Caché Monitoring Guide

Version 2018.1
2024-05-02

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

Caché Monitoring Guide
Caché Version 2018.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

About This Book

This book describes several tools available for monitoring Caché.

The following chapters describe how to monitor Caché with tools included with Caché:

- Monitoring Caché Using the Management Portal describes how to monitor the many metrics displayed on the System Dashboard of the Management Portal, which shows you the state of your Caché instance at a glance.
- Using the Caché Diagnostic Report describes how to generate a Diagnostic Report and send it to the WRC.
- Using Caché Monitor — Caché Monitor monitors the Caché console log for entries of the configured severity and generates a notification for each. These notifications are written to the alerts log by default but can instead be sent by email to specified recipients.
- Using Caché System Monitor — Caché System Monitor is a flexible, user-extensible utility used to monitor a Caché instance and generate notifications when the values of one or more of a wide range of metrics indicate a potential problem. System Monitor includes a status and resource monitor, which samples important system status and resource usage indicators and generates notifications based on fixed statuses and thresholds; Caché Health Monitor, which samples key system and user-defined metrics and generates notifications based on user-configurable parameters and established normal values; and Caché Application Monitor, which samples significant system metrics, stores them in local namespace globals, and generates notifications based on user-created alert definitions. Alert notifications written to the console log by System Monitor and Health Monitor can be sent by email using Caché Monitor; Application Monitor alerts can be configured for email by the user.

The following chapters describe how to monitor Caché with system routines:

- Gathering Global Activity Statistics Using ^GLOSTAT
- Monitoring System Performance Using ^PERFMON
- Monitoring Routine Performance Using ^PROFILE
- Examining Routine Performance Using ^%SYS.MONLBL
- Monitoring Block Collisions Using ^BLKCOL
- Monitoring Performance Using ^pButtons
- Monitoring Performance Using ^mgstat
- Using the Caché History Monitor

The following appendixes describe how to monitor Caché with various third-party tools:

- Monitoring Caché Using BMC PATROL
- Monitoring Caché Using SNMP
- Monitoring Caché Using WMI
- Monitoring Caché Using Web Services
- Monitoring Caché Using the cstat Utility

For detailed information, see the Table of Contents.

For general information, see *Using InterSystems Documentation*.

1

Monitoring Caché Using the Management Portal

You can monitor many aspects of your Caché instance starting at the System Dashboard of the Management Portal. From the dashboard you can view performance indicators and then, for selected indicators, navigate to more detailed information. This chapter describes the following monitoring tasks:

- Monitoring System Dashboard Indicators
- Monitoring System Performance
- Monitoring Locks
- Monitoring Log Files

See Caché System Monitoring Tools in the "Using Caché Monitor" chapter of this guide for an overview of general Caché instance monitoring tools.

1.1 Monitoring System Dashboard Indicators

The **System Operation** > **System Dashboard** page of the Management Portal groups the status of key system performance indicators into the following categories. Each category is described in one of the tables that follow.

- System Performance Indicators
- ECP and Shadowing Indicators
- System Time Indicators
- System Usage Indicators
- Errors and Alerts Indicators
- Licensing Indicators
- Task Manager Indicators

In most cases, you can click an indicator listed in one of these categories to display a description of the indicator in the bottom detail box at the lower left corner of the page.

Table 1–1: System Performance Indicators

Indicator	Definition
Globals/Second	Most recently measured number of global references per second.
Global Refs	Number of global references since system startup.
Global Sets	Number of global **Set** and **Kill** operations since system startup.
Routine Refs	Number of routine loads and saves since system startup.
Logical Requests	Number of logical block requests since system startup.
Disk Reads	Number of physical block read operations since system startup.
Disk Writes	Number of physical block write operations since system startup.
Cache Efficiency	Most recently measured cache efficiency (Global references / (physical reads + writes)).

Click the **... more details** link in the bottom detail box to display the **System Operation** > **System Usage** page. See the Monitoring System Performance section for details.

