

MSM to Caché v3.x

Conversion Guide

April 12, 2000

Introduction

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

This document will address all of these concerns. We'll cover a variety of topics ranging from hardware requirements to data migration to system management, and we'll make frequent reference to the InterSystems documentation (most of which is now online) for greater detail on these topics. The documents we'll be referencing are:

- *Caché Installation Guide for Windows*
- *Caché Installation Guide for UNIX and Linux*
- *Caché Installation Guide for OpenVMS*
- *Caché Networking Guide*
- *Caché Basic System Management Guide*
- *Caché ObjectScript Language Reference*
- *Caché Programming Guide*
- *Caché Development Guide*
- *Caché ODBC Driver Guide*
- *Caché Direct Release Information*
- *F to CDL Export utility Guide*

Since 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, you'll want to familiarize yourself with the Caché documentation as much as possible. Whether you view Caché's documentation set from the Technical Information CD or from the books themselves, these references will be your most complete source for moving ahead with Caché. Also, consult the InterSystems web site, <http://www.intersys.com>, for the most up-to-date information on Caché.

Disclaimer

All information written in this document is believed to be accurate as of the date of publication. We welcome any comments or suggestions that you might have regarding this guide. Also, while most of the strategies behind migrating to Caché 3.x 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. Further, 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.

Table of Contents

DISCLAIMER.....	II
1. EXECUTIVE OVERVIEW.....	1
2. GETTING STARTED.....	2
2.1. DETERMINING HARDWARE REQUIREMENTS	2
2.2. DETERMINING NETWORKING REQUIREMENTS	3
2.3. INSTALLING CACHÉ	4
2.3.1. Installing the Caché License Key.....	4
2.3.2. Running Caché and MSM on the Same System.....	5
2.4. ENTERING CACHÉ PROGRAMMER'S MODE	5
3. CREATING CACHÉ DATABASES	6
3.1. OVERVIEW OF DATABASE CREATION.....	6
3.2. CREATING PHYSICAL DATABASE VOLUMES	6
3.3. CREATING A NAMESPACE CONFIGURATION	6
4. CONVERTING MSM GLOBALS	8
4.1. A SUGGESTED STRATEGY FOR CONVERTING GLOBALS BY DATABASE	8
4.2. CONVERTING INDIVIDUAL OR GROUPS OF GLOBALS	9
4.3. PRESERVING MSM STRING COLLATION.....	9
4.4. ENABLING NULL SUBSCRIPT SUPPORT IN CACHÉ	10
5. CONVERTING YOUR MSM APPLICATION.....	11
5.1. MIGRATING ROUTINES	11
5.1.1. Porting MSM Routines Less Than 32 KB in size	11
5.1.2. Porting MSM Routines Greater Than 32 KB	12
5.2. MSM LANGUAGE COMPATIBILITY MODE.....	13
5.3. HANDLING MSM'S IMPLEMENTATION SPECIFIC FUNCTIONS AND FEATURES	13
5.3.1. Generalized approach to get you started.....	14
5.3.2. MSM-XCALL Functions.....	15
5.3.3. OS Functions via %OS and \$ZOS	15
5.3.4. Converting ZWINTERM Calls.....	15
5.3.5. MSM Preprocessor Directives	16
5.3.6. Extended References for Globals and Routines.....	17
5.3.7. Handling End-Of-File Situations in Caché	17
5.4. CONNECTING THE APPLICATION TO CACHÉ	18
5.4.1. MSM-Activate	18
5.4.2. MSM-SQL, KB-SQL, and M/SQL.....	20
5.4.3. MSM-PDQWeb.....	20
5.4.4. MSM-Workstation.....	21
5.4.5. TELNET and LAT Terminal Servers	21
5.4.6. Serial Port Expander Boards	22
5.4.7. Calling into a Caché Routine from the Command Prompt or from a Script File	22
6. CACHÉ SYSTEM MANAGEMENT	23
6.1. CONFIGURING CACHÉ	23
6.2. NETWORKING BETWEEN CACHÉ AND OTHER SYSTEMS.....	24
6.2.1. Ensuring Unique \$JOB Values Across the Network	24

6.3.	CONFIGURING DEVICES	25
6.4.	BACKING UP AND RESTORING YOUR DATA	26
6.4.1.	Automating Caché Backups.....	26
6.5.	CACHÉ JOURNALING.....	27
6.5.1.	Shadow System Journaling	27
APPENDIX A: MSM AND CACHÉ: SUPPORTED PLATFORMS		29
APPENDIX B: HARDWARE RECOMMENDATIONS		30
APPENDIX C: MSM AND CACHÉ UTILITIES CATALOG.....		2
APPENDIX D: CREATING USER DEFINED COMMANDS, FUNCTIONS, AND SPECIAL VARIABLES		6
APPENDIX E: M LANGUAGE DIFFERENCES		7
APPENDIX F: COMMAND LINE AND BATCH FILE FUNCTIONS		13
APPENDIX G: %ZOS FUNCTION		1
APPENDIX H: OVERVIEW OF SYSTEM ARCHITECTURE		1
APPENDIX I: AUTOMATING CACHÉ BACKUPS		1

1. Executive 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 program
 - Teach staff internally

2. Getting Started

2.1. Determining Hardware Requirements

Before beginning this project, you'll 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.

Please refer to the *Caché Installation Guide* relevant to your operating system for an understanding of how Caché uses memory, and how to calculate the appropriate memory configuration.

At the time of writing, Caché 3.2 is supported on the following platforms:

- Windows Platforms:
 - Windows 95/98
 - Windows NT 4.0 with Service Pack 4 or above (Intel and Alpha)
- UNIX Platforms:
 - Red Hat Linux 6.x
 - Tru64 UNIX 4.0D, 4.0E, 4.0F (Alpha)
 - DG/UX 4.1, 4.2 (Intel and Motorola)
 - HP/UX 10.10, 10.20, 11.0
 - IBM AIX 4.2, 4.3 (PowerPC and RS/6000)
 - SCO Open Server 5.0, 5.0.2, 5.0.4, 5.0.5 (Intel)
 - SCO UNIXWare 7.0 (Intel)
 - Sun Solaris 2.6, 2.7 (Intel and SPARC)
 - Unisys 4.1.2
- VMS Platforms:
 - OpenVMS 6.1, 6.2, 7.0, 7.1, 7.2 (Alpha)

****NOTE:** For a more current list of Cache's supported platforms, please refer to our web site at <http://www.intersys.com> or contact your InterSystems account manager. Also, Appendix A outlines which MSM platforms are supported by Cache

Appendix B makes some hardware recommendations for Windows NT (Intel) platforms that are based on total Caché user counts. While these examples are a good place to start, they may not necessarily be a good example for your needs. For recommendations specific to your requirements, please contact your InterSystems account manager.

****NOTE:** 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'll need to carefully consider how your application will exist in the new hardware configuration.

2.2. Determining Networking Requirements

Caché networking supports the following message format protocols...

- 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

Often, MSM to Caché migrations happen over a period of time. At some stages, you might have some machines running MSM and some running Caché, with Caché-level networking connecting your systems. If you are in this situation, there's some important information you'll need to know before getting started:

- Peer-to-peer networking between MSM and Caché is achieved by using DDP over Raw Ethernet. With this method, MSM can act as either a client or a server to Caché, although it is recommended to use Caché as the database server. This recommendation is based on both database performance and ease of migration for the application.
- MSM does *not* support DCP Networking.
- MSM *cannot* use a Caché database in a Remote Volume Group configuration.

****NOTE:** Caché does not support the Open MUMPS Interconnect (OMI) protocol.

For more information on Caché's supported network protocols, please read chapters one and two in the *Caché Networking Guide*. Chapters two and three of this guide will cover setting up and managing your Caché network configuration.

2.3. 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 the appropriate *Caché Installation Guide* for more details.

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

- System privileges for your machine (either Administrator, root, or system, depending on your operating system)
- Approximately 150 MB of free hard disk space for the core product installation (although half of this is taken by temporary installation files that can be deleted later)
- TCP/IP configured for your OS
- The license key supplied to you by InterSystems to activate Caché's product features (recommended)

****NOTE:** Caché requires that you have the TCP/IP protocol installed on your machine, even for stand-alone configurations. TCP/IP is necessary because all of the GUI utilities connect to the server via TCP/IP. In addition, you may require more disk space if you install any of the add-on products such as Caché ViewPoint or Caché WebLink Developer.

2.3.1. Installing the Caché License Key

Loading and configuring your InterSystems license is a two-step process:

- A. Entering in the license via the Caché License wizard in Configuration manager (installed on the PC client), or via a host-based editor such as vi, emacs, or edit.
- B. Optionally configuring the *license server* for all Departmental, Division, and Enterprise licenses.

On Windows platforms, the installation program will ask you for your license key at the end of the installation. If the UNIX or VMS installation programs do not prompt you for the license, you'll need to enter the license manually using any host-based editor or by using the License wizard in the Configuration Manager from a PC client running Caché. In either case, the information must be entered exactly as shown in your paper copy of the license. **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!**

****NOTE:** The license information is stored in an ASCII file called *cache.key*, found in your manager's directory.

Caché licensing also introduces the concept of a *license server*, which is required for all Departmental, Division, and Enterprise licensing. You **must** configure the license server via the License wizard, or by manually editing the *cache.cpf* file found in your installation directory prior to starting Caché for multi-user use. Failure to configure the license server will result in Caché starting in single-user mode. Basic configuration of the license server is however done for you during installation.

****NOTE:** The license server allows you to centralize license tracking on a single server for an entire Caché network. You can optionally use this feature, or you can start a license server on each individual Caché machine.

For more information on how to install and configure your license, please consult the appropriate *Caché Installation Guide* for your platform.

2.3.2. Running Caché and MSM on the Same System

Your porting circumstances may require that Caché and MSM be installed and running on the same machine. In order for this to be possible, there's some information you should know:

- If you plan on using DSM-DDP networking between Caché and MSM on the same machine, you'll need two separate Ethernet cards.
- If you want both Caché and MSM to run connectivity services simultaneously on Windows NT, you'll need to do the following:
 - For Telnet, you'll need to 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.

Please see chapter four of this guide for more information on Telnet, LAT, and Serial Port services.

2.4. Entering Caché Programmer's Mode

The means of entering Caché programmer's mode varies depending on operating system:

- On Windows platforms, right click on the Cache cube, and select the Terminal option
- On UNIX and VMS platforms, simply enter *cache* at the command prompt.

To learn more about Caché's System Management, please refer to the *Caché Basic* and *Advanced System Management Guides*. Also read *Appendix C* which contains a complete listing of MSM utilities and their Caché equivalents, as well as *Appendix G* which demonstrates other ways of accessing Caché from a command prompt.

3. Creating Caché Databases

3.1. Overview of Database Creation

Creating a database and accessing its data is a two-step process in Caché:

1. Create the physical database volumes via the Configuration Manager wizard.
2. Use the Configuration Manager wizard again to create a new namespace that includes this database as one of its data sources. Or, you can add this database to an existing namespace as a data source.

3.2. Creating Physical Database Volumes

Caché's physical database is made up of a primary database volume, called CACHE.DAT, and up to 7 secondary database volumes, called CACHE.EXT. These files are logically linked together and can total 16 gigabytes. On a given system you can have up to 256 database volumes. One or more logically linked database volumes collectively are called a database, and are referenced by the directory name of the primary volume.

