Enable null subscript support interactively:
  - For the issuing process *only*, run the command:

    ```
    Set x=$ZU(68,1,1)
    ```

  - For *all* processes in the system, run the command:

    ```
    Set x=$ZU(69,1,1)
    ```

  - Convert as described previously in "Converting Selected Globals".

- Enable null subscript support at each Caché startup for *all* processes. To do so, add the following line of code to either ^ZSTU or SYSTEM ^%ZSTART:

  ```
  Set x=$ZU(69,1,1)
  ```

  Then restart Caché and convert as described previously in "Converting Selected Globals".

3.4 Converting Globals by Database

If you plan to convert all globals in a particular volume group or UCI at once, it is recommended that you convert it in its entirety, rather than convert individual globals or groups of globals. Converting an entire UCI or volume group at once is not only a much faster process, but it also drastically reduces the risk of error.

1. Make sure that any incompatible MSM globals have been already converted:
   - MSM globals that use a string collation must be converted as described previously in "Preserving MSM String Collation".
   - MSM globals that contain null subscripts must be converted as described previously in "Enabling NULL Subscript Support in Caché".
2. Back up your MSM databases.
3. Run ^VALIDATE on your MSM database to ensure physical integrity.
4. If converting in block mode, run ^RECOVER to ensure that there are no unused blocks that are still marked as allocated in their map blocks. (If globals have been removed from the global directory block but ^RECOVER has not been run, %MSMCVT will see this data as valid global data and convert it to Cache data).

5. Run ^OLC on your MSM database to maximize global efficiency. This will help the conversion's overall performance.
6. Kill any MSM system or user-defined scratch globals.
7. Ensure that the database to be converted is not active, either by shutting down MSM or by dismounting the database.
8. Use either binary-mode FTP, network copy, or tape to move your MSM database over to the machine hosting Caché. You can put this database in any directory you wish.
9. Configure as many global buffers as possible on your Caché system, and then start Caché.
10. Make sure your local destination database(s) and namespace are ready (as described previously in "Creating a Namespace Configuration").
11. From the new namespace, run the ^%MSMCVT utility. At the prompt, enter the path and filename of your MSM database. When you run the conversion, your database is converted into a Caché database in the current namespace.
12. If you are importing data from several MSM databases into one Caché namespace, run the utility repeatedly from the same namespace, once for each database.

3.4.1 Running %MSMCVT in Block Mode

^%MSMCVT can run in two modes, block mode and non-block mode. There is a prompt during the setup that says "Convert with block mode?" If you answer "No" to this prompt ^%MSMCVT will use non-block mode. The difference is important:

- In block mode, ^%MSMCVT scans through the entire database sequentially, block by block, looking for data blocks that are marked as allocated in the corresponding map block. When it finds one, it sets it into the new Caché database.
- In non-block mode, ^%MSMCVT converts the database global by global (which allows you to avoid converting some globals if you want) and follows the tree for each global.

Running ^RECOVER on the MSM side is an absolute requirement if you use block mode conversion, unless you are 100% sure that you have no "orphan" blocks in the MSM database from previous repair work. An "orphan" block is a block that is marked as allocated in its map block but is not a part of any valid global tree. This can happen if a block was cut out of a tree during database repair, but the block was never unallocated in the map. If there are orphan blocks and ^RECOVER is not run, old data that was not visible in MSM will be present in Caché, and will be indistinguishable from valid data.

If you run ^%MSMCVT in non-block mode ^RECOVER is not required (and in fact will make no difference).

4

Converting Your MSM Application

It is strongly recommended that you convert any MSM-specific language features to native Caché code, even though the Caché MSM language compatibility mode may allow you to run your MSM application with fewer changes. Conversion will be the first step in taking advantage of Caché-specific features such as ObjectScript.

This chapter discusses the following topics:

- Differences between Caché routines and in MSM routines
- How to port your MSM routines to Caché
- How to switch on and use MSM language mode
- Converting Nonstandard Functions and Features
- MSM and Cache Database Networking (DDP)

4.1 How Routines Differ

Working with routines is slightly different in Caché than in MSM.

The MSM approach:

- A programmer works with routine source code, ZSaves it (implicitly or explicitly) and it is compiled implicitly into object code, regardless of how it is modified.
- Both source and object code are stored on disk together in routine blocks.
- At runtime the object code is loaded into generic buffers in the buffer pool.

The Caché approach:

- A programmer works with either INT (intermediate) or MAC (macro) code, or possibly INC (include) code. These are actually extensions to the routine name. I.e. A123.mac.

  The Caché equivalent to MSM source is INT code. Macro code is useful when embedding macro statements such as embedded SQL directly into your code.
- A programmer edits a routine using the Studio, but routines require explicit compilation to generate the OBJ object code files (the equivalent to MSM p-code).

  Compiling MAC code in Caché automatically generates both INT and OBJ code. Compiling INT code automatically generates the OBJ code as well.

- All code is stored in globals (^rMAC, ^rINV, ^rOBJ and ^ROUTINE).
- At runtime, only the OBJ code is loaded into special routine buffers that are separate from the main global buffer pool, with the obvious advantages that this brings.

4.2 Porting MSM Routines

Use the following procedure to migrate MSM routines that are less than 64KB in size:

- Save your MSM routines in MSM format using the ^%RS utility.
- Restore your code into Caché using ^%SYS.RI in the destination namespace.
- From the destination namespace, use ^%RCOMPIL recompile your code in Caché and check for syntax errors. Once you find which routines have generated <SYNTAX> errors, you should look to the *Caché ObjectScript Reference* to familiarize yourself with Caché's language syntax.

Although MSM can support as large a routine as the current stack and partitions sizes will allow (tested as high as 200 KB), Caché routines must compile to less than 64KB in size. You can use MSM's ^%RSIZE utility to check your routine's size before importing into Caché. To figure out how large a routine is, multiply the number of Text Blocks, as reported by ^%RSIZE, by 1 KB.

Any MSM routines larger than 64KB can be addressed in one of two ways:

- Split the routines in MSM first so that no routine is larger than 64KB, then import into Caché.
- Import the routines into Caché first, with Syntax Checking and Compile options off. Once in Caché, split the routines and then use ^%RCOMPIL to compile them.

You cannot migrate an MSM routine that is only stored as p-code (known as object code in Caché). If you need to migrate a routine that was only supplied as p-code, you will need to contact your application supplier to get the routine's source code before proceeding.

4.3 MSM Language Compatibility Mode

Caché operates in MSM language mode when working with a MSM routine that has been ported. A language mode can be set individually for each routine, and a routine compiled in one language mode can call or be called by a routine compiled in another mode. Thus, for instance, an MSM mode routine could call a Caché mode routine which could in turn call another MSM mode routine.

Use **$ZU(55, mode)** to switch language modes, where *mode* is the number of the language mode. The relevant settings are:

- Switch to MSM mode:

```
Set x=$ZU(55,8)
```

- Switch to native Caché mode:

```
Set x=$ZU(55,0)
```

- Return the current language mode:

```
Set x=$ZU(55)
```

After an MSM application is compiled in the correct language mode, it can be installed and run on any Caché system, no matter what other applications or language modes are used on that system. Almost all language mode processing occurs at compile time, not runtime. As a result, using a language mode such as MSM will generally deliver the same high performance as the Caché native language mode.

Native Caché mode is the default for each process. To change this default for specific process types, you will need to modify the appropriate line tag within the ^%ZSTART routine.

From within any of these modes, you can add your own commands, functions, or special variables, allowing you to convert your MSM %ZZCMNDS, %ZZFUNCS, and %ZZSVARS MSM routines. Depending on the current language mode, corresponding Caché routines will be searched for these added features.

	Commands	Functions	Variables
Original (native MSM) routines	%ZZCMNDS	%ZZFUNCS	%ZZSVARS
Native Caché mode routines	^%ZLANGC00	^%ZLANGF00	^%ZLANGV00
MSM mode routines	^%ZLANGC008	^%ZLANGF008	^%ZLANGV008

The line tags you enter in these routines will be the names of your new commands, functions, or special variables. The line tags entered must begin with a “Z” and must be all capital letters. Actual execution of the command, function, or special variable is not case-sensitive.