Table 1–2: ECP and Shadowing Indicators

Indicator	Definition
Application Servers	Summary status of ECP (Enterprise Cache Protocol) application servers connected to this system.
Application Server Traffic	Most recently measured ECP application server traffic in bytes per second.
Data Servers	Summary status of ECP data servers to which this system is connected.
Data Server Traffic	Most recently measured ECP data server traffic in bytes per second.
Shadow Source	Summary status of shadow connections on this data source.
Shadow Server	Summary status of shadows configured on this shadow server.

For more information on the first four indicators, the ECP indicators, see the "Configuring Distributed Systems" chapter of the *Caché Distributed Data Management Guide*.

Click the **... more details** link in the bottom detail box for the two shadow indicators to display the **System Operation** > **Shadow Servers** > **System as Data Source** page or the **System Operation** > **Shadow Servers** > **System as Shadow Server** page. For more information on shadowing, see the "Shadowing" chapter of the *Caché Data Integrity Guide*.

Table 1–3: System Time Indicators

Indicator	Definition
System Up Time	Elapsed time since this system was started.
Last Backup	Date and time of last system backup.

You can run backups or view the backup history from the **System Operation** > **Backup** page. For more information on developing a backup plan, see the "Backup and Restore" chapter of the *Caché Data Integrity Guide*.

Table 1–4: System Usage Indicators

Indicator	Definition
Database Space	Indicates whether there is a reasonable amount of disk space available for database files. Clicking **... more details** displays the **System Operation** > **Databases** page.
Journal Space	Indicates whether there is a reasonable amount of disk space available for journal files. Clicking **... more details** displays the **System Operation** > **Journals** page.
Journal Entries	Number of entries written to the system journal. Clicking **... more details** displays the **System Operation** > **Journals** page.
Lock Table	Current status of the system Lock Table. Clicking **... more details** displays the **System Operation** > **Locks** > **Manage Locks** page.
Write Daemon	Current status of the system Write daemon.
Transactions	Current status of open local and remote (ECP) transactions. If there are no open transactions, status is **Normal**; status may also be **Warning** (if the duration of the longest open local or remote transaction is greater than 10 minutes) and **Troubled** (if greater than 20 minutes). Clicking **... more details** displays the **Transactions** page.
Processes	Most recent number of running processes. Clicking **... more details** displays the **Processes** page (**System Operation** > **Processes**).
CSP Sessions	Most recent number of CSP sessions. Clicking **... more details** displays the **CSP Sessions** page (**System Operation** > **CSP Sessions**).
Most Active Processes	Running processes with highest amount of activity (number of commands executed). Clicking **... more details** displays the **Processes** page (**System Operation** > **Processes**).

For more information on any of these topics, click the **Help** link on the portal page displayed when you click the **... more details** link.

Table 1–5: Errors and Alerts Indicators

Indicator	Definition
Serious Alerts	Number of serious alerts that have been raised. Clicking **... more details** displays the **View Console Log** page (**System Operation** > **System Logs** > **Console Log**).
Application Errors	Number of application errors that have been logged. Clicking **... more details** displays the **View Application Error Log** page (**System Operation** > **System Logs** > **Application Error Log**).

See the Monitoring Log Files section in this chapter for more details.

Table 1–6: Licensing Indicators

Indicator	Definition
License Limit	Maximum allowed license units for this system.
Current License Use	License usage as a percentage of available license units.
Highest License Use	Highest license usage as a percentage of available license units.

Click the **... more details** link in the bottom details box to display the **System Operation** > **License Usage** page. For more information on licensing, see the "Managing Caché Licenses" chapter of the *Caché System Administration Guide*.

Table 1–7: Task Manager Upcoming Tasks

Indicator	Definition
Upcoming Tasks	Lists the next five tasks scheduled to run.
Task	Name of the upcoming task.
Time	Time the task is scheduled to run.
Status	Task status—one of: scheduled, completed, running.

Click the **... more details** link in the bottom details box to display the **System Operation** > **Task Manager** > **Upcoming Tasks** page. For details on the Task Manager, see the Using the Task Manager section of the "Managing Caché" chapter