The key points to consider while creating databases are:

- There can only be one database volume of any type (.DAT or .EXT) in a given directory.
- On Windows NT, each volume can be up to 16 gigabytes in size (assuming use of NTFS), but the total logical database cannot exceed 16 gigabytes. On non-Windows NT platforms, each volume has a maximum size constrained by the host operating system, with the same maximum database size also being 16 gigabytes.
- When creating a multi-volume set, create the primary volume first, and then add secondary volumes one at a time.
- Creating as large a database volume 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. A future version of Caché will allow mixed case for database names.

Please read chapter three of the *Caché Basic System Management Guide* for instructions on creating, editing, and deleting databases.

3.3. Creating a Namespace Configuration

Once your databases are defined, you'll add them to your namespace configuration. Through namespace mapping you'll be able to access data from all of your databases from one logical location. Caché namespaces are created via the Configuration Manager namespace wizard.

Some useful facts on namespaces:

- Namespaces allow you to break the 16-gigabyte physical limitation of a single database for your system. And, through subscript-level mapping, a single global is no longer limited to 16 gigabytes.
- 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.
- A future version of Caché will support dynamic namespace mapping, and will use a centralized map file. Currently, mapping is a function of each system in the network, and is stored in a local network configuration file. Until dynamic namespace mapping is supported, activating a new namespace configuration requires stopping and starting the Caché environment.
- You can list all available namespaces by using either ^%CD or LIST^%NSP. Changing namespaces is done by using either the ^%CD utility, ZN “namespace” command, or the \$ZU(5) function.
- You can see the active namespace mappings by running either ^%NSP or ^%GXLINFO.
- As with databases, namespace names currently are translated internally to all uppercase.

****NOTE:** See the section titled *Handling MSM’s Implementation Specific Functions and Features* in chapter four of this guide for tips on how to handle extended global references from your MSM application.

See the *Caché Basic System Management Guide* for more details on creating databases and namespaces.

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 it’s translation table. This translation table also includes locations of % globals, and both % and non-% 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.

THE CACHESYS, CACHELIB AND CACHETEMP DATABASES CAN BE EITHER PURGED OF GLOBALS AND/OR ROUTINES DURING A CACHE’ UPGRADE OR EVEN COMPLETELY REPLACED BY A NEW VERSION.

Thus, it can be considered dangerous to simply load any user written % routines into 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.

4. Converting MSM Globals

Caché provides several mechanisms for converting your MSM data. If you plan on converting 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 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.

****NOTE:** Be sure to read the sections called *Preserving MSM String Collation* and *Enabling NULL Subscript Support in Caché* described later in this chapter **before** converting your data.

4.1. A Suggested Strategy for Converting Globals by Database

1. Backup your MSM databases.
2. Run ^VALIDATE on your MSM database to ensure physical integrity.
3. Run ^OLC on your MSM database to maximize global efficiency. This will help the conversion's overall performance.
4. Kill any MSM system or user-defined scratch globals.
5. Ensure that the database to be converted is not active, either by shutting down MSM or by dismounting the database.
6. 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.
7. Configure as many global buffers as possible on your Caché system, and then start Caché.
8. Make sure your local destination database(s) and namespace are ready (see above section).
 - Change to this namespace with the ^%CD utility.
9. From this namespace, run the ^%MSMCSV utility. At the prompt, enter the path and filename of your MSM database.
 - If you are importing data from several MSM databases into one Caché namespace, run the utility again, once for each database, from the same namespace and enter the new UCI to be converted.

****NOTE:** The ^%MSMCSV utility is endian-inspecific. This allows you to convert database files from one machine's byte order to another.

4.2. Converting Individual or Groups of Globals

Caché also offers several methods for converting selected globals, rather than an entire database. Some examples are:

- Set up a network between MSM and Caché (refer to the *Caché Networking Guide*). 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 either the Caché Explorer or ^%GI to import the data. It is more reliable to use the ANSI^%GS entry point.
 - For %FGS-format global saves, use ^%GIFMSM to import the data.

****NOTE:** The ^%GS approach to migrating specific globals only works for globals which do not contain control characters in either the subscripts or data. For these special globals, you must either convert them as part of a complete database with ^%MSMCVT, use ^%FGS, or use MERGE to move them over a network.

****NOTE:** As with the ^%MSMCVT conversion approach, you'll want to run two performance-enhancing steps prior to executing the global conversion. First, run ^VALIDATE and ^OLC on your MSM database to maximize global efficiency. Then, configure as many global buffers as possible on your Caché machine.

4.3. 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, in order to convert MSM globals that use a string collation, you'll need to prepare Caché.

For example, let's say 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 ^%GCREATE utility
 - A. Change to the appropriate namespace.
 - B. Invoke ^%GCREATE, and select type129 (string collation) when prompted for the globals collation sequence.
 - C. Run the appropriate conversion mechanism described earlier in this section.
2. Change the characteristics of the process performing the conversion to use string collation
 - A. Issue the command: `set x=$ZU(23,1,129)`
 - B. Run the appropriate conversion mechanism described earlier in this section.

****NOTE:** Option 1 is recommended unless you want to convert all MSM globals with a string collation type.

4.4. Enabling NULL Subscript Support in Caché

By default, Caché does not allow the use of NULL subscripts in your global data. If your globals do contain NULL subscripts, you'll need to prepare Caché *before* running your conversion.

There are two ways you can convert MSM globals that contain NULL subscripts:

1. Enable NULL subscript support interactively:
 - A. For the issuing process *only*
 - Run the command `set x=$zu(68,1,1)` from a Caché Programmer's Prompt
 - B. For *all* processes in the system
 - Run the command `set x=$zu(69,1,1)` from a Caché Programmer's Prompt
 - C. Run the appropriate conversion mechanism described earlier in this section.
2. Enable NULL subscript support at each Caché startup for *all* processes:
 - A. Choose one of the two following methods:
 - Enable NULL subscripts from the Advanced Language tab of Configuration Manager
 - Add the following line of code to either `^ZSTU` or `SYSTEM^%ZSTART`:

```
set x=$zu(69,1,1).
```
 - B. Restart Caché
 - C. Run the appropriate conversion mechanism described earlier in this section.

****NOTE:** It is recommended that you modify your application and data so that NULL subscripts are not used.

5. Converting Your MSM Application

Despite the fact that Caché provides an MSM language compatibility mode, while you can often run your MSM application with few changes using it, it is *strongly* recommended that you convert any MSM-specific language features to native Caché code. This will be the first step in taking advantage of Caché-specific features such as ObjectScript, as well as \$LIST and \$INCREMENT functions.

5.1. Migrating Routines

Migrating the routines over to Caché is just one of the steps necessary to convert the application. After following the migration steps, please look to the next chapter for more information on the application conversion process.

****NOTE:** If you have implemented your application using M/SQL for MSM, the migration steps provided in the following section require that you are running F.12 (or later F release) of the Caché DBMS.

5.1.1. Porting MSM Routines Less Than 32 KB in size

1. Save your MSM routines using one, or both, of the following steps:
 - A. For users of M/Sql for MSM, save any .MAC or .INC code (*.MAC and *.INC) with the ^%urprint utility.
 - B. Save any standard MSM source code with the ^%RS utility. Save these routines in MSM format.
2. Restore your code into Caché using one of the following steps:
 - A. For users of M/Sql for MSM, load your ^%urprint output into Caché using ^%urload from your destination namespace.
 - B. Load your ^%RS output with either the Caché Explorer or ^%RI in the destination namespace.

****NOTE:** With both restore options, be sure to disable syntax checking and compiling options before loading in your routines.

- C. Recompile your code in Caché using ^%RCOMPIL from the destination namespace to compile these routines and check for syntax errors.
 - Once you find which routines have generated <SYNTAX> errors, you should look to the *Caché ObjectScript Language Reference* to familiarize yourself with Caché's language syntax.

Please refer to the section titled *MSM Language Compatibility Mode* in chapter four of this document for more information on MSM language mode.

****NOTE:** 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'll need to contact your application supplier to get the routine's source code.

****NOTE:** It is recommended that you do *not* migrate any M/SQL-generated routines, such as ^mt*, ^mq*, or ^mw*, from MSM to Caché since these will be re-generated upon converting your M/SQL objects.

If you are converting an M/SQL application built on top of MSM, you should refer to the section called *Relational Tools Conversion* found in chapter four of this guide for more information.

5.1.2. Porting MSM Routines Greater Than 32 KB

While MSM can support as large a routine as the current stack and partitions sizes will allow (tested as high as 200 KB), Caché supports routines up to 32 KB in size. Any MSM routines larger than 32 KB can be addressed in 1 of 2 ways:

1. Split the routines in MSM first so that no routine is larger than 32 KB, then import into Caché.
2. 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.

****NOTE:** Option 2 is *not* recommended. Since Caché Studio currently supports routines up to 32 KB large, you'll need to restructure the routines either through querying the `^ROUTINE` global and using `ZINSERT` and `ZSAVE` commands, or by directly modifying the `^ROUTINE` global. A future versions of Caché *will* support a larger size for routine source code.

****NOTE:** You can use MSM's `^%RSIZE` utility to check your routines' 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.

5.2. MSM Language Compatibility Mode

You use `$ZU(55, num)` to switch language modes, where *num* is equal to the language mode.

- Issue the command `s x=$ZU(55, 8)` from either programmer's mode or from within a routine to switch to MSM mode
- To return to native Caché mode, use the command `s x=$ZU(55, 0)`.
- Use the command `s x=$ZU(55)` to return the current language mode.

****NOTE:** Native Caché mode is the default for each process. To change this default for specific process types, you'll 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. Depending on the value of *num*, corresponding Caché routines will be searched for these added features. When in native Caché mode, the `^%ZLANGC00`, `^%ZLANGF00`, and `^%ZLANGV00` routines are used. In MSM mode, the routines `^%ZLANGC008`, `^%ZLANGF008`, and `^%ZLANGV008` are used.

Notes for the `^%ZLANGC00*`, `^%ZLANGF00*`, and `^%ZLANGV00*` routines:

- 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 entered in all capital letters. However, actual execution of the command, function, or special variable is case-insensitive.
- Adding code to `^%ZLANGC00*` creates a new command, such as `ZSS`.
- Adding code to `^%ZLANGF00*` creates a new function, such as `$ZOS()`.
- Adding code to `^%ZLANGV00*` creates a new special variable, such as `$ZTIME`.
- Follow the lead of the code that already exists in these routines.

****NOTE:** Through this facility, you'll be able to convert MSM's `%ZZCMNDS`, `%ZZFUNCS`, and `%ZZSVARS` routines.

To date, there is no formal documentation in the Caché documentation on `^%ZLANGC00*`, `^%ZLANGF00*`, or `^%ZLANGV00*`. See either *Appendix D* or the `^%ZLANG*` routines themselves for some code examples. However, information on basic features of language compatibility mode is available in Appendix B of the *Caché Programming Guide*.

5.3. Handling MSM's Implementation Specific Functions and Features

Converting MSM's non-standard 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 rewritten for Caché.
- Any `VIEW` commands or `$VIEW` functions which access MSM memory structures will need to be removed or modified.