- Adding code to ^%ZLANGC* creates a new command, such as ZSS:

```
%ZLANGC008
ZSS   ; do a System Status
   Do ^%SS
   Quit
```

- Adding code to ^%ZLANGF* creates a new function, such as $ZRTN(x):

```
%ZLANGF008
ZRTN(x)   ; find out what routine a process is currently running
   Quit $Print($View(-1,x),"^",6)
```

- Adding code to ^%ZLANGV* creates a new special variable, such as $ZTIME:

```
%ZLANGV008
ZTIME   ; return the current time
   Do INT^%T
   Quit %TIM
```

Here is an example of custom code that calls the routines defined above:

```
MyRtn   ; Call the code added to the ^%ZLANG* routines
   ;
   New pid,rtn,x
   Set x=$ZU(55,8)   ; Change to MSM mode
   ZSS
   Read !,"Enter a process ID: ",pid
   Set rtn=$ZRTN(pid)
   If rtn=""  Set rtn="no routine"
   Write !,"At "_$ZTIME_", process #"_pid_" was running "_rtn_"."
   Set x=$ZU(55,0)   ; Return to native mode
   Quit
```

4.4 Converting Nonstandard Functions and Features

Converting MSM's implementation-specific functions and features will be the most intricate part of your conversion. Issues to keep in mind include:

- Many MSM commands such as ZUSE, functions such as $ZOS, and special variables such as $SYSTEM will need to be handled differently for Caché.
- Any VIEW commands or $VIEW functions which access MSM memory structures will need to be removed or modified.
- Any references to MSM utilities may need to be converted.
- Caché does not support the hex operator #. However, support for $ZHEX is identical and so this function should be used in its place.
- Caché does not support printer mnemonics.
- In some cases, the ^%RCHANGE utility can be used to make some simple syntax changes, but you should use this method with care, as blind modifications to a routine can make unwanted changes to your code. Most code changes should be done interactively by a programmer.

Here are a few suggested actions that you can take to get a rough idea of how much effort will be involved in an MSM to Caché conversion.

- Load all your MSM routines into Caché and run %RCOMPIL and analyze the syntax errors
  - Most of the open and use commands should appear here.
  - Extraneous whitespace is a common problem. You might run into cases where MSM successfully compiles a routine, but Caché generates a <SYNTAX> error upon compiling due to extra whitespace. Whitespace issues are best corrected interactively by a programmer. A good example of this is less than two spaces between a QUIT command and a semicolon.
  - Duplicate line tag names in routines are not supported in Caché. Compiling such a routine will give you a <LABELREDEF> error. You will need to eliminate or change one of the line tags, and make any necessary code changes to maintain the same application logic.
- Use %RFIND to search for the following character strings:
  - $V (View)
  - $Z (system functions generally)
  - ^%
  - ^[
  - ^|
  - ^ (; note space between the "^" and the "(" (SPACE operator!)
  - O 51, O 52, O 53 or O 54 (Open number rather than name)

See the appendix "MSM and Caché Utilities Catalog" for a list of MSM commands, functions, operators, preprocessor directives, special variables, and structured system variables, and their Caché equivalents.

4.4.1 MSM-XCALL Functions

XCALL functions allow the use of external programs, such as C or Assembler, to be called from within M. For backward compatibility support, MSM also supports a ZCALL interface. Both XCALLs and ZCALLs are similar in concept and in implementation to Caché's **$ZF()** Callout interface (see *Using the Caché Callout Gateway*). XCALL and ZCALL functions must be converted to Caché **$ZF()** functions.

4.4.2 OS Functions via %OS and $ZOS

MSM's %OS utility and $ZOS function allow many OS-related functions to be performed from within M. There are no direct equivalents to %OS or $ZOS in Caché, although through the use of $ZF(-1), $ZF(-2), or a pipe to an OS command (using the "Q" parameter to the OPEN command), you can easily execute any OS functions you wish.

Generally speaking, any file related operations (as used in $ZOS for example) should be replaced with method calls to the %Library.File class. For example, to delete a file:

```
Set status=##class(%Library.File).Delete("filename")
```

Caché's $ZF(-1) function allows you to run OS commands from a Caché programmer's prompt. On Windows platforms, these commands will hang if they expect user input. For more details see "Issuing Operating System Commands" in *Using the Caché Callout Gateway*.

4.4.3 Converting ZWINTERM Calls

MSM allows you to use pop-up windows to display messages via the ZWINTERM interface. With Caché, you have several options to gain this same character-based windowing functionality.

- Use mnemonic spaces to create pop-up windows .
- Use the $ZF(-1) function to issue an operating system level command to open a new terminal window.
- Keep MSM as a network client to Caché so that your existing character windowing functions can be used. This is a good short-term goal while your migration to Caché is in progress.

4.4.4 MSM Preprocessor Directives

Both MSM and Caché provide mechanisms by which you can use preprocessor directives to perform tasks such as defining macros, defining macro libraries, and including source code from another routine.

Caché stores its directives in .MAC and .INC code. The MSM globals ^ZMSMMAC and ^ZMSMSRC, which store the preprocessor directives and source code respectively, have conceptual equivalents in Caché's ^rMAC and ^ROUTINE globals.

While MSM and Caché have similar facilities conceptually, the syntax varies between the two systems. For example, in MSM you can use one of two mechanisms to build formal parameter lists for your macros:

- Through the use of enumerated variables, such as $1, $2, ... $N, as the arguments' formal names:

  ```
  #define copyright() Copyright $1-$2
  ```

- Through alphanumeric names as the arguments' formal names:

  ```
  #define copyright(from,to) Copyright from-to
  ```

Caché does not support the use of enumerated variables for macro preprocessing. Instead, you will need to adopt Caché's method of referencing alphanumeric names. The MSM examples above would look like this in Caché:

```
#define copyright(%var1,%var2) "Copyright ",%var1,"-",%var2
```

Arguments of the formal parameter list in Caché must begin with a % sign. Caché does not support MSM's $0 variable, which stores the number of arguments passed to a given macro.

MSM and Caché also differ in the way that macros are referenced from within the application. Using the above macro definition, in MSM you could reference the copyright macro using one of two methods:

- Passing arguments in a parenthesized parameter list

  ```
  %%copyright(1997,1998)
  ```

- Passing arguments as a comma-delimited list

  ```
  #%copyright 1997,1998
  ```

In Caché, the %% and #% syntax for referencing macros is not supported. You would call the copyright macro using this syntax:

```
$$$copyright(1997,1998)
```

4.4.5 Extended References for Globals and Routines

As mentioned earlier, Caché relies on namespaces to access data from the actual physical database volumes. It is strongly recommended that you adopt Caché's namespaces from within your application. In some cases however, changing every hard-coded extended reference in the application will be a difficult short-term goal.

Caché supports several forms of extended references:

For globals:

```
^["namespace"]global
^["namespace",""]global
        (this is useful if piecing apart an MSM UVI/VOL string)
^["directory","system"]global
^|"namespace"|global
^|"^system^directory"|global
```

For routines:

```
DO ^|"namespace"|routine
DO ^|"^system^directory"|routine
JOB ^routine["namespace"]
JOB ^routine["namespace",""]
JOB ^routine["directory","system"]
JOB ^routine["^system^directory"]
```

You might want to create a Caché namespace called MGR that uses the CACHESYS database as the default location for globals and routines. Extended references that expect MGR to contain system globals and routines will not need modification.

4.4.6 Handling End-of-file Situations in Caché

MSM uses the special variable $ZC to store status information from the last device access. In particular, MSM applications made wide use of $ZC to check if the READ command reached the end of a sequential file ($ZC returns -1 in this case). Caché, unlike MSM, triggers an <ENDOFFILE> error in this situation. To handle an End-Of-File situation in Caché, you will need to create a custom error trap in your application using $ZTRAP.