- MSM error trapping used by the application will need to be changed to use Caché's error trapping functions. See chapter seven of the *Caché Programming Guide* for more information on Caché error handling.
- Any references to MSM utilities will 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.

****NOTE:** See *Appendix E* for a list of MSM commands, functions, operators, preprocessor directives, special variables, and structured system variables, and their Caché equivalents.

****NOTE:** 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. It is recommended that code changes be done interactively by a programmer.

Appendix B of the *Caché Programming Guide* outlines some syntactic and formatting issues when migrating routines from MSM.

5.3.1. Generalized approach to get you started

The following is a list of suggested actions that you can take to get a rough idea of how much effort will be involved in an MSM to Caché conversion. It is NOT meant to be a definitive list.....

- 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 also 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 semi-colon.
 - Duplicate line tag names in routines are *not* supported in Caché. Compiling such a routine will give you a <LABELREDEF> error. You'll need to eliminate or change one of the line tags, and make any necessary code changes to maintain the same application logic. InterSystems supplies a routine called ^%LBLRDEF which will rename any duplicate line tags for you, although this should of course be used with care.
- Using %RFIND or %RSE, search for the following character strings:
 - \$V
 - \$Z
 - ^%
 - ^[
 - ^|
 - ^ ((note space between the '^' and the '(')
 - O 51, O 52, O 53 or O54
 - TCP

5.3.2. 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() Call-Out Interface.

- XCALL and ZCALL functions must be converted to Caché \$ZF() functions. While there is no automated process for this, the code samples represented in *czf.c* (found in the *bin* subdirectory of *cachesys*) exemplify how to write external functions for Caché.

****NOTE:** Caché also supports a Call-In Interface, allowing you to execute Caché code from within a C program. See *MSM-Activate* found under *Connection the Application to Caché* later in this chapter for more information.

For more information on Caché's Call-Out Interface, please chapter twenty-one of the *Caché Programming Guide*.

5.3.3. 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.

Appendices E and F in the back of this document exemplify how to perform various DOS functions from Caché. You are free to use these code samples, but please remember that this code is meant to serve as an example only, and is unsupported and without warranty.

****NOTE:** Caché's %CLI utility allows you to run OS commands from a Caché programmer's prompt. On Windows platforms, these commands will hang if they expect user input. See chapter twenty-one of the *Caché Programming Guide* for more details.

5.3.4. 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.

- You can use mnemonic spaces to create pop-up windows. Chapter sixteen of the *Caché Programming Guide* goes into extensive detail on how to use the windowing API - %CHARWIN
- Use the \$ZF(-1) function to issue an operating system level command to open a new terminal window. For a list of examples, please refer to *Appendix G*.
- Another interim option is to 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.

5.3.5. 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.

****NOTE:** Caché stores its directives in .MAC and .INC code. See chapter three of the *Caché Programming Guide* for more information on Caché routine types.

****NOTE:** 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'll 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`

****NOTE:** Arguments of the formal parameter list in Caché *must* begin with a % sign.

****NOTE:** 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, you'd call the copyright macro using this syntax:

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

****NOTE:** The %% and #% syntax for referencing macros are *not* supported in Caché.

- For more information on Caché's macro preprocessor directives, please read chapter five of the *Caché Development Guide*

5.3.6. 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. For these cases, you'll need to know the following:

- Caché supports several forms of extended references:
 - For globals:
 - `^["namespace"]global`
 - `^["namespace", ""]global`
 - `^["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"]`

****NOTE:** You might want to create a Caché namespace called MGR that uses the CACHESYS database as the default location for globals and routines. This way, extended references that expect MGR to contain system globals and routines will not need modification.

5.3.7. 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'll need to create a custom error trap in your application using \$ZTRAP.

****NOTE:** Caché 3.x also supports the \$ZC (if used in MSM language compatibility mode) as well as the \$ZEOF (native Caché language mode) special variables.

5.4. Connecting the Application to Caché

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

5.4.1. 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
- A COM object
 - This is in fact the same item as the ActiveX control, but has no visual representation on the client system.
- 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.

****NOTE:** MSM-Activate's server code can be ported to Caché, enabling you to run your MSM-Activate applications against Caché servers without any client-side application changes. For more information, please contact your InterSystems Sales Representative.

InterSystems' recommended long-term goal is to convert your MSM-Activate based applications to native Caché technology. Depending on the tool you've used to build your client interface, you'll have different Caché options available.

Caché Objects allow you to use the following tools:

- Any Active X client, such as Visual Basic, PowerBuilder, or Delphi
- Any Java-based tool such as J++, Visual Café, or JBuilder
- C++

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, such as:

- Visual Basic
- PowerBuilder
- Delphi

****NOTE:** The Caché Direct control *requires* the use of some container, such as a PowerBuilder visual object or a Visual Basic form. For this reason, you *cannot* use the Caché Direct control with PowerBuilder's non-visual objects. A future version of Caché Direct *will*, however, allow Caché Direct to be scripted as an Active X automation server.

****NOTE:** The Caché Direct control is called VisM.OCX.

Please see the *Caché Direct Release Information* for more information on Caché Direct's features and use.

Caché's Call-In Interface is very similar to MSM-Activate's Call-In Interface, and allows you to access Caché from C programs on the following platforms:

- All Caché-supported Windows platforms
- All Caché-supported UNIX platforms
- All Caché-supported VMS platforms

****NOTE:** The *ccallin.h* file, found in the *bin* subdirectory of your Caché installation directory, provides the necessary function definitions for Caché's Call-In Interface, and should be included in your C program. The *ccallin.c* file, also found in the *bin* subdirectory, contains a sample C program that demonstrates the major call-in functions defined in *ccallin.h*.

****NOTE:** While MSM can be called as a Windows DLL from a C program, Caché *cannot*. A future version of Caché *may* support this. Please contact your InterSystems Sales Representative for more information.

Please see chapter twenty-two of the *Caché Programming Guide* for more information on the Caché Call-In Interface.

5.4.2. MSM-SQL, KB-SQL, and M/SQL

For relational access, you could have chosen one of three relational environments for MSM:

- MSM-SQL
- KB-SQL
- InterSystems' M/SQL

Since the globals containing your data dictionaries will have been already migrated from MSM to Caché, you'll be able to convert the relational environment from MSM's DBMS to either CacheSQL or version F.15 (or later F version) of M/Sql for Caché directly on your Caché system.

1. For the MSM-SQL and/or 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 CachéObjects class definitions. These definitions automatically expose your data in terms of relational tables.
2. For the M/SQL DBMS, you'll need to do two steps to convert to Caché's DBMS. These steps *must* be run from any namespace that contains DBMS objects.
 - A. Use the Conversion Manager, found under the System Management menu option of the ^%msql utility, to run any necessary DBMS conversion steps.
 - B. Recompile all DBMS objects by running the command `d all^%mcompil`.

For information on converting your MSM-SQL and KB-SQL data dictionaries, please contact your InterSystems Sales Representative. For details on converting from M/SQL to F.15 or greater on Caché, please refer to the *F to CDL Export utility Guide*.

****NOTE:** Once your data dictionaries have been converted to F.15 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'll benefit from much-improved overall relational performance. See the *Caché ODBC Driver Guide* for more information on Caché's ODBC driver.

5.4.3. 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.

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, including:
 - Microsoft IIS 1.0+ (NT)
 - Microsoft Personal Web Server 1.0+ (95/98)

- Microsoft Peer Web Server 1.0+ (NT Workstation)
- Netscape 2, 3 (NT Server, NT Workstation, 95/98)
- Netscape 2, 3 (HP, PowerPC, RS/6000, SGI, Solaris/Sparc, Digital Unix)
- Netscape Fastrack 2.0+ (95/98)
- SCO FastStart 2, 3
- Apache

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. Please consult the *Caché WebLink Guide* AND detailed information on this conversion process.

5.4.4. 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'll 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

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

For more information, please contact your InterSystems Sales Representative.

5.4.5. 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. TELNET and LAT connectivity is configured through the Configuration Manager.

Under Windows NT 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:

1. Tie individual or groups of terminals to a particular \$ZIO value.
 - Identify a specific terminal by creating a Caché user account (via the User Accounts utility) whose name is made up of the terminal's corresponding server or domain name and port.
 - This terminal information is available to your application in 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.
2. 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).
 3. 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.

Please refer to the *Caché Advanced System Manager's Guide* for more information on setting up tied terminals.

****NOTE:** Caché's TELNET and LAT services do *not* run on Windows 95 or 98 platforms. For this functionality, you must use a Windows NT platform.

Note that 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.

5.4.6. Serial Port Expander Boards

Caché for Windows NT 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 take two steps:

1. Make sure the Auto-Start COM Ports box is checked. This option is found on the Terminal tab of the System Configuration Utilities. You'll need to restart Caché after enabling this option. This launches COM port server routine that will listen on ports defined as usernames.
2. Enable the desired COM port application. This is accomplished through the Security tab in Cache Control Panel.

5.4.7. Calling into a Caché Routine from the Command Prompt or from a Script File

You can execute a Caché routine from the command prompt or from script file by following the examples in *Appendix F*

6. Caché System Management

6.1. Configuring Caché

Configuring Caché is done via the Configuration Manager. These utilities manage a file called, by default, *cache.cpf*. The *Caché Advanced System Manager's Guide* includes detailed information on how to use the utilities, 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:

- Make sure Multiprocessor is checked if you have an SMP machine.
- If you have a Workgroup, Departmental, Division, or Enterprise license, you must run a License Server. You should contact your InterSystems account manager for license verification.
- 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. Over-setting this value will use memory and swap space inefficiently. Note that you cannot dynamically change the partition size as there is no %PARTSIZ equivalent utility in Caché for the moment. Note also that Caché partitions are fixed size and do not expand and contract as do MSM partitions.
- In many cases, increasing the number of Global and Routine Buffers is the best and quickest way to improve performance.

Appendix C contains a complete listing of MSM utilities and their Caché equivalents. Also, *Appendix H* outlines an MSM to Caché comparison of maximum values for miscellaneous system specifications.

6.2. Networking Between Caché and Other Systems

As mentioned earlier, Caché networking supports the following message format protocols...

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

...running on top of one of these communication protocols:

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

****NOTE:** For MSM to Caché networking, you *must* use DDP over Raw Ethernet.

You'll use Configuration Manager utilities to set up your network configuration. There are also tunable startup parameters for networking under the Advanced|Network tab in Configuration Manager, including the enabling of Remote Jobs and Network Hardening.

Before setting up your network, there's some basic information you should know:

- You *cannot* share routines across a network between MSM and Caché.
- You *cannot* share string-collated globals over a DSM-DDP network.
- You have control over the percentage of global buffers that are allocated for networking. By default, this value is 25 percent.
- You can specify a timeout value that represents how long a process will wait to make a network connection before generating a <NETWORK> error. This value defaults to 30 seconds.
- Network hardening, which is enabled by default, allows jobs to pend rather than receive an error in the event of a network outage.

Chapter four of the *Caché Networking Guide* includes important information on MSM to Caché networking via the DSM-DDP message format protocol.

6.2.1. 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. So, if your application sets globals indexed by \$JOB from multiple clients to the same server, you'll need to make a slight modification.

If this situation applies to you, consider 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 (probably a good idea anyway!!)

- Use subscript-level global mapping to distribute the global differently.
- Change the way in which the global is indexed, if possible.

Please refer to the *Caché Networking Guide* for more information on networking setup and features.

6.3. 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, define them in Configuration Manager. 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 both the ^SYS("MTDEV") global, from where magnetic tape devices are accessed, and the ^%IS global, from where all other devices are accessed.

Tips on configuring devices:

- 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.
 - Mnemonic names are used by the character-based utility called ^%IS. ^%IS 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.
- You can recreate the ^SYS("MTDEV") and ^%IS globals on a running system by choosing the Reload Device Table option in the GUI System Utilities.
- Pipes are a very effective way of accessing printers and other devices on your machine. Chapter twenty of the *Caché Programming Guide* reviews 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.

****NOTE:** 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.

Please refer to the *Caché Advanced System Manager's Guide* for information on configuring devices in Caché.

6.4. Backing Up and Restoring Your Data

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. Please refer to the *Caché Advanced System Manager's Guide* for more information on the basic backup types, and how to perform them

****NOTE:** Restoration of system backups is performed via the character-based utility called ^BACKUP. Use options 2 or 3 from the menu to restore your databases.

6.4.1. Automating Caché Backups

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:
 - A. As a pre-backup command, clear incremental bitmaps:
 - `s x=$$CLRINC^DBACK(1)`
 - B. Run the OS backup on the *live* system
 - C. As a post-backup command, perform cumulative backup:
 - `s x=$$BACKUP^DBACK(Arg1,Arg2,...Arg10)`

****NOTE:** The fourth backup option requires that your OS-level backup allow files to change while the backup is being performed. Please choose your OS backup software with care.

Appendix I includes sample Caché code that demonstrates the first and third strategies from above.

6.5. 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.

The Before Image Journaling (BIJ) equivalent in Caché is 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) called *cache.wij* that lives in the \cachesys\mgr directory. Its size is set equal to the size of the global buffer pool (set in Configuration manager).

Likewise, After Image Journaling is also automatically enabled. Unlike MSM though, it is not necessary to define journal files in advance, rather they are created as needed. Each time Caché is restarted a new journal file is begun. System settings determine at what 'age' a journal file is automatically deleted, and can be modified in Configuration Manager.

Most journaling tasks are accessible via the Journaling tab in the Control Panel. This includes basic tasks such as starting and stopping journaling, as well as managing the journal shadow clients and servers. The name of the journal file is in current date format (yyyymmdd.num) and, by default, exists in the *journal* subdirectory of the system manager's directory. You may add a prefix to the name of your journal file, but no other editing of the journal file name is allowed.

Journaling Tips:

- A process can enable or disable journaling for itself via the ENABLE and DISABLE line tags of ^%NOJRN, respectively.
- Do not use the *Journal All Globals* option unless you 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.
- Shadow System Journaling can be used for data redundancy.
 - 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.

****NOTE:** In addition to shadow system journaling, data redundancy can easily be accomplished through a variety of other mechanisms, including hardware or software driven mirroring, or Caché's global replication.

****NOTE:** Caché does *not* support transaction restarts, as MSM does through TRESTART.

6.5.1. Shadow System Journaling

While Caché's Shadow System Journaling is very similar conceptually to MSM's Cross-System Journaling, you'll 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. Block-mode Transmission

- A. The shadow connects to the database server via TCP and captures the live journal flat file.
- B. 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.
- C. Since many transactions are captured at once via a binary journal block, block-mode tends to be quicker than record-mode.
- D. Applied transactions are optionally logged in the shadow machine's local journal file.

2. Record-mode Transmission

- A. The shadow connects to the database server via TCP and captures transactions via packaged strings.
- B. 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
- C. Since transactions are captured via packaged strings, record-mode tends to be slower than block-mode.
- D. Applied transactions are optionally logged in the shadow machine's local journal file.

****NOTE:** Since Caché uses native TCP as its transfer protocol, rather than DDP as MSM uses, you'll benefit from much improved shadowing performance.

****NOTE:** Currently, Caché does not support automatic failover as a feature of Shadow System Journaling. Failover can be accomplished manually by following the steps outlined in the *Caché Advanced System Manager's Guide*.

****NOTE:** Caché *can* be part of a UNIX High Availability Cluster, such as HACMP or MC/ServiceGuard. Please contact your InterSystems Sales Representative for more information.

Note also that 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. Thus WAN support is easier to setup. In addition multiple shadow servers can be defined for each primary server.

APPENDIX A: MSM and Caché: Supported Platforms

MSM Platform	Caché Available?	Notes
DOS	No	
Windows 95	Yes	Windows 98 also
Windows NT 4.0 – Intel	Yes	
Windows NT 4.0 – Alpha	Yes	
DEC Alpha / Digital UNIX 4.0	Yes	
DEC ULTRIX 4.5	No	
DG AViiON (Intel) – DG/UX 4.20	Yes	
DG AViiON (Motorola) – DG/UX 4.11	Yes	
HP – HP/UX 10.20	Yes	
IBM RS/6000 – AIX 4.2	Yes	
ICL DRS6000 – R4.2 V7.22	No	
Motorola 88110 – R40 V4.4	Yes	
NCR 3000 – ATT 5.4.2	No	
SCO UNIX – Open Server 5	Yes	
Sequent S series – DYNIX/ptx R3.2.0 V2.1	No	
Siemens RM400/600 – SINIX 5.43B or Reliant UNIX 5.43B	No	
Sun SPARC – Solaris 2.6	Yes	

APPENDIX B: Hardware Recommendations

****NOTE:** The following recommendations supply *minimum* figures, and assume that Caché is the only running application on the machine. Also, depending on your operating system's resource requirements, these numbers might need to be adjusted.

There is NO substitute for benchmarking your own application!

50 Caché Users

Pentium Pro 200 MHz
64 MB physical memory
64 MB swap space

400 Caché Users

Quad Pentium Pro 200 MHz
512 MB physical memory
512 MB swap space

100 Caché Users

Dual Pentium Pro 200 MHz
128 MB physical memory
128 MB swap space

750 Caché Users

Quad Pentium Pro 200 MHz
784 MB physical memory
784 MB swap space

200 Caché Users

Dual Pentium Pro 200 MHz
256 MB physical memory
256 MB swap space

1000 Caché Users

Quad Pentium Pro 200 MHz
1 GB physical memory
1 GB swap space

The 50-user memory recommendation was based on the following data:

- Windows NT 16,777,216 bytes
- 25 Global Buffers per user = $50 * 25 * 2.2K * 1024$ = 2,252,800 bytes
- 1000 Routine Buffers = $1000 * 24K * 1024$ = 24,576,000 bytes
- 256K partition per user = $50 * 256K * 1024$ = 13,107,200 bytes

Total Memory Used: 56,713,216 bytes

****NOTE:** The number of global and routine buffers required is very application-specific. Routine Buffer count should be based on the total number of core application routines that users frequently hit while working. Also, if your application requires a partition size larger than 256K, you will need to recalculate your total memory requirements.

Caché ViewPoint will be your greatest tool for monitoring system performance and requirements.

APPENDIX C: MSM and Caché Utilities Catalog

****NOTE:** See the documentation for each function's location within the GUI Utilities, and for instructions on how to use it.

<u>MSM</u>	<u>Caché</u>	<u>Description / Notes</u>
%ACTJOB	No Direct Equivalent	Provides a ^-delimited list of all job numbers in the system. In Caché, use the following code: s (j,p)=" " f s j=\$o(^\$j(j)) q:j=" " s p=p_j_ "^" <i>or</i> s (j,p)=" " f s j=\$zj(j) q:j=" " s p=p_j_ "^" **NOTE: Use of ^\$JOB is recommended over \$ZJOB.
%CHKSUM	Not Available	Computes a checksum (ASCII summation) of one or more routines.
%D	%D	Displays the date currently stored in \$HOROLOG. %D in MSM reports the date in the format DD-MMM-YY, while Caché uses the format MMM-DD-YY. Use \$zd(\$h,2) to mimic MSM's %D output in Caché.
%DEBUG	No Direct Equivalent	Invokes an interactive program debugging facility. See the <i>Caché ObjectScript Reference</i> for information on BREAK and ZBREAK.
%DEVUSE	No Direct Equivalent	Displays a list of all opened devices and the number of the job that owns each one. In Caché, use TTYFREE to check reserved TTY devices, and the processes that own them. Also use %SS to see all processes and the devices they have open.
%DH	%DX	Converts a decimal value to hexadecimal.
%DI	%DATE	Converts a date from external form (for example: 8-SEP-97) to internal \$HOROLOG format. In Caché, you can call %DATE programmatically via the INT line tag.
%DO	\$ZDATE(\$H_Value)	Converts a date from internal \$HOROLOG format to external format.
%ECHO	No Direct Equivalent	Allows the program to control the echoing of characters at the terminal. Entry points are provided to turn ECHO on and off. 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: u 0:(:"s") r rec See chapter fifteen of the <i>Caché Programming Guide</i> for more details.
%EDP	Not Available	Performs macro lookup, expansion, and parameter substitution.
%EDP1	Not Available	Processes directives, has supplementary entry points.
%ER	%ER	Displays error information trapped by the %ET routine.
%ERRCODE	Not Available	Display an explanation for database-specific error codes.
%ET	%ET, %ETN	Error trap routine supplied with MSM and Caché. Caché's %ET(%ETN) routine is much more feature-rich.
%FGR	%GIF, %GIFMSM	Fast global restore (block format). Use %GIFMSM to import %FGS-format global saves into Caché.
%FGS	%GOF	Fast global save (block format)