Caché also supports the $ZC (if used in MSM language compatibility mode) as well as the $ZEOF (native Caché language mode) special variables.

4.5 MSM and Cache Database Networking (DDP)

There are a number of differences between MSM database networking (DDP) and Caché database networking (DSM-DDP/DCP/ECP).

- MSM can communicate with Caché only using DSM-DDP over a raw Ethernet (sometimes referred to as a MSM Data Link) link. This link will have different performance characteristics than a MSMV3 link. For MSM it will be slower.
- Caché networking does not support the MSM RVG concept. The Caché ECP protocol uses similar net traffic optimizations as the MSM RVG on all connections, but it is not compatible with MSM.
- Only the MSM DDP protocol supports jobbing with parameter passing.
- When communicating with Caché using DSM-DDP, you can only access data sets (databases) and not namespaces. This means that namespace mapping in Caché cannot be used by MSM.
- When starting remote jobs on Caché from MSM, you cannot start jobs that will make global access from a namespace. MSM can only start jobs that have a default data set (database). This means that there is no namespace mapping in Cache by default for jobs started from MSM.

5

Connecting Applications to Caché

5.1 Connection Tools

Depending on the nature of your application, you might be using a variety of tools to connect your user interface to the database.

- MSM-Activate
- MSM-SQL, KB-SQL, and M/SQL
- MSM-PDQWeb
- MSM-Workstation

5.1.1 MSM-Activate

MSM-Activate provides the means to call M functions, run XECUTE commands, or perform low-level operations on an MSM Server from any of the following methods:

- A Windows DLL callable from C
- An ActiveX control or a COM object
- A C library from various UNIX® platforms
- A set of Java classes and an associated Java Bean

Regardless of the connection method you choose, the clients all connect to a listening process on the MSM server via TCP/IP.

MSM-Activate's server code has been ported to Caché, enabling you to run your MSM-Activate applications against Caché servers without any client-side application changes.

InterSystems recommends that, as a long-term goal, you convert your MSM-Activate based applications to native Caché technology. Depending on the tool you have used to build your client interface, you will have different Caché options available.

- *Caché objects* allow you to use a number of different approaches, including C++, any Active X client (such as Visual Basic, PowerBuilder, or Delphi), or any Java-based tool (such as J++, Visual Café, or JBuilder). See the various Caché Binding books for details.
- *Caché Direct*, which is conceptually quite similar to MSM-Activate's Active X component, allows you to use any tool that can manage an OCX. See *Using Caché Direct* for detailed information.

- *Caché Callin interface* is very similar to MSM-Activate's Call-In Interface, and allows you to access Caché from C programs on all platforms. See *Using the Caché Callin API* for detailed information.

5.1.2 MSM-SQL, KB-SQL, and M/SQL

For relational access, you have probably chosen one of two relational environments for MSM:

- MSM-SQL (which is just a badged version of older versions of KB-SQL)
- InterSystems M/SQL (also known as the FDBMS)

Since the globals containing your data dictionaries will have been already migrated from MSM to Caché, you will be able to convert the relational environment from MSM to either Caché SQL or version F.17 of M/SQL for Caché directly on your Caché system.

- For the KB-SQL contact your InterSystems Account Manager for access to the latest version of the data dictionary converter. This will take the ^SQL dictionary global and convert the table mappings into Caché objects class definitions. These definitions automatically expose your data as relational tables.
- For the M/SQL FDBMS, you will need to do two steps to convert to Caché's DBMS. These steps *must* be run from any namespace that contains the FDBMS objects.
  - Use the Conversion Manager, found under the System Management menu option of the ^%msql utility, to run any necessary DBMS conversion steps.
  - Recompile all DBMS objects by running the command:

    ```
    Do all^%mcompil
    ```

  - Run the F to CDL Export Utility/

Once your data dictionaries have been converted to F.17 or greater of Caché, you may still convert these F data dictionaries to the latest Caché SQL, which is highly integrated with Caché objects.

For ODBC connectivity, MSM and Caché function in a similar fashion, though you will typically benefit from much-improved overall relational performance and lower running costs. See the *Using Caché SQL* guide for more information on ODBC.

For information on converting your MSM-SQL and KB-SQL data dictionaries, contact your InterSystems Account Manager.

If you have implemented your application using M/SQL for MSM, contact your Account Manager or the WRC for further information on migrating M/SQL Forms to Caché.

5.1.3 MSM-PDQWeb

MSM-PDQWeb connects web servers with MSM servers. It enables processing requests to be sent from the web to MSM, which replies in the form of static or dynamically generated HTML. MSM-PDQWeb is quite similar, in both purpose and implementation, to InterSystems WebLink technology, which in turn has been superseded by Caché Server Pages (CSP). WebLink is no longer being actively enhanced by InterSystems.

PDQWeb and WebLink are converged in Caché, enabling applications written for either technology to run unchanged or with only minor changes. This convergence delivers the two most frequently requested PDQWeb enhancements to MSM customers:

- Support for state-aware connections (in addition to the currently available stateless connections).
- Support for a wider variety of web servers.

There are some important steps you should follow for running your PDQWeb application in Caché. Depending on your web server platform, the steps will vary. See the *Caché WebLink Guide* for detailed information on this conversion process.

CSP is recommended as the preferred approach to building browser-based applications, and should be used instead of WebLink if at all possible.

5.1.4 MSM-Workstation

In the area of GUI development tools, InterSystems and Micronetics strategies differed greatly. Where Micronetics focused on developing its own interactive development environment, InterSystems focused on leveraging mass market GUI development tools such as Visual Basic and Delphi.

We strongly recommend the use of Caché Object technology, in place of your current MSM-Workstation applications. With Caché objects, any industry-standard GUI or web development tool can be used for all of your GUI development projects. We believe that through use of Caché objects, in conjunction with industry-standard tools, you will be able to more effectively keep up with the rapidly changing industry.

Caché currently offers some of MSM-Workstations main features:

- A graphical debugging environment
- A graphical routine editor
- Royalty-free licensing for single user systems

Caché does not support the M/WAPI standard defined by the MDC.

For more information, contact your InterSystems Sales Representative.

5.2 Terminal Servers

5.2.1 TELNET and LAT Terminal Servers

Caché supports terminal servers as a method of connecting terminals to your application. These terminals can connect via TELNET or LAT using Caché's built-in terminal capabilities. To configure TELNET and LAT connectivity, use **[Home] > [Configuration] > [Advanced Settings]** page of the Management Portal. (Note that Caché Telnet settings apply only to Windows configurations in which InterSystems supplies the Telnet servers.)

For more information on using terminal devices, see the *Caché I/O Device Guide* chapters on I/O Devices and Commands and Terminal I/O.

Under Windows, there does not appear to be a standard way to tie a certain physical terminal server port to a particular terminal device. This means that MSM applications using $IO to identify a specific user or device may need to be modified. There are several solutions to this problem:

- Tie individual or groups of terminals to a particular value of the special variable $ZIO. For a terminal device, $ZIO will contain the TELNET port number and host IP address, or LAT server name and port name.
  - Identify a specific terminal by creating a Caché user account whose name is made up of the terminal's corresponding server or domain name and port (taken from the $ZIO special variable). From within the application, many references to the $IO special variable will then need to be changed to either the $ZIO variable or another application defined variable.
  - You can also tie this terminal device to a particular routine via the User Accounts utility.

- Use the dumb terminal's *answerback* feature to get a unique ID. For example, on a DEC VT420, set answerback in Setup|Comm|Answerback, and retrieve the string with $character(5).
- If available on your terminal server, you can use telnet to report session and port information. There may be a need to do some scripting, either at the NT or Caché levels, which queries the terminal server for this information and parses the data that the terminal server returns.

Caché's TELNET and LAT services do not run on Windows 95 or 98 platforms. For this functionality, you must use another Windows platform such as NT, 2000 or XP. LAT is only supported directly by Caché on these Windows platforms. For LAT support on non-Windows platforms, a third-party product would need to be used.