%FL	GUI, %RFIRST	Display the first line of code for selected routines.
%FLIST	Not Available	Lists a file stored in the host file system.
%GCH	GUI, %GDISP,PROTECT	Display characteristics of global(s) and modify
%GCHANGE	%GCHANGE	Changes all occurrences of a string in one or more globals.
%GCMP	%GCMP	Compares two globals in the same or different namespace.
%GCOPY	%GCOPY, MERGE	Copies one or more globals from one namespace to another. The namespace may be on the same machine or on a remote machine.
%GD	GUI, %GD	Display global directory for current namespace.
%GDE	GUI, %GDISP	Provides an extended global directory display.
%GDEL	Not Available	Deletes one or more globals from a namespace.
%GE	INTEGRIT, BLKDIST	Display efficiency of global(s).
%GEDIT	GUI Global Viewer	Allows you to edit all or selected portions of a global file.
%GL	GUI, %G	Lists all or selected portions of a global file.
%GR	GUI, %GI, %GIGEN, %GIF	Restore global(s) from a device, and allows them to be renamed. Caché does <i>not</i> allow the renaming of globals on import.
%GS	GUI, %GO, %GOGEN, %GOF	Saves all or selected portions of one or more globals to a device.
%GSE	GUI Global Search	Searches one or more globals for one or more user-specified strings.
%GSEL	%GSET	Allows you to select one or more globals from the current namespace.
%GSIZE	%GSIZE	Display size of one or more globals
%GUCI	%DIR	Returns the three-character name and internal UCI number for the current UCI. Caché's %DIR will report the current namespace and default global directory.
%HD	%XD	Converts a hexadecimal number to a decimal.
%HELP	No Direct Equivalent	Provides online help for character-based utilities. In Caché, you can get online help by entering a '?' at any of the character-based utility prompts.
%HL	%PRIO	Allows you to change the priority of the current job from high to low or from low to high. Call Caché's %PRIO utility through the LOW, NORMAL, and HIGH line tags.
%HOSTCMD	^%CLI, \$ZF(-1,"CMD")	Allows you to issue host operating system commands from within an M program.
%INDEX	Not Available	Provides a cross-reference listing of one or more routines, and optionally provides a structured program listing of selected routines.
%LOGON	^%CD, ZN, \$ZU(5)	Allows you to switch from one UCI to another.
%MDMP	Not Available	Provides a display in hexadecimal format, character format, or both for selected memory locations or the VIEW buffer.
%MFUNC	^%math	Provides mathematical functions including E, PI, SIN, and COS. ^%math must be called by the appropriate line tag—see source code.
%MODESET	Not Available	Allows you to change environmental mode flags such as maximum length of routine lines.
%MTCHK	Not Available	Allows you to interrogate the status of a magnetic tape drive.
%NEWED	%RD	Lists routines that were filed by the program editor during a specified range of dates.
%OS	No Direct Equivalent	Performs operating system-specific tasks. See <i>Appendices G & H</i> for more

		details.
%PARTSIZ	Not Available	Allows you to dynamically change the partition size of the current job.
%RCHANGE	%RCHANGE	Changes all occurrences of a string in one or more routines.
%RCMP	GUI, %RCMP	Compares two routines in either the current namespace or different namespaces.
%RCOPY	GUI, %RCOPY	Copies a routine from one UCI to another. Caché's %RCOPY will <i>not</i> allow you to copy the routine to another namespace. This utility renames a routine in the current namespace.
%RD	%RD	Display a routine directory for the current namespace.
%RDEL	%RDELETE	Deletes one or more routines from the current namespace.
%RELOAD	%RCOMPIL	Re-compiles one or more routines in a namespace.
%RPRT	GUI, %RD	Prints a listing of one or more routines stored in the current namespace.
%RR	GUI, %RI, %RIMF, %urload	Restores all or selected routines from an external device and allows them to be renamed. Caché does <i>not</i> allow the renaming of routines on import.
%RS	GUI, %RO, %ROMF, %urprint	Allows one or more routines to be saved on an external device.
%RSAND	GUI, %RFIND	Searches one or more routines for occurrences of one or more character strings. 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	GUI, %RFIND	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	%RSET	Allows you to select one or more routines from the current namespace.
%RSIZE	GUI, %RD, \$\$^%ROUOBJ(...)	Displays the number of blocks used by selected routines.
%SBP	Not Available	Displays the current status, block location, and buffer offsets for the Sequential Block Processor device.
%SDEV	%IS	Allows you to select and open a device, and specify the OPEN parameters.
%SI	GUI, %SS, MKEY	Displays general system information, including the status of system-related processes.
%SP	GUI, %FREECNT	Displays the total amount of disk space within a volume group and the amount of free space.
%SQRT	sqr^%math	Computes the approximate square root value of a number.
%SS	GUI, %SS \$V(-1,PID)	Displays status information about each job currently active on the system.
%T	%T	Displays the time stored in \$HOROLOG in the form HH:MM. In Caché, use the INT tag to programmatically get the time.
%TI	%TI	Converts a time value in external format (for example: 1:05 P.M.) to an internal \$HOROLOG format. In Caché, use the INT tag to programmatically get the \$HOROLOG value.
%TO	\$ZTIME	Converts a time value from internal \$HOROLOG format to external format.
%TRANS	Not Available	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	GUI, ^UTIL	Provides a way to invoke most MSM utilities, based on the type of function to be performed.
%VIDEO	Not Available	Allows users to modify contents of the PC Console video buffer. (MSM-PC/PLUS and MSM for Windows only).
%XMIT	Not Available	Enables communication with another port on the system; this is useful for transferring information between machines.
%ZSTIME	%RD	Displays the last-saved time of one or more routines.
APIMGR	No Direct Equivalent	MSM-Activate management utilities In Caché, use the Caché Direct Client and Server Management Utilities.
BCS	GUI, BROADCAST, \$ZU(9), \$ZU(94)	Broadcast messages to other terminals or process IDs
BIJ	GUI, cwdimj	Manage Before Image Journaling (Write Image Journaling)
DBMAINT	GUI, MSU, MOUNT, DISMOUNT, etc.	Perform database maintenance
GLBPLACE	%GCREATE	Create a new global
JOBEXAM	GUI, JOBEXAM	Display detailed information for a process
JRNL	GUI, JRNSTART, JRNSTOP, JRNDUMP, JRNSWTCH, JRNRESTO, %NOJRN	Manage After Image Journaling (Flat File Journaling)
KILLJOB	GUI, RESJOB, \$ZU(4)	Terminate a job
LOCKTAB	GUI, LOCKTAB	Display all active locks in the system
OLB	GUI, BACKUP	Perform concurrent backup of databases
OLC	GUI, GCOMPACT	Perform online compression
PEEK	Not Available	Monitors another terminal device's activity.
RECOVLCK	GUI, LOCKTAB	Recover a lock
SETBAUD	Not Available	Temporarily modifies terminal characteristics such as number of data bits, number of stop bits, parity, and baud rate of a terminal. On Caché for UNIX, you can try: s x=\$zf(-1,"stty ispeed 9600") and/or s x=\$zf(-1,"stty ospeed 9600")
SSD	SHUTDOWN, ZSHUTDOWN, %ZSTOP	System shutdown utility
STU	STU, ZSTU, %ZSTART	System startup utility
SYSGEN	GUI	Generate system configuration
UCIMGR	GUI	Manage UCI configurations
VALIDATE	GUI, INTEGRIT, CHECKPNT, CHECKMAP	Check physical integrity of a database
VERIFY	See VALIDATE	Verify a database's physical integrity
XCALLMGR	Not Available	Manage XCALL functions

APPENDIX D: Creating User Defined Commands, Functions, and Special Variables

****NOTE:** The code presented here is unsupported and without warranty.

Setting up the %ZLANG* Routines

```
%ZLANGC008 ; MSM command routines
```

```
;  
ZSS ; do a System Status  
d ^%SS  
q
```

```
%ZLANGF008 ; MSM function routines
```

```
;  
ZRTN(x) ; find out what routine a process is currently running  
q $p($v(-1,x), "^", 6)
```

```
%ZLANGV008 ; MSM special variable routines
```

```
;  
ZTIME ; return the current time  
d INT^%T  
q %TIM
```

Example of Calling the Custom Code

```
myrtn ; Call the code added to the ^%msql* routines
```

```
;  
n pid,rtn,x  
s x=$ZU(55,8) ; Change to MSM mode  
ZSS  
r !,"Enter a process ID: ",pid  
s rtn=$ZRTN(pid)  
i rtn="" s rtn="no routine"  
w !,"At "_$ZTIME_", process #"_pid_" was running "_rtn_"."  
s x=$ZU(55,0) ; Return to native mode  
q
```

APPENDIX E: M Language Differences

****NOTE:** Only language features that differ between MSM and Native Caché mode are listed. **Please read chapter twenty-four of the *Caché Programming Guide* to learn how MSM language compatibility mode will affect the behavior of these language features.**

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

****NOTE:** The ↵ character is used to wrap a single line of code across multiple lines.

Commands

<u>MSM</u>	<u>Native Caché Mode</u>	<u>Description / Notes</u>
BREAK	BREAK, ZBREAK	Invokes the debugger. ? not supported from interactive debugger. ZGO not supported in Caché—use GO instead.
CLOSE	CLOSE	Closes a device. Caché supports MSM-like numeric devices after you define a Numeric Alias for that device—see OPEN.
DO	DO	Executes a routine or block of code. Square brackets ([]) not supported for 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). See chapter four of this guide for more information.
JOB	JOB	Spawns a new background process. In Caché, use \$ZCHILD to return the jobbed process' PID 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): s x=\$zf(-2,"c:\cachesys\bin\cache -s.\mgr -b 1024 -U %SYS ^^Test")
NEW	NEW	Stacks one or more local variables. Caché does <i>not</i> allow \$TEST or \$ZREFERENCE as arguments to the NEW command.
OPEN	OPEN	Opens a device. 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	Not Implemented	Causes current transaction to be restarted.

TSTART	TSTART	Marks the beginning of a transaction. Caché does <i>not</i> support Restart Variables or Transaction Parameters.
USE	USE	Uses a device, and sets \$IO to this current device. Caché supports MSM-like numeric devices after you define a Numeric Alias for that device—see OPEN.
VIEW	VIEW	Reads and writes blocks to disk, and writes locations in memory. Where MSM expects a Volume Group, Caché expects a namespace. VIEW commands <i>must</i> respect Caché disk and memory structures—see documentation for more details.
WRITE	WRITE	Sends output to current device. Mnemonic Spaces may need to be re-written for Caché.
ZCALL	\$ZF	Calls an external procedure
ZFLUSH	Not Implemented	Flushes all disk blocks out of the internal disk buffer cache.
ZGO	GO	Resumes execution of a program after a BREAK command.
ZHOROLOG	\$ZU(71,\$H value)	Sets date and time for current process. Caché's \$ZU(71) only allows a new date value, <i>not</i> time.
ZMSM	No Direct Equivalent	Traces the sequence of program execution within a routine and from routine to routine. In Caché, try: f i=0:1:\$stack(-1) d . w !,"Context level: ",i,?25,"Context type: ",\$stack(i) . w !,?5,"Current place: ",\$stack(i,"place") . w !,?5,"Current source: ",\$stack(i,"mcode") . w ! q
ZNEW	Not Implemented	Similar to NEW command, but variable is persistent after sub-routine explicitly or implicitly quits.
ZQUIT	ZQUIT	MSM passes control to the next higher error-processing routine that's been specified by \$ZTRAP. Caché clears entire stack, unless an argument representing the number of error trap levels to quit back is specified.
ZSETOBJ	SET	Assigns an object reference to a variable. Caché ObjectScript uses the native SET command, such as: s var=Car.Make
ZUSE	\$ZU(9), \$ZU(94)	Allows write access to any device, even if in use—broadcasting

Operators

<u>MSM</u>	<u>Native Caché Mode</u>	<u>Description / Notes</u>
#	\$ZHEX	Performs numeric conversions from hexadecimal to decimal. Caché does <i>not</i> support MSM's hexadecimal operator—use \$ZHEX instead.

Structured System Variables

<u>MSM</u>	<u>Native Caché Mode</u>	<u>Description / Notes</u>
^\$DEVICE	Not Implemented	Provides information on the existence, operational characteristics, and availability of a device.

Functions

<u>MSM</u>	<u>Native Caché Mode</u>	<u>Description / Notes</u>
\$ORDER	\$ORDER	<p>Returns next subscript at the same level of a given variable. Also loops through a list of local variables set in a partition. This looping functionality differs on the two platforms. For example, let's assume we have these variables set: %=1,%USER="mikel",var=123.</p> <p>MSM code w \$o() ; this returns “%” w \$o(%) ; this returns “%USER” w \$o(%USER) ; this returns “var”</p> <p>Caché code w \$o(@("")) ; this returns “%” w \$o(%) ; this returns “%USER” w \$o(%USER) ; this returns “var”</p>
\$VIEW	\$VIEW	Returns contents of memory locations. VIEW commands <i>must</i> respect Caché disk and memory structures—see documentation for more details.
\$ZASCII	Not Available	Returns the Unicode character code of a specified character.
\$ZBN	Not Available	Returns the starting block number for a routine or global, allocates a disk block, or de-allocates a disk block.
\$ZBname	\$ZBITname	A collection of functions that are used to perform logical operations on bit strings.
\$ZCALL	\$ZF	<p>Calls an external procedure and returns a value. Caché's \$ZF expects function names in double-quotes [<code>\$ZF("MyFunction")</code>], while MSM's \$ZCALL does not [<code>\$ZCALL(MyFunction)</code>].</p>
\$ZCHAR	Not Available	Returns a string of characters, given a list of Unicode character codes.
\$ZCRC	\$ZCRC	<p>Returns a computed checksum or cyclic redundancy check. Caché does not support the abbreviation \$ZCR—use \$ZCRC. Caché requires the <i>Type</i> argument. Caché does not support 32-bit CRC computations.</p>
\$ZCREATEOBJECT	SET	<p>Returns an object reference to a newly instantiated object. Caché ObjectScript uses the native SET command, such as: <code>s var=##class(Car).%New()</code></p>
\$ZDATE	\$ZDATE	<p>Returns an external date value, given a \$HOROLOG date. 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.</p>

\$ZDEVICE	No Direct Equivalent	Returns the actual device name, given the internal device ID. Through the use of tied terminals in Caché, the special variable \$ZIO will contain TELNET and LAT terminal information including <i>Terminal Server Address</i> , and <i>Virtual Port Number</i> . See pages 2-31 to 2-37 and pages 4-48 to 4-50 of the <i>Caché Basic System Manager's Guide</i> for more information.
\$ZGETOBJECT	SET	Retrieves database object, and returns object reference to the instantiated object. Caché ObjectScript uses the native SET command, such as: s var=##class(Car).%Open(OREF)
\$ZHEX	\$ZHEX	Performs numeric conversions from decimal to hexadecimal, and vice-versa. While MSM and Caché have identical \$ZHEX functions, MSM's hexadecimal operator (#) is <i>not</i> supported in Caché—use \$ZHEX instead.
\$ZHL	\$ZDATE, \$ZTIME, \$ZDT	Returns an external date or time value, given a \$HOROLOG date. Use \$ZDATE to convert dates and \$ZTIME to convert times, or \$ZDT to convert both.
\$ZOBJREFERENCE	Not Available	Identifies whether an expression refers to an object, and whether two expressions refer to the same object.
\$ZOS	\$ZF(-1), \$ZSEARCH, OPEN, ^%CLI	Invokes commonly used Windows functions from within M. See <i>Appendix G</i> for Windows-functions code samples.
\$ZPOSITION	Not Available	Returns the number of positions of a string that can fit in a field, on an output device.
\$ZUCI	No Direct Equivalent	Returns the UCI internal number or external name. In Caché, use \$ZNSPACE or \$ZU(5) to return the current namespace.
\$ZVERIFY	No Direct Equivalent	Returns a string of errors, if any exist, in the logical structure of the database. In Caché, use \$ZU(36) to verify database labels or \$ZU(37) to verify database header information.
\$ZWIDTH	Not Available	Returns the width that a string occupies when it is displayed on an output device.

Preprocessor Directives

<u>MSM</u>	<u>Native Caché Mode</u>	<u>Description / Notes</u>
#comment	No Direct Equivalent	Turns on the insertion of pre-expansion lines of code that contain macros into the generated code as comments. In Caché, you can use #show to enable the inclusion of comments from .INC code in the generated .INT code.
#defarray	Not Available	Defines a macro to be used for referencing an array.
#deflabel	Not Available	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	#include	Includes source code in a given routine. In Caché, #include can only be used to reference .INC code.
#library	Not Available	Specifies path to library files.
#makelib	Not Available	Creates a macro library.

#nocomment	No Direct Equivalent	Stops the inclusion of unprocessed source code lines as comments. In Caché, you can use #noshow to exclude comments from .INC code from the generated .INT code.
#noroutine	Not Available	Prevents generation of an M routine.
#prefix	Not Available	Defines the prefix used to identify a macro reference.
#routine	Not Available	Specifies the name of a routine to be generated.
#undefine	#undef	Removes a macro definition. In Caché, you <i>must</i> change all #undefine statements to #undef.
#upplib	Not Available	Updates a macro library.
#x	Not Available	Executes M code during preprocessing.

Special Variables

MSM	Native Caché Mode	Description / Notes
\$DEVICE	\$DEVICE	Indicates whether last I/O operation was successful. Caché always returns the NULL string indicating a successful I/O operation, unless you programmatically change \$DEVICE with \$ZU(96,5,"String"). For example, the following Caché code sets \$DEVICE equal to "Read Timed Out": i \$za\2#2 s x=\$ZU(96,5,"Read Timed Out")
\$ECODE	\$ECODE	Returns a list of errors encountered by the application. While MSM's and Caché's \$ECODE are the same conceptually, Caché will use Caché-specific error strings such as: ,ZSYNTAX,ZNOROUTINE,ZDISKHARD,
\$IO	\$IO	Contains the currently active device. 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	\$JOB	Contains the job number for the current process. Caché's \$JOB values correspond to the process' PID number at the OS level, while MSM's \$JOB values are MSM-specific numbers.
\$PRINCIPAL	\$PRINCIPAL	Contains a job's principal device. 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	No Direct Equivalent	MSM uses \$SYS to return 3 pieces of information: an M User Group # (43), the Serial # from the MSM license, and a unique # for the current instance of M. There is no direct equivalent to \$SYS in Caché. Here's a list of related functions: <ul style="list-style-type: none"> • \$ZU(110) returns the hostname of the machine hosting Caché • \$ZU(200,3) returns the Caché order number • \$ZU(200,4) returns the Caché serial number • \$ZU(200,8) returns the customer name
\$TRESTART	Not Implemented	Indicates the number of transaction restarts that have occurred since the initiation of the transaction.
\$ZB	\$ZB, \$ZCHILD	Returns device-specific information for the current device. When used with the JOB command. MSM's \$ZB returns the jobbed process' PID. For this functionality in Caché, use \$ZCHILD.
\$ZC	Not Implemented	Contains device-specific information for the current device. In Caché, \$ZC is used to represent both the \$ZCHILD special variable and \$ZCYC function, depending on context.
\$ZERROR	\$ZERROR	Contains the text of the error message most recently produced by the application or programmer. While MSM and Caché use \$ZERROR in the same fashion, Caché will report Caché-specific error text that may or may not correspond to MSM's error text.
\$ZLEVEL	No Direct Equivalent	Contains a number that indicates the current nesting level—use \$STACK.

APPENDIX F: Command Line and Batch File Functions

****NOTE:** The ↵ character is used to wrap a single line of code across multiple lines.

Windows Platforms

1. Under Windows NT from the command line or a batch file
 - A. To run ^%SS in the default namespace:
 - CSS CTERMINAL CACHE ^^%SS
 - B. To run ^XYZ in the USER namespace:
 - CSS CTERMINAL CACHE ^^XYZ USER
 - C. This puts the user into programmer's mode in the USER namespace:
 - CSS CTERMINAL ^^%PMODE USER
 - D. This puts the user into programmer's mode in the default namespace:
 - CSS CTERMINAL CACHE

2. Under Windows 95 / 98 from the command line and/or a batch file
 - A. To run ^%SS in the %SYS namespace:
 - CCONTROL CTERMINAL CACHE ^%%SS
 - B. To run ^XYZ in the USER namespace:
 - CCONTROL CTERMINAL CACHE ^XYZ USER
 - C. This puts the user into programmer's mode in the USER namespace:
 - CCONTROL CTERMINAL CACHE ^%%PMODE USER
 - D. This puts the user into programmer's mode in the %SYS namespace:
 - CCONTROL CTERMINAL CACHE

3. Calling out to Windows (NT, 95, and 98)
 - A. Using the \$ZF(-1) function
 - s x=\$ZF(-1,"mkdir c:\temp\cache")
 - B. Using pipes

```
dirtmp s $zt="err",dev="dir c:\temp"
      o dev:"RQ" f u dev r record u 0 w record,!
      q
err    c dev w !!,"No More Files"
      q
```

****NOTE:** On all Windows platforms, \$ZF(-1) will *not* display command output.

4. Performing startup, shutdown, and force functions under Windows NT
 - A. Under Windows NT:
 - CSS START <CONFIG>, CSS STOP <CONFIG>, CSS FORCE <CONFIG>
 - B. Under Windows 95 / 98:
 - CCONTROL START <CONFIG>, CCONTROL STOP <CONFIG>, CCONTROL FORCE <CONFIG>

UNIX Platforms

1. Under UNIX from the command line and/or a script file
 - A. To run ^%SS in the %SYS namespace:
 - `cache -U "%SYS" "^%SS"`
 - B. To run ^XYZ in the USER namespace:
 - `cache -U "USER" "^XYZ"`
 - C. This puts the user into programmer's mode in the USER namespace:
 - `cache -U "USER"`
 - D. This puts the user into programmer's mode in the %SYS namespace:
 - `cache -U "%SYS"`
2. Under UNIX from a script file
 - A. Passing script variables, string constants, globals, and environment variables to a Caché program
 - ```
The global ^GLO is set to "From"
var="Many"
echo "d ^test(\"$var\", \"Greetings\", ^GLO, \"$LOGNAME\") h" ↵
| cache -U "%SYS"
```
  - B. Passing only string constants and global references to a Caché program
    - ```
# The global ^GLO is set to "UNIX"
cache -U "SYS" << EOF
d ^test("Hello", "From", ^GLO)
h
EOF
```
3. Calling out to UNIX from Caché
 - A. Using the \$ZF(-1) function
 - `s x=$ZF(-1, "ls -l /tmp")`
 - B. Using pipes
 - ```
lstmp s $zt="err", dev="ls -l /tmp"
o dev:"RQ" f u dev r record u 0 w record,!
q
err c dev w !!, "No More Files"
q
```
4. Performing startup, shutdown, and force functions under UNIX
  - `cstart [<CONFIG>], cstop, cforce`

## APPENDIX G: %ZOS Function

---

**\*\*NOTE:** The ↵ character is used to wrap a single line of code across multiple lines.

---