The load balancing algorithms differ between MSM and Caché. The MSM LAT Service Rating (LATSR) is calculated based upon current process load and maximum process load on a given server. The Caché LATSR is calculated by taking a specific figure defined by the system manager and multiplying that by the current CPU load on the system, thus producing a figure that is higher or lower than the figure set by the system manager.

The Caché LATSR is set to a value of one by default. Thus, connecting an MSM and Caché application/compute server side by side with both advertising the same LAT service will almost certainly result in every user logging into the MSM server!

To avoid this, simply increase the Caché LATSR level to something more appropriate based upon the known range of the MSM server's LATSR.

5.2.2 Serial Port Expander Boards

Caché for Windows fully supports the use of any intelligent serial port expander board produced by Digi International, provided the board has a supported driver for Windows NT. Prior to using your Digi board with Caché, you must make sure the Auto-Start COM Ports box is checked. This option is found on the Terminal tab of the System Configuration Utilities. You will need to restart Caché after enabling this option. This launches COM port listener routines on the specified ports that will listen on specified ports allowing user login.

Unlike MSM, most Caché devices can be used dynamically and do not need to be predefined in the system setup.

5.2.3 Conversion of MSM Terminal Device Characteristics

The following table lists the equivalent Caché OPEN/USE protocols for MSM terminal device characteristics:

Bit Pos	Decimal Value	MSM Name	Cache Equivalent (name)
31-28	n/a	Reserved	*** n/a ***
27	1.3E+08	Pass-All Flow	Enabled in image mode if supported by device.
26	6.7E+07	Prompted Read	Part of /FLUSH or "F" protocol (Flush).
25	3.4E+07	Type Ahead	See /FLUSH or "F" protocol (Flush).
24	1.7E+07	ZUSE	No ZUSE available in Cache.
23	8388608	Pass All	See /IMAGE or "I" protocol (Image Mode).
22	4194304	No <XOFF>	Cache uses OS settings.
21	2097152	<CTRL_O>	Controlled by OS.
20	1048576	Control Character	No direct correlation.
19	524288	Empty Line Delete	Not supported in Cache.

Bit Pos	Decimal Value	MSM Name	Cache Equivalent (name)
18	262144	Data Length	Controlled by OS.
17	131072	$X and $Y Update	Controlled through /XYTABLE or "Y\name\" protocol ($X/$Y Action Mode).
16	65536	Type of CRT	See /CRT or "C" and "P" protocol (CRT Terminal / Printer Device).
15	32768	Interrupt	Not supported in Cache.
14	16384	Lowercase	Controlled through /TRANSLATE or "K" protocol (I/O Translation Mode).
13	8192	Line Feed	Controlled in configuration for device subtype.
12	4096	<TAB> Control	Controlled in configuration for device subtype.
11	2048	Line Carrier	Supported in Windows only, see $ZA and COMMCTRL.
10	1024	Printer	See /CRT or "C" and "P" protocol (CRT Terminal / Printer Device).
9	512	Connect	Supported in Windows only, see COMMCTRL.
8	256	Logon	Supported in Windows only, see COMMCTRL.
7	128	Cursor Position	Not supported in Cache.
6	64	Escape	Not supported in Cache.
5	32	<CTRL_S>	Controlled by OS.
4	16	<CTRL_O>	Controlled by OS.
3	8	Modem Control	Supported in Windows only, see COMMCTRL.
2	4	CRT	See /CRT or "C" and "P" protocol (CRT Terminal / Printer Device).
1	2	Output Only	Not directly supported in Cache, in Windows see COMMCTRL.
0	1	Echo	See /ECHO or "S" protocol (Secret Input).

6

Caché System Management

6.1 Configuring Caché

To configure Caché, you use the Management Portal. The *Caché System Administration Guide* includes detailed information on how to use the manager, and also highlights the minimum and maximum values for each parameter. Alternatively, you can access Caché's online help by pressing the **<F1>** key with a particular field within the utilities selected.

Tips and tricks on basic system configuration:

- Check that Maximum # of User Processes equals the total process count of your Caché license, unless you want fewer users to gain access.
- Keep your partition size at a sensible level (the default is 1MB), unless your application requires larger values. If you set this value too large, you will use memory and swap space inefficiently. Caché partitions are fixed size and do not expand and contract as do MSM partitions. You cannot dynamically change the partition size because there is no %PARTSIZ equivalent utility in Caché.
- In many cases, increasing the number of Global and Routine Buffers is the best and quickest way to improve performance.

See the appendix "MSM and Caché Utilities Catalog" for a list of MSM utilities and their Caché equivalents.

6.2 Configuring Devices

In most cases, devices must first be set up at the operating system level. Once configured at the OS level, you can immediately begin using these devices from within Caché. If you need to set up mnemonic names or numeric aliases for these devices, use the device configuration utility in Management Portal. These Caché-level device configurations are stored in the cache.cpf file. At each Caché startup, the system will read the cache.cpf file and recreate the system level globals in the CACHESYS database.

Tips on configuring devices:

- Caché has a set of reserved, built-in device numbers which are generally different from MSM. See the *Caché I/O Device Guide* for more information.
- You only need to enter devices into Caché's device tables if you want to access them via a mnemonic name, such as SUN, or a numeric alias, such as 100 (common on MSM systems).
- Mnemonic names are used by the character-based utility called ^%IS, which is used by utilities such as ^INTEGRIT and ^%G.

- Numeric aliases are used directly by Caché's OPEN, USE, and CLOSE commands. The use of aliases is the best way to get MSM-like device handling.
- If you do not require the use of mnemonic names or numeric aliases, you can still access your devices through either the ^%IS utility or through OPEN, USE, and CLOSE commands. For example, the command OPEN "/dev/rmt0":"R" is perfectly valid, provided that /dev/rmt0 is a valid device on your system.
- Pipes are a very effective way of accessing printers and other devices on your machine. See the *Caché I/O Device Guide* for details on setting up and using pipes.
- If you use Caché's SPOOL device, it might be a good idea to store the ^SPOOL global in an isolated location so that it does not take up space in your production environment. You can then reference this global through a namespace configuration.

See the *Caché System Administration Guide* for information on configuring devices in Caché.

6.3 Automating Caché Backups

Caché backups and restorations are designed to run on live systems. These backups can either be system wide, on a per-database basis, or on globals and routines individually. Automating Caché backups from the OS level can be done with a few considerations. There are four recommended strategies:

1. An OS scheduler can call into Caché's backup API, performing Caché's concurrent backup. Any of Caché's backup strategies are available through this API, including full, cumulative, and incremental backups. This strategy is recommended for live automated backups.
2. Caché can be brought down via OS scripting, and then an OS level backup can be run.
3. Caché's databases can be frozen while an OS level backup is being performed.
4. An OS level backup can perform a full backup of a live Caché database. For this strategy, a valid cumulative backup must also be performed immediately after to ensure physical integrity. The Caché backup API can be called here to automate the cumulative backup. The steps for this procedure are as follows:
   - As a pre-backup command, clear incremental bitmaps:

     ```
     Set x=$$CLRINC^DBACK(1)
     ```

   - Run the OS backup on the live system
   - As a post-backup command, perform cumulative backup:

     ```
     Set x=$$BACKUP^DBACK(Arg1,Arg2,...Arg10)
     ```

   This option requires that your OS-level backup allow files to change while the backup is being performed. Choose your OS backup software with care.

Restoration of system backups is performed via the character-based utility called ^BACKUP. See the *Caché System Administration Guide* for more information on the basic backup types, and how to perform them.

6.4 Caché Journaling

Journaling in Caché is very similar to what you are used to in MSM. When used in conjunction with backups, it is the best mechanism for bringing a system as up-to-date as possible after a system failure. Journaling is also used to keep track of your application's transactions. For information on journaling, see the High Availability Guide.

The Before Image Journaling (BIJ) feature is equivalent to the Caché feature called Write Image Journaling (WIJ). This is automatically enabled on Caché systems. It consists of a single file (unlike MSM that has a separate BIJ file for each bullet proofed volume group) named cache.wij, located in the \cachesys\mgr directory. Its size is not fixed and grows as necessary to accommodate modified blocks.

After Image Journaling is also automatically enabled. Unlike MSM, Caché journal files are created as needed and do not have to be defined in advance. Each time Caché is restarted a new journal file is begun. The "age" at which a journal file is automatically deleted is determined by settings that can be modified in the Management Portal.

Journaling Tips:

- A process can enable or disable journaling for itself via the ENABLE and DISABLE line tags of ^%SYS.NOJRN, respectively.
- Do not use the *Journal All Globals* option unless you really need to. Choosing this option will journal every global in your database, which can lead to an extraordinarily large journal file, reduced system performance and increased network traffic if Shadow System Journaling is employed. Temporary globals that can be deleted upon system restart should never be journaled.
- It is a good idea to switch your journal files after each backup. This process can be automated.
- Under a shadow system journaling configuration, a global READ will access only the local version of the global, unless an extended global reference is used.

6.4.1 Shadow System Journaling

While Caché's Shadow System Journaling is very similar conceptually to MSM's Cross-System Journaling, you will find that Caché's Shadow System Journaling is much more feature-rich. For example, in Caché you have two modes of data transfer available to you:

1. Fast (previously block-mode) Transmission
    - The shadow connects to the database server via TCP and captures the live journal flat file.
    - Acquired transactions are optionally applied to the Shadow machine. Choosing not to apply the acquired transaction is a good way to keep redundant journal files.
    - Since many transactions are captured at once via a binary journal block, block-mode tends to be quicker than record-mode.
    - Applied transactions are optionally logged in the shadow machine's local journal file.
2. Compatible (previously record-mode) Transmission
    - The shadow connects to the database server via TCP and captures transactions via packaged strings.
    - Acquired transactions can be programmatically scanned before applying the transactions. You have access to the following information when scanning:
        - Address of current record
        - Transaction type, such as SET or KILL

  - Global reference, if any
  - New value to which global is set

- Since transactions are captured via packaged strings, record-mode tends to be slower than block-mode.
- Applied transactions are optionally logged in the shadow machine's local journal file.
- Record mode shadowing can be employed when differing endian systems are to be linked (e.g. Intel and Sun), unlike block mode shadowing, which is limited to transmissions of the same endian type.

The transport and delivery mechanism differs between MSM and Caché. MSM utilizes a "push" mechanism via DDP -- individual sets and kills are applied as regular cross system updates to the shadow server. Caché instead pulls the data from the primary server to the shadow server via TCP/IP.

The Caché use of native TCP results in benefits such as much improved shadowing performance and easier setup of WAN support. It also allows enhanced capabilities such as shadowing across the Internet and the ability to define multiple shadow servers for each primary server.

6.4.2 Shadowing and Switching from MSM to Caché

Given that MSM Cross System Journaling is implemented using simple DDP cross-system sets and kills (i.e. using extended global references), and Caché supports DDP, a MSM server can in fact shadow to a Caché server. This can be a big help in preparing for a switch over with minimal down time of a production system.

For example, consider the following steps when replacing a MSM Primary/Shadow pair with its Caché equivalent:

1. Take the MSM shadow off line, back it up, enable journaling (or switch journal spaces if already enabled).
2. Restore the backup onto the Caché server.
3. Convert the MSM UCIs to Caché databases (see previous chapters).
4. Configure DDP between Caché and MSM so that MSM can see the Caché databases.
5. Configure Cross System Journaling on the MSM shadow to point to the correct databases on the Caché primary.
6. Enable Cross System Journaling on the MSM shadow.

Updates will then begin to arrive on the Caché primary and will continue to do so. Of course a Caché shadow can be created (using the original Caché databases created in #3 above), and connected to the Caché primary.

Once the systems are stable and a switchover date and time has been set, disable all access to the MSM primary server, allow all updates to filter down to the Caché primary, shut down the MSM servers (to be safe), and then give access to the users to the new Caché primary instead of the old MSM one. The last step maybe as simple as changing an IP address of a server or in the client configuration.

Using this technique can result in a total downtime literally measured in minutes.

A

MSM and Caché Utilities Catalog

This appendix lists MSM commands, functions, operators, preprocessor directives, special variables, and structured system variables, and their Caché equivalents. Bear in mind that many of the Caché equivalents mentioned here are deprecated or have been rendered obsolete by more modern tools. They are presented here only to provide you with the most direct substitutes when porting legacy code. Documentation for most of the utilities mentioned here can be found in the "Legacy Documentation" chapter in *Using InterSystems Documentation*.

%ACTJOB

Caché equivalent: No direct equivalent

Purpose: Provides a ^-delimited list of all job numbers in the system.

Notes: In Caché, use the following code:

```
Set (j,p)=""
For{
   Set j=$Order(^$Job(j))
      ;$Order(^$JOB(j)) is recommended over $ZJOB(j)
   Quit:j=""
   Set p=p_j_"^"}
```

%CHKSUM

Caché equivalent: $ZCRC

Purpose: Computes a checksum (ASCII summation) of one or more routines.

%D

Caché equivalent: %D

Purpose: Displays the date currently stored in $HOROLOG.

Notes: %D in MSM reports the date in the format DD-MMM-YY, while Caché uses the format MMM-DD-YY. Use $ZDATE($HOROLOG,2) to mimic MSM's %D output in Caché.

%DEBUG

Caché equivalent: No Direct Equivalent

Purpose: Invokes an interactive program debugging facility.

Notes: See the *Caché ObjectScript Reference* for information on BREAK and ZBREAK.

%DEVUSE

Caché equivalent: No Direct Equivalent

Purpose: Displays a list of all opened devices and the number of the job that owns each one.

Notes: In Caché, use TTYFREE to check reserved TTY devices and the processes that own them. Use %SS to see all processes and the devices they have open.

%DH

Caché equivalent: %DX

Purpose: Converts a decimal value to hexadecimal.

%DI

Caché equivalent: %DATE

Purpose: Converts a date from external form (for example: 8-SEP-97) to internal HOROLOG format.

Notes: In Caché, you can call %DATE programmatically via the INT line tag.

%DO

Caché equivalent: $ZDATE($H_Value)

Purpose: Converts a date from internal $HOROLOG format to external format.

%ECHO

Caché equivalent: No Direct Equivalent

Purpose: Allows the program to control the echoing of characters at the terminal. Entry points are provided to turn ECHO on and off.

Notes: In Caché, use the Secret-Mode feature of Caché's terminal I/O. For example, to hide the user's input from a Caché program, try:

```
Use 0:(:"s")
Read rec
```

%EDP

Caché equivalent: Not Available

Purpose: Performs macro lookup, expansion, and parameter substitution.

%EDP1

Caché equivalent: Not Available

Purpose: Processes directives, has supplementary entry points.

%ER

Caché equivalent: %ER

Purpose: Displays error information trapped by the %ET routine.

%ERRCODE

Caché equivalent: Not Available

Purpose: Display an explanation for database-specific error codes.

%ET

Caché equivalent: %ET, %ETN

Purpose: Error trap routine

Notes: Caché's %ET(%ETN) routine is much more feature-rich.

%FGR

Caché equivalent: %GIF, %SYS.GIFMSM

Purpose: Fast global restore (block format).

Notes: Use %SYS.GIFMSM to import %FGS-format global saves into Caché.

%FGS

Caché equivalent: %GOF

Purpose: Fast global save (block format)

%FL

Caché equivalent: %RFIRST

Purpose: Display the first line of code for selected routines.

%FLIST

Caché equivalent: Not Available

Purpose: Lists a file stored in the host file system.

%GCH

Caché equivalent: %Library.Global class, %GDISP (not available in Caché 5.0 or later), PROTECT

Purpose: Display characteristics of global(s) and modify

%GCHANGE

Caché equivalent: %GCHANGE

Purpose: Changes all occurrences of a string in one or more globals.

%GCMP

Caché equivalent: %GCMP

Purpose: Compares two globals in the same or different namespace.

%GCOPY

Caché equivalent: %GCOPY, MERGE

Purpose: Copies one or more globals from one namespace to another. The namespace may be on the same machine or on a remote machine.

%SYS.GD

Caché equivalent: %SYS.GD

Purpose: Display global directory for current namespace.

%GDE

Caché equivalent: %Library.Global class, %GDISP (not available in Caché 5.0 or later)

Purpose: Provides an extended global directory display.

%GDEL

Caché equivalent: Not Available

Purpose: Deletes one or more globals from a namespace.

%GE

Caché equivalent: INTEGRIT, BLKDIST

Purpose: Display efficiency of global(s).

%GEDIT

Caché equivalent: Management Portal

Purpose: The System > Globals > Edit Global Data page allows an administrator or operator to edit and delete global data values.

%GL

Caché equivalent: %G

Purpose: Lists all or selected portions of a global file.

%GR

Caché equivalent: %GI, %GIGEN, %GIF

Purpose: Restore global(s) from a device, and allows them to be renamed.

Notes: Caché does not allow the renaming of globals on import.

%GS

Caché equivalent: %GO, %GOGEN, %GOF

Purpose: Saves all or selected portions of one or more globals to a device.

%GSE

Caché equivalent: Management Portal

Purpose: The System > Globals > Edit Global Data page allows an administrator or operator to view globals and search for global data values.

%GSEL

Caché equivalent: %SYS.GSET

Purpose: Allows you to select one or more globals from the current namespace.

%GSIZE

Caché equivalent: %GSIZE

Purpose: Display size of one or more globals

%GUCI

Caché equivalent: %DIR

Purpose: Returns the three-character name and internal UCI number for the current UCI.

Notes: Caché's %DIR will report the current namespace and default global directory.

%HD

Caché equivalent: %XD

Purpose: Converts a hexadecimal number to a decimal.

%HELP

Caché equivalent: No Direct Equivalent

Purpose: Provides online help for character-based utilities.

Notes: In Caché, you can get online help by entering a "?" at any of the character-based utility prompts.

%HL

Caché equivalent: %PRIO

Purpose: Allows you to change the priority of the current job from high to low or from low to high.

Notes: Call Caché's %PRIO utility through the LOW, NORMAL, and HIGH line tags.

%HOSTCMD

Caché equivalent: $ZF(-1,"CMD")

Purpose: Allows you to issue host operating system commands from within an M program.

%INDEX

Caché equivalent: Not Available

Purpose: Provides a cross-reference listing of one or more routines, and optionally provides a structured program listing of selected routines.

%LOGON

Caché equivalent: ^%CD, ZN, $ZU(5)

Purpose: Allows you to switch from one UCI to another.

%MDMP

Caché equivalent: Not Available

Purpose: Provides a display in hexadecimal format, character format, or both for selected memory locations or the VIEW buffer.

%MFUNC

Caché equivalent: ^%math

Purpose: Provides mathematical functions including E, PI, SIN, and COS.

Notes: ^%math must be called by the appropriate line tag—see source code.

%MODESET

Caché equivalent: Not Available

Purpose: Allows you to change environmental mode flags such as maximum length of routine lines.

%MTCHK

Caché equivalent: Not Available

Purpose: Allows you to interrogate the status of a magnetic tape drive.

%NEWED

Caché equivalent: %RD

Purpose: Lists routines that were filed by the program editor during a specified range of dates.

%OS

Caché equivalent: No Direct Equivalent

Purpose: Performs operating system-specific tasks.

%PARTSIZ

Caché equivalent: Not Available

Purpose: Allows you to dynamically change the partition size of the current job.

%RCHANGE

Caché equivalent: %RCHANGE

Purpose: Changes all occurrences of a string in one or more routines.

%RCMP

Caché equivalent: %RCMP

Purpose: Compares two routines in either the current namespace or different namespaces.

%RCOPY

Caché equivalent: %RCOPY

Purpose: Copies a routine from one UCI to another.

Notes: Caché's %RCOPY will not allow you to copy the routine to another namespace. This utility renames a routine in the current namespace.

%RD

Caché equivalent: %RD

Purpose: Display a routine directory for the current namespace.

%RDEL

Caché equivalent: %RDELETE

Purpose: Deletes one or more routines from the current namespace.

%RELOAD

Caché equivalent: %RCOMPIL

Purpose: Recompiles one or more routines in a namespace.

%RPRT

Caché equivalent: %RD

Purpose: Prints a listing of one or more routines stored in the current namespace.

%RR

Caché equivalent: %SYS.RI, %SYS.RIMF, %urload

Purpose: Restores all or selected routines from an external device and allows them to be renamed.

Notes: Caché does not allow the renaming of routines on import.

%RS

Caché equivalent: %RO, %ROMF, %urprint

Purpose: Allows one or more routines to be saved on an external device.

%RSAND

Caché equivalent: %RFIND

Purpose: Searches one or more routines for occurrences of one or more character strings.

Notes: Unlike MSM, if more than one string is specified in Caché, each string may be anywhere in the routine. MSM requires that both strings be on the same line.

%RSE

Caché equivalent: %RFIND

Purpose: Searches one or more routines for any occurrence of one or more character strings. If more than one string is specified, any one of the strings found satisfies the search.

%RSEL

Caché equivalent: %RSET

Purpose: Allows you to select one or more routines from the current namespace.

%RSIZE

Caché equivalent: %RD, $$^%ROUOBJ(…)

Purpose: Displays the number of blocks used by selected routines.

%SBP

Caché equivalent: Not Available

Purpose: Displays the current status, block location, and buffer offsets for the Sequential Block Processor device.

%SDEV

Caché equivalent: %IS

Purpose: Allows you to select and open a device, and specify the OPEN parameters.

%SI

Caché equivalent: %SS

Purpose: Displays general system information, including the status of system-related processes.

%SP

Caché equivalent: %FREECNT

Purpose: Displays the total amount of disk space within a volume group and the amount of free space.

%SQRT

Caché equivalent: sqr^%math

Purpose: Computes the approximate square root value of a number.

%SS

Caché equivalent: %SS $V(-1,PID)

Purpose: Displays status information about each job currently active on the system.

%T

Caché equivalent: %T

Purpose: Displays the time stored in $HOROLOG in the form HH:MM.

Notes: In Caché, use the INT tag to programmatically get the time.

%TI

Caché equivalent: %TI

Purpose: Converts a time value in external format (for example: 1:05 P.M.) to an internal $HOROLOG format.

Notes: In Caché, use the INT tag to programmatically get the $HOROLOG value.

%TO

Caché equivalent: $ZTIME

Purpose: Converts a time value from internal $HOROLOG format to external format.

%TRANS

Caché equivalent: Not Available

Purpose: Enables you to transfer routines and globals between machines. Includes all of the necessary controls (checksums) to ensure proper transmission of the routines and globals.

%UTL

Caché equivalent: ^UTIL

Purpose: Provides a way to invoke most MSM utilities, based on the type of function to be performed.

%VIDEO

Caché equivalent: Not Available

Purpose: Allows users to modify contents of the PC Console video buffer. (MSM-PC/PLUS and MSM for Windows only).

%XMIT

Caché equivalent: Not Available

Purpose: Enables communication with another port on the system; this is useful for transferring information between machines.

%ZSTIME

Caché equivalent: %RD

Purpose: Displays the last-saved time of one or more routines.

APIMGR

Caché equivalent: No Direct Equivalent

Purpose: MSM-Activate management utilities

Notes: In Caché, use the Caché Direct Client and Server Management Utilities.

BCS

Caché equivalent: BROADCAS, $ZU(9), $ZU(94)

Purpose: Broadcast messages to other terminals or process IDs

BIJ

Caché equivalent: Management Portal

Purpose: Manage Before Image Journaling (Write Image Journaling)

DBMAINT

Caché equivalent: MSU, MOUNT, DISMOUNT, etc.

Purpose: Perform database maintenance

GLBPLACE

Caché equivalent: %SYS.GCREATE

Purpose: Create a new global

JOBEXAM

Caché equivalent: JOBEXAM

Purpose: Display detailed information for a process

JRNL

Caché equivalent: JRNSTART, JRNSTOP, JRNDUMP, JRNSWTCH, JRNRESTO, %SYS.NOJRN

Purpose: Manage After Image Journaling (Flat File Journaling)

KILLJOB

Caché equivalent: RESJOB, , $ZU(4)

Purpose: Terminate a job

LOCKTAB

Caché equivalent: LOCKTAB

Purpose: Display all active locks in the system

OLB

Caché equivalent: BACKUP

Purpose: Perform concurrent backup of databases

OLC

Caché equivalent: GCOMPACT

Purpose: Perform online compression

PEEK

Caché equivalent: Not Available

Purpose: Monitors another terminal device's activity.

RECOVLCK

Caché equivalent: LOCKTAB

Purpose: Recover a lock

SETBAUD

Caché equivalent: Not Available

Purpose: Temporarily modifies terminal characteristics such as number of data bits, number of stop bits, parity, and baud rate of a terminal.

SSD

Caché equivalent: SHUTDOWN, ZSHUTDOWN, %ZSTOP

Purpose: System shutdown utility

STU

Caché equivalent: STU, ZSTU, %ZSTART

Purpose: System startup utility

SYSGEN

Caché equivalent: Management Portal

Purpose: Generate system configuration

UCIMGR

Caché equivalent: Management Portal

Purpose: Manage UCI configurations

VALIDATE

Caché equivalent: INTEGRIT, CHECKPNT, CHECKMAP

Purpose: Check physical integrity of a database

VERIFY

Caché equivalent: See VALIDATE

Purpose: Verify a database's physical integrity

XCALLMGR

Caché equivalent: Not Available

Purpose: Manage XCALL functions

B

M Language Differences

Only language features that differ between MSM and Native Caché mode are listed.

While MSM will allow varied abbreviations for many of its language features, Caché will only allow the abbreviations stated in the *Caché ObjectScript Reference*.

B.1 Commands

BREAK

Caché equivalent: BREAK, ZBREAK

Purpose: Invokes the debugger.

Notes: Not supported from interactive debugger.

ZGO not supported in Caché — use argumentless GOTO instead.

CLOSE

Caché equivalent: CLOSE

Purpose: Closes a device.

Notes: Caché supports MSM-like numeric devices after you define a numeric alias for that device—see OPEN.

DO

Caché equivalent: DO

Purpose: Executes a routine or block of code.

Notes: Square brackets ([]) not supported from routines — use vertical bars instead (||). And, MSM's UCI and VOL values must be changed to namespace and system values, respectively. Or, eliminate routine extended references and use namespace routine mapping (Preferred).

JOB

Caché equivalent: $ZCHILD, JOB

Purpose: Spawns a new background process.

Notes: In Caché, use $ZCHILD to return the PID of the jobbed process rather than $ZB as done in MSM.

Using the JOB command to specify a new partition size [JOB:(:PartitionSize)] is only supported in Caché for UNIX®—For Windows, use $ZF(-2) to spawn an external Caché job with a new partition. For example, this code runs ^Test in the %SYS namespace with a 1024 KB partition size on Windows NT (assumes default installation directory):

```
Set x=$zf(-2, "c:\cachesys\bin\cache -s..\mgr -b 1024 -U %SYS ^^Test")
```

Note these differences between MSM jobs and Caché jobs:

- When jobbing to Caché (or any DSM-DDP system), you cannot pass parameters. Parameter passing across the network is an MSMV3 circuit option only. You need to set passed values into a global on one of the systems and fetch these values in the JOBed process.
- When jobbing from MSM to Caché, you can only start jobs with a default database. That is, you cannot specify a namespace in the JOB command in MSM to start the job on cache in. Once started, the job on the Caché system will need to specifically change (ZNAMESPACE) to a namespace to use mapping.

NEW

Caché equivalent: NEW

Purpose: Stacks one or more local variables.

Notes: Caché does not allow $TEST or $ZREFERENCE as arguments to the NEW command.

OPEN

Caché equivalent: OPEN

Purpose: Opens a device.

Notes: Devices in Caché correspond to device name at the OS level (such as /dev/tty3a), unless a numeric alias is specified in the System Configuration utilities for the device. You must use a numeric alias to emulate MSM's device structure.

TRESTART

Caché equivalent: Not implemented.

Purpose: Causes current transaction to be restarted.

TSTART

Caché equivalent: TSTART

Purpose: Marks the beginning of a transaction.

Notes: Caché does not support restart variables or transaction parameters.

USE

Caché equivalent: USE

Purpose: Uses a device, and sets $IO to this current device.

Notes: Caché supports MSM-like numeric devices after you define a numeric alias for that device—see OPEN.

VIEW

Caché equivalent: VIEW

Purpose: Reads and writes blocks to disk, and writes locations in memory.

Notes: VIEW commands must respect Caché disk and memory structures—see documentation for more details.

WRITE

Caché equivalent: WRITE

Purpose: Sends output to current device.

Notes: Mnemonic spaces may need to be rewritten for Caché.

ZCALL

Caché equivalent: $ZF

Purpose: Calls an external procedure.

ZFLUSH

Caché equivalent: Not implemented.

Purpose: Flushes all disk blocks out of the internal disk buffer cache.

ZGO

Caché equivalent: argumentless GOTO

Purpose: Resumes execution of a program after a BREAK command.

ZHOROLOG

Caché equivalent: the %SYSTEM.Process **FixedDate()** method.

Purpose: Sets date and time for current process.

Notes: Caché's **FixedDate()** method only allows a new date value, not time.

ZMSM

Caché equivalent: No direct equivalent.

Purpose: Traces the sequence of program execution within a routine and from routine to routine.

Notes: In Caché, try:

```
For i=0:1:$stack(-1)  Do
. Write !,"Context level:",i,?25,"Context type:",$stack(i)
. Write !,?5,"Current place:",$stack(i,"place")
. Write !,?5,"Current source:",$stack(i,"mcode")
Quit
```

ZNEW

Caché equivalent: Not implemented.

Purpose: Similar to NEW command, but variable is persistent after subroutine explicitly or implicitly quits.

ZQUIT

Caché equivalent: ZQUIT

Purpose: Passes control to the next higher error-processing routine that has been specified by $ZTRAP.

Notes: Caché clears entire stack, unless an argument representing the number of error trap levels to quit back is specified.

ZSETOBJ

Caché equivalent: SET

Purpose: Assigns an object reference to a variable.

Notes: ObjectScript uses the native SET command, such as:

```
Set var=Car.Make
```

ZUSE

Caché equivalent: the %Library.Device **Broadcast()** method or the %SYSTEM.Process **Broadcast()** method.

Purpose: Allows write access to any device, even if in use — broadcasting

B.2 Operators

#

Caché equivalent: $ZHEX

Purpose: Performs numeric conversions from hexadecimal to decimal.

B.3 Structured System Variables

^$DEVICE

Caché equivalent: Not implemented.

Purpose: Provides information on the existence, operational characteristics, and availability of a device.

B.4 Functions

$ORDER

Caché equivalent: $ORDER

Purpose: Returns next subscript at the same level of a given variable. Also loops through a list of local variables set in a partition.

Notes: This looping functionality differs on the two platforms. For example, assume we have these variables set:

```
%=1, %USER="mikel", var=123.
```

MSM code

```
Write $Order()   ; this returns "%"
Write $Order(%)   ; this returns "%USER"
Write $Order(%USER)    ; this returns "var"
```

Caché code

```
Write $Order(@(""))   ; this returns "%"
Write $Order(%)   ; this returns "%USER"
Write $Order(%USER)   ; this returns "var"
```

$VIEW

Caché equivalent: $VIEW

Purpose: Returns contents of memory locations.

$ZASCII

Caché equivalent: $ASCII

Purpose: Returns the Unicode character code of a specified character.

$ZBN

Caché equivalent: Not available.

Purpose: Returns the starting block number for a routine or global, allocates a disk block, or de-allocates a disk block.

$ZBname

Caché equivalent: $ZBIT<name>

Purpose: A collection of functions that are used to perform logical operations on bitstrings.

$ZCALL

Caché equivalent: $ZF

Purpose: Calls an external procedure and returns a value.

Notes: Caché's $ZF expects function names in double-quotes [$ZF("MyFunction")], while MSM's $ZCALL does not [$ZCALL(MyFunction)].

$ZCHAR

Caché equivalent: Not available.

Purpose: Returns a string of characters, given a list of Unicode character codes.

$ZCRC

Caché equivalent: $ZCRC

Purpose: Returns a computed checksum or cyclic redundancy check.

$ZCREATEOBJECT

Caché equivalent: SET

Purpose: Returns an object reference to a newly instantiated object.

Notes: ObjectScript uses the native SET command, such as:

```
Set var=##class(Car).%New()
```

$ZDATE

Caché equivalent: $ZDATE

Purpose: Returns an external date value, given a $HOROLOG date.

Notes: Caché will return a <VALUE OUT OF RANGE> error for any $H value below 0 and above 2980013. While MSM will not generate an M error for dates out of range, invalid dates are reported for $H values below 0 and after 94598.

$ZDEVICE

Caché equivalent: No direct equivalent.

Purpose: Returns the actual device name, given the internal device ID.

Notes: For a terminal device, the special variable $ZIO contains the TELNET port number and host IP address, or LAT server name and port name. For more information on using terminal devices, see the *Caché I/O Device Guide* chapters on I/O Devices and Commands and Terminal I/O.

$ZGETOBJECT

Caché equivalent: SET

Purpose: Retrieves database object, and returns object reference to the instantiated object.

Notes: ObjectScript uses the native SET command, such as:

```
Set var=##class(Car).%Open(OREF)
```

$ZHL

Caché equivalent: $ZDATE, $ZTIME, $ZDATETIME

Purpose: Returns an external date or time value, given a $HOROLOG date.

Notes: Use $ZDATE to convert dates and $ZTIME to convert times, or $ZDATETIME to convert both.

$ZOBJREFERENCE

Caché equivalent: Not available.

Purpose: Identifies whether an expression refers to an object, and whether two expressions refer to the same object.

$ZOS

Caché equivalent: $ZF(-1), $ZSEARCH, OPEN

Purpose: Invokes commonly used host OS functions from within M.

$ZPOSITION

Caché equivalent: Not available.

Purpose: Returns the number of positions of a string that can fit in a field, on an output device.

$ZUCI

Caché equivalent: No direct equivalent.

Purpose: Returns the UCI internal number or external name.

Notes: In Caché, use $ZNSPACE or $NAMESPACE to return the current namespace.

$ZVERIFY

Caché equivalent: Not available.

Purpose: Returns a string of errors, if any exist, in the logical structure of the database.

$ZWIDTH

Caché equivalent: Not available.

Purpose: Returns the width that a string occupies when it is displayed on an output device.

B.5 Preprocessor Directives

#comment

Caché equivalent: No direct equivalent.

Purpose: Turns on the insertion of pre-expansion lines of code that contain macros into the generated code as comments.

Notes: In Caché, you can use #show to enable the inclusion of comments from .INC code in the generated .INT code.

#defarray

Caché equivalent: Not available.

Purpose: Defines a macro to be used for referencing an array.

#deflabel

Caché equivalent: Not available.

Purpose: Defines a unique local label or variable, and is guaranteed to be unique in a routine as long as the prefix is not used directly.

#include

Caché equivalent: #include

Purpose: Includes source code in a given routine.

Notes: In Caché, #include can only be used to reference .INC code.

#library

Caché equivalent: Not available.

Purpose: Specifies path to library files.

#makelib

Caché equivalent: Not available.

Purpose: Creates a macro library.

#nocomment

Caché equivalent: No direct equivalent.

Purpose: Stops the inclusion of unprocessed source code lines as comments.

Notes: In Caché, you can use #noshow to exclude comments from .INC code from the generated .INT code.

#noroutine

Caché equivalent: Not available.

Purpose: Prevents generation of an M routine.

#prefix

Caché equivalent: Not available.

Purpose: Defines the prefix used to identify a macro reference.

#routine

Caché equivalent: Not available.

Purpose: Specifies the name of a routine to be generated.

#undefine

Caché equivalent: #undef

Purpose: Removes a macro definition.

Notes: In Caché, you must change all #undefine statements to #undef.

#updlib

Caché equivalent: Not available.

Purpose: Updates a macro library.

#x

Caché equivalent: Not available.

Purpose: Executes M code during preprocessing.

B.6 Special Variables

$DEVICE

Caché equivalent: $DEVICE

Purpose: Indicates whether last I/O operation was successful.

Notes: Caché always returns the NULL string indicating a successful I/O operation You can use the SET command to place a value in $DEVICE. By convention, this value should describe the outcome of an I/O operation as a string in the form: `"standard_error,user_error,explanatory_text"`.

$ECODE

Caché equivalent: $ECODE

Purpose: Returns a list of errors encountered by the application.

Notes: While MSM's and Caché's $ECODE are the same conceptually, Caché will use Caché-specific error strings such as: *,ZSYNTAX, ZNOROUTINE, ZDISKHARD,*

$IO

Caché equivalent: $IO

Purpose: Contains the currently active device.

Notes: While MSM will represent $IO as an internal device number, Caché will use an actual device name, with a device type header. For example, a printer in Caché might look something like this:

```
|PRN|\\salesserver\printer1
```

For MSM-like devices, you must create a numeric alias for your device via the System Configuration Wizard.

$JOB

Caché equivalent: $JOB

Purpose: Contains the job number for the current process.

Notes: Caché's $JOB values correspond to the process' PID number at the OS level, while MSM's $JOB values are MSM-specific numbers.

$PRINCIPAL

Caché equivalent: $PRINCIPAL

Purpose: Contains a job's principal device.

Notes: While MSM will represent $PRINCIPAL as an internal device number, Caché will use an actual device name, with a device type header. For example, a user login in Caché might look something like this:

```
|TNT|192.9.204.64:1097|316
```

In this case, |TNT| specifies a TELNET device, 192.9.204.64 represents the TERMINAL server IP, 1097 the virtual port number, and 316 the OS level process ID.

$SYSTEM

Caché equivalent: No direct equivalent.

Purpose: MSM uses $SYSTEM to return three pieces of information: an M User Group # (43), the Serial # from the MSM license, and a unique # for the current instance of M.

Notes: Caché license information can be accessed through methods of the %SYSTEM.License class:

- **KeyMachineID()** returns the contents of the *MachineID* field in the active key.
- **KeyOrderNumber()** returns the active key *OrderNumber* field.
- **KeyAuthorizationKey()** returns the *AuthorizationKey* field in the active key.
- **KeyCustomerName()** returns the active key *CustomerName* field.

These methods are implemented in the special $SYSTEM object and can be accessed with the `$SYSTEM.License.Method()` syntax.

$TRESTART

Caché equivalent: Not implemented.

Purpose: Indicates the number of transaction restarts that have occurred since the initiation of the transaction.

$ZB

Caché equivalent: $ZB, $ZCHILD

Purpose: Returns device-specific information for the current device. When used with the JOB command. MSM's $ZB returns the jobbed process' PID.

Notes: For this functionality in Caché, use $ZCHILD.

$ZC

Caché equivalent: Not implemented.

Purpose: Contains device-specific information for the current device.

Notes: In Caché, $ZC is used to represent both the $ZCHILD special variable and $ZCYC function, depending on context.

$ZERROR

Caché equivalent: $ZERROR

Purpose: Contains the text of the error message most recently produced by the application or programmer.

Notes: Caché will report Caché-specific error text that may or may not correspond to MSM's error text.

$ZLEVEL

Caché equivalent: No direct equivalent.

Purpose: Contains a number that indicates the current nesting level.

Notes: Use $STACK