```
Cache for Windows NT^INT^
%RO on 26 May 1999 3:04 PM
%ZOS.INT.1.57854,53838.
%ZOS ; Cache' Functions to Emulate some DOS Functions... ; Compiled May 26,
1999 11:32:29 for Cache for Windows
;-----
; Description: Cache' Functions to Emulate some DOS Functions.
; I N F O R M A T I O N
;-----
;
; ***** WARNING *****
; This routine makes some assumptions about the output of the
; DIR command and should be treated as a template that can be
; modified to suit local conditions. It should also be noted
; that if any DOS level errors occur then the Cache' process may
; well hang. This is especially so if DOS produces a prompt
; such as 'Abort, Retry, Fail?' and then waits for input.
;
; THIS UTILITY IS UNSUPPORTED AND WITHOUT WARRANTY.
;
; Entry points:
; =====
; shell - (Also d ^%cdos) Pseudo DOS shell
; copy(source,target) - Copy file(s)
; $$curdrive - Display current DOS drive letter
; del(file) - Delete file(s)
; dir(path) - Display directory listing
; $$exists(file) - Check to see if a file exists:
; Returns: 0 (no) or 1 (yes)
; $$files(path) - Return list of file(s):Delimited by ;
; 1) FILE.EXT - UPPERCASE NAME
; 2) Attribute - Lower Case
; 3) File Size - Number
; 4) Modified Date - YYYY-MM-DD
; 5) Modified Time - HH:MM (24 hour)
; 6) File.ext - Actual File
; md(dir) - Make directory
; mkdir(dir) - Make directory
; $$msmfdt(file) - Get MSM Field Date & Time Information
; Return: DD-MMM-YY~HH:MM {AM/PM}
; $$msmgdd(device) - Return 'data' from given 'device' and
; "eof" when no more data exists.
; $$msmodf(file) - Open DOS file either as "rs" or "wns"
; Return: 'file' name if open,
; otherwise ""
; rd(dir) - Remove directory
; ren(source,target) - Rename file(s) (??)
; rename(source,target) - Rename file(s) (??)
; rmdir(dir) - Remove directory
```

```

; $$size(file) - Size of a file
; $$subdir(path) - Current DOS directory
; $$var - Return DOS variable(s)
; $$verify - Show verify
; verify(mode) - Set Verify ("on" or "off")
; $$version - Return the OS Version
; $$volospace(drive) - Number of free bytes on a drive
;
;-----
shell ; Provide pseudo DOS shell
n
s drive=$$curdrive
s dir=$$subdir(drive)
s prompt=drive_:"
i $(dir)'="\ " s prompt=prompt_"\ "
s prompt=prompt_dir_>"
shell1 s $zt="shell2"
w !,prompt
r line
i line="" g shell1
s line=$zcvr(line,"l")
i line="q"!(line="exit") q
s cmd=$p(line," "),a=$p(line," ",2,$l(line," "))
d lspb
w !
i cmd="copy" d copy($p(a," ",1),$p(a," ",2)) g shell1
i cmd="del" d del(a) g shell1
i cmd="dir" d dir(a) g shell1
i cmd="md" d md(a) g shell1
i cmd="mkdir" d mkdir(a) g shell1
i cmd="rd" d rd(a) g shell1
i cmd="ren" d ren($p(a," ",1),$p(a," ",2)) g shell1
i cmd="rename" d rename($p(a," ",1),$p(a," ",2)) g shell1
i cmd="rmdir" d rmdir(a) g shell1
i cmd="set" d g shell1
.s vars=$$var
.f i=1:1:$l(vars,$c(0)) w $p(vars,$c(0),i),!
i cmd="verify" d g shell1
.i a="" w $$verify,!
.i a'="" d verify(a)
i cmd="volospace" w $$volospace(a)," bytes free",! g shell1
i cmd="?"!(cmd["help"]) d g shell1
.w "Valid commands are:",!
.f
i="copy","del","dir","md","mkdir","rd","rmdir","ren","rename","set","verify","v
olospace" w i,!
shell2 w "Bad command or file name",!
g shell1
shellq q
;-----
;
;
dir(spec) ; Output directory listing
n exec,file,a
s spec=$g(spec)

```

```

s exec="dir "_spec
s file=$$exec(exec,1)
s $zt="dir1"
o file:"r":0 e g dir1
f u file r a u 0 w a,!
dir1 c file
d del(file)
l -^%dosfile
dirq q
;-----
;
;
del(spec) ; Delete specified files
n exec
s spec=$g(spec)
s exec="del "_spec
d exec(exec,0)
delq q
;-----
;
;
files(spec) ; Return list of files that match pattern
n (spec)
s spec=$g(spec)
i $$os="NT" s exec="dir "_spec_" /v /a"
e i "NT"="NT" s exec="dir "_spec_" /x"
e s exec="dir "_spec_" /v /a"
s file=$$exec(exec,1)
s $zt="files1"
o file:"r":0 e g files1
s str=""
i $$os="WIN" DO ; --- Windows Specific "DIR" Output ---
.f d
..u file r a
..i $e(a)=" "!(($e(a,9)'=" ")&($e(a,9)'=".")) q
..s pre=$tr($e(a,1,8)," ") ; 8.3 DOS file name
..s suf=$tr($e(a,10,12)," ")
..s dosnam=pre
..i pre[".",suf'="" s dosnam=dosnam_"."
..s dosnam=dosnam_suf
..s a=$e(a,13,$l(a))
..d lspc ; No leading spaces
..s f=$f(a," ") ; Find next space
..s size=$e(a,1,f-2),size=+$tr(size,".") ; Actual size (no punctuation)
..s a=$e(a,f-1,$l(a))
..d lspc
..s f=$f(a," ") ; Skip allocated size
..s a=$e(a,f-1,$l(a))
..d lspc
..s f=$f(a," ")
..s dat=$e(a,1,f-2) ; 'Modified' date
..s a=$e(a,f-1,$l(a))
..s yy=$p(dat,"-",3)
..s mm=$p(dat,"-",2)
..s dd=$p(dat,"-",1)

```

```

..i yy>70 s yy=19_yy
..s dat=yy_"-"_mm_"_"_dd
..d lspc
..s f=$f(a," ")
..s tim=$e(a,1,f-2) ; 'Modified' time
..s a=$e(a,f-1,$l(a))
..s hh=+tim,mm=$p(tim,":",2)
..i mm["p" s hh=hh+$s(hh=12:0,1:12)
..i mm["a",hh=12 s hh=0
..s mm=$e(100+mm,2,3),hh=$e(hh+100,2,3)
..s tim=hh_"": "_mm
..d lspc
..s f=$f(a," ") ; Skip Accessed date
..s a=$e(a,f-1,9999)
..s attr=$e(a,1,8),a=$e(a,9,9999)
..s attr=$zcvr(attr,"l"),attr=$str(attr," ")
..d lspc
..s name=a ; 'Real' file name
..s str=str_dosnam_";"_attr_";"_size_";"_dat_";"_tim_";"_name_$c(10)
e DO ; --- DEFAULT: Windows NT Specific "DIR" Output ---
.f d
..u file r a
..i $e(a,1)=" !((($e(a,9)'=" ")&($e(a,3)'="/")) q
..s f=$f(a," ")
..s dat=$e(a,1,f-2) ; 'Modified' date
..s a=$e(a,f-1,$l(a))
..s mm=$p(dat,"/",1)
..s dd=$p(dat,"/",2)
..s yy=$p(dat,"/",3)
..i yy>70 s yy=19_yy
..s dat=yy_"-"_mm_"_"_dd
..d lspc
..s f=$f(a," ")
..s tim=$e(a,1,f-2) ; 'Modified' time
..s a=$e(a,f-1,$l(a))
..s hh=+tim,mm=$p(tim,":",2)
..i mm["p" s hh=hh+$s(hh=12:0,1:12)
..i mm["a",hh=12 s hh=0
..s mm=$e(100+mm,2,3),hh=$e(hh+100,2,3)
..s tim=hh_"": "_mm
..d lspc
..i $e(a)="<" DO ; Handle '<DIR>' stuff...
...s f=$f(a," ") ; Ignore '<DIR>'
...s attr=$e(a,1,f-2)
...s a=$e(a,f-1,$l(a))
...s attr="d"
...d lspc
..e s attr=""
..s f=$f(a," ") ; Find next space
..i a'="",'f s size=a
..e s size=$e(a,1,f-2)
..i 'f s
(dosnam,name)=$s(size="."$zcvr($p(spec,"\\", $l(spec,"\\")), "u"),1:size),size="",
str=str_dosnam_";"_attr_";"_size_";"_dat_";"_tim_";"_name_$c(10) QUIT
..s size=+$str(size,".") ; Actual size (no punctuation)

```

```

 ..s a=$e(a,f-1,$l(a))
 ..d lspc
 ..s pre=$p($p(a,".",1)," ") ; 8.2 DOS file name
 ..s suf=$p($p(a,".",2)," ")
 ..s dosnam=pre
 ..i pre'[".",suf'="" s dosnam=dosnam_". "
 ..s dosnam=dosnam_suf
 ..s a=$e(a,f-1,9999)
 ..d lspc ; No leading spaces
 ..s name=$s(a="":dosnam,1:a) ; 'Real' file name
 ..s str=str_dosnam_"_attr_"_size_"_dat_"_tim_"_name_$c(10)
filesl c file
d del(file)
l -^%dosfile
filesq q str
;-----
;
;
size(spec) ; Get size of single file (-1 indicates failure)
s spec=$g(spec) q:spec="" -1
s spec=$$files(spec)
i $l(spec,$c(10))>2!(spec="") q -1
sizeq q $p(spec,";",3)
;-----
;
;
volSPACE(spec) ; Get space left on drive
n exec,file,a,b,f,i
s spec=$g(spec,$$curdrive)
i spec?la s spec=spec_" ":"
s exec="dir "_spec_"\"
s file=$$exec(exec,1)
s $zt="volSPACE1"
o file:"r":0 e g volSPACE1
s b="" f i:1:1 s a=b u file r b
volSPACE1 ;
c file
d del(file)
l -^%dosfile
s a=b
d lspc
i $$os="WIN" DO ; --- Windows 95 Specific "DIR" Output ---
.s f=$f(a," "),a=$e(a,f,9999)
.s f=$f(a," "),a=$e(a,f-1,9999)
.d lspc
.s a=$p(a," ")
e DO ; --- DEFAULT: Windows NT Specific "DIR" Output ---
.s a=$p(a," ")
volSPACEq q $str(a,".")
;-----
;
;
curdrive() ; Get current drive
n file,a
s file="dir"

```

```

s $zt="curdrive1"
o file:"rq":0 e g curdrive1
u file r a
i $$os'="NT" r a
curdrive1 c file
curdriveq q $p(a," ",5)
;-----
;
;
subdir(spec) ; Get current directory
n file,a,dir
s spec=$g(spec)
i spec="" s spec=$$curdrive
i spec?la s spec=spec_" ":"
s file="dir "_spec
s $zt="subdir1"
o file:"rq":0 e g subdir1
u file r a,a,a,a
subdir1 c file
s dir=$p(a," ",4)
s dir=$p(dir,"\\",2,$l(dir,"\\"))
i dir="" s dir="\"
subdirq q dir
;-----
;
;
verify(spec) ; Check/Set verification of file copy
n flag,exec,file,a
s spec=$g(spec)
s spec=$zcvl(spec,"l")
s flag=0 i spec'="on",spec'="off",spec'="" s spec="",flag=1
i spec'="" s exec="verify "_spec d exec(exec,0) q
s exec="verify"
s file=$$exec(exec,1)
s $zt="verify1"
o file:"r":0 e g verify1
u file r a
verify1 c file
d del(file)
l -^%dosfile
i flag w a,! q
verifyq q a
;-----
;
;
copy(source,target) ; Copy files
n exec
s source=$g(source),target=$g(target)
i source=""!(target="") q
s exec="copy "_source_" "_target
d exec(exec,0)
copyq q
;-----
;
;

```

```

rename(source,target) ; Rename files
 s source=$g(source),target=$g(target)
 d ren(source,target)
renameq q
 ;
ren(source,target) ; Rename files
 n exec
 s source=$g(source),target=$g(target)
 i source="!"(target="") q
 s exec="ren "_source_" "_target
 d exec(exec,0)
renq q
 ;-----
 ;
 ;
mkdir(spec) ; Make a directory
 s spec=$g(spec)
 d md(spec)
mkdirq q
 ;
md(spec) ; Make a directory
 n exec
 s spec=$g(spec)
 s exec="md "_spec
 d exec(exec,0)
mdq q
 ;-----
 ;
 ;
rmdir(spec) ; Remove a directory
 s spec=$g(spec)
 d rd(spec)
rmdirq q
 ;
rd(spec) ; Remove a directory
 n exec
 s spec=$g(spec)
 s exec="rd "_spec
 d exec(exec,0)
rdq q
 ;-----
 ;
 ;
type(spec) ; Type a file
 n a
 s spec=$g(spec) q:spec=""
 s $zt="type1"
 o spec:"r":0 e q
 f u spec r a u 0 w a,!
type1 c spec
typeq q
 ;
 ;
 ;
var(spec) ; DOS variables

```

```

n exec,file,str,flag,a
s spec=$g(spec)
s exec="set"
s file=$$exec(exec,1)
s $zt="var1"
o file:"r":0 e g var1
s str="",flag=0
f u file r a s str=str_a_$c(0) i $zcvt($p(a,"="),"1")=$zcvt(spec,"1")
s str=$p(a,"=",2,999),flag=1 ;q
var1 c file
d del(file)
l -^%dosfile
i spec'=""','flag s str=""
varq q str
;-----
;
;
exists(file) ; Check to see if a file exists and return 0 or 1
n data
s $zt="existse^"_$zn
o file:"r":0 e q 0
u file r data
c file
q 1
existse ; Error Trap for exists
c file
q $s($ze["<ENDOFFILE>":0,1:1)
;-----
;
;
version() ; Get DOS Version
n file,a
s file="ver"
s $zt="version1"
o file:"rq":0 e g version1
u file r a,a
version1 c file
d del(file)
versionq q a
;-----
;
;
msmfdt(file) ; Get MSM Field Date & Time Information.
; Return: DD-MMM-YY~HH:MM {AM/PM}
n date,hh,mm,time
s date=$$files(file)
s time=$p(date,";",5),date=$p(date,";",4)
g:date="" msmfdtq
s date=$p(date,"-",3)_"-
"_$p("JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC",",",+$p(date,"-",2))_"-
"_$e($p(date,"-"),3,4)
s mm=$p(time,":",2)_" "_$s(+time>11&(time'=24):"PM",1:"AM")
s hh=$s(+time<13:+time,1:time-12)
s time=hh_"":_mm
msmfdtq q date_"~"_time

```

```

;
msgdd(device) ; Return 'data' from given 'device' and
; "eof" when no more data exists.
n data s $zt="msgdde^"_$zn
u device r data
q data
msgdde ; Error Trap for msgdd
q $(ze["<ENDOFFILE>":"eof",1:""])
;
msmodf(file) ; Open a DOS file either as "rs" or "wns"
; Return: 'file' name if open, otherwise ""
o file:"rs":0 i u file q file
o file:"wns":0 i u file q file
q ""
;
exec(exec,out) ; Executable string
n x,file
i out d q file
.l +^%dosfile
.s file="cachexec.tmp"
.s x=$zf(-1,exec_" > "_file)
i 'out d
.s x=$zf(-1,exec)
q
;-----
;
lspc ; Strip leading spaces
i $(a)=" " s a=$(a,2,9999) g lspc
q
;
os() q $(zv["NT":"NT",1:"WIN"])

```

## APPENDIX H: Overview of System Architecture

| <u>Parameter</u>                                | <u>MSM (Maximum) Value</u>                              | <u>Caché (Maximum) Value</u> |
|-------------------------------------------------|---------------------------------------------------------|------------------------------|
| Database file names                             | Any valid filename                                      | CACHE.DAT / CACHE.EXT        |
| Global buffer size                              | 1kb                                                     | 2kb                          |
| Routine buffer size                             | 1kb                                                     | 32kb                         |
| Routine source code size                        | At least 200kb, depending on stack and partition sizes. | 32kb                         |
| Number of global buffers                        | 2047mb **See NOTE                                       | 2048mb                       |
| Number of routine buffers                       | 2047mb **See NOTE                                       | 2048mb                       |
| User partition size                             | 16mb                                                    | 16mb                         |
| Framestack size                                 | 256kb                                                   | 64kb                         |
| Database size                                   | 16gb                                                    | 16gb                         |
| Local variable, global, and routine name length | 8 significant characters                                | 31 significant characters    |
| Length of all local variable subscripts         | **See NOTE                                              | 255 bytes                    |
| Length of all global subscripts                 | 255 bytes                                               | 255 bytes                    |
| Number of local variable subscript levels       | **See NOTE                                              | 129                          |
| Number of global subscript levels               | 127                                                     | 127                          |
| Data length for local variables                 | **See NOTE                                              | 32kb                         |
| Data length for globals                         | 511 bytes                                               | 32kb                         |
| Numeric precision after decimal point           | 15 digits                                               | 17 digits                    |

---

**\*\*NOTE:** MSM uses a single buffer pool for both global and routine buffers. MSM will allow as many global and routine buffers that can fit within the configured memory pool.

**\*\*NOTE:** MSM allows as large a subscript length, as many subscript levels, and as large a data length for local variables as MSM's system expression stack can handle. These values are generally lower in Caché, with the exception of permissible data length which is generally much higher in Caché.

---

## APPENDIX I: Automating Caché Backups

**\*\*NOTE:** The ↵ character is used to wrap a single line of code across multiple lines.

### Automate Caché's Concurrent Backup from OS

```
ZBACKUP ; Routine to perform various backups from Operating
; System level
;
; This utility currently performs the backups like this:
; Full backups done on Sunday and Wednesday
; Incrementals done on Monday, Thursday, and Friday
; Cumulatives done on Tuesday and Saturday
;
; This sample code does not use an 'E'xternal backup, but
; the code is set up to easily take this change if you want.
;
; See the ^DBACK routine for documentation.
;
; This utility is unsupported and without warranty.

START n
d BCKPREP, BACKUP
q

BCKPREP ; Do some prep work for the backup
d DAYOFWK ; Figure out what day it is
s argfile="", logfile="c:\temp\backup.log"
s type=$s("14"[CurrentDay:"F", "256"[CurrentDay:"I", "37"[↵
CurrentDay:"C", 1:"")
s desc="Cach"_$c(233)_ " Backup Performed on "$_zd($h)
s outdev="c:\temp\test.bak" ; Change this to your output device
s mode="NOISY"
s kiljrn="N", clrjrn="N", swjrn="Y"
s nwjrnfil=$p($p(^%SYS("JOURNAL", "CURRENT"), "."), "^", 2)_"_ ↵
str($j($p(^%SYS("JOURNAL", "CURRENT"), ".", 2)+1, 3), " ", 0)
q

DAYOFWK ; OUTPUT : Day of month 1-7 (1=Sunday, 2=Monday...7=Saturday)
n %DS, %DN, date, month, year, FirstDay
s date=$zd($h)
s month=$p(date, "/"), year=$p(date, "/", 3)
s %DS=month_"/1/"_year
d INT^%DATE
s FirstDay=%DN+4#7+1
s CurrentDay=(FirstDay+$p(date, "/", 2)-2)#7+1
q

BACKUP ; Perform the backup
i type="E" d CLRINC
s OK=$$BACKUP^DBACK(argfile, type, desc, outdev, kiljrn, logfile, ↵
```

```
mode,clrjrn,swjrn,nwjrnfil)
i 'OK o logfile:"WAS" u logfile w !!, "*** Backup was ↵
 UNSUCCESSFUL! ***",!! u 0 c logfile
q
```

```
CLRINC ; Clear out the Incremental Bitmaps for 'E'xternal backups so
; that our cumulatives don't pick up tons of extra data!
s x=$$CLRINC^DBACK(1)
q
```

## Freeze Caché while an OS Backup is Performed

```

%cbbackup ;
#IF 0 ;-----
; Description: External Backup Module
; This utility is unsupported and without warranty.
#ENDIF ;-----
#;-----
; Note: sw12 Disables Sign-On
startb ; Call the $$start stuff as a call-in for batch processing
n file,start
s file="c:\cachebu.flg"
s start=$$start(1)
o file:"wns":5 i $t u file w start c file
QUIT
#;-----
start(quiet,detail) ; Call Before External Backup Performed...
n ns
s $zt="starte",$ze=""
s quiet=+$g(quiet),detail=+$g(detail)
s ns=$ZU(5) ; Retain Original Namespace
i $ZU(5,"%SYS") ; Change to %SYS Namespace
g:$${%swset^SWSET(10,1)<0 starte ; Disable Database Reads
g:$${%swset^SWSET(13,1)<0 starte ; Disable Database Writes
d quiesce(quiet) ; Quiesce the Daemons
g:$${%swset^SWSET(10,0)<0 starte ; Reset Database Reads
i $ZU(5,ns) ; Change to Orig. Namespace
QUIT 1
starte ; Error On Start...
i $${%swset^SWSET(10,0) ; Reset Database Reads
i $${%swset^SWSET(13,0) ; Reset Database Writes
i $g(ns)'="",$ZU(5,ns) ; Change to Orig. Namespace
QUIT 0_$s($g(detail):$c(1)_$ze,1:"")
#;-----
quiesce(quiet) ; Quiesce the Daemons...
s $zt="quiescee",$ze=""
i '$ZU(51,1+2+4+8) DO ; Wait for Daemons to Quiesce...
. i 'quiet w !,"Waiting for disk cleanup to finish..."
. n i
. f i=1:1 h .1 Q:$ZU(51,1+2+4+8) i 'quiet,i#10=0 w "."
quiescee ; Exit Quiesce Here...
s $ze=""
QUIT
#;-----
stopb ; Call the $$stop stuff as a call-in for batch processing
n file,stop
s file="c:\cachebu.flg"
i $zf(-1,"del "_file)
s stop=$$stop(1)
o file:"wns":5 i $t u file w stop c file
QUIT
#;-----
stop(quiet,desc,detail) ; Call After External Backup Performed...
n ns

```

```

s $zt="stope",$ze=""
s quiet=+$g(quiet),detail=+$g(detail)
s:$g(desc)="" desc="External Cache' Backup Successfully ↵
 Completed: "_$zdt($h,3)
s ns=$ZU(5) ; Retain Orignal Namespace
i $ZU(5,"%SYS") ; Change to %SYS Namespace
g:$%swset^SWSET(13,0)<0 stope ; Reset Database Writes
g:'$$BACKUP^DBACK("", "E",desc,"","") stope ; Mark ↵
 Backup
i $ZU(5,ns) ; Change to Orig. Namespace
QUIT 1
stope ; Error On Stop...
i $$%swset^SWSET(13,0) ; Reset Database Writes
i $g(ns)'="", $ZU(5,ns) ; Change to Orig. Namespace
QUIT 0_$s($g(detail):$c(1)_$ze,1:"")
#;-----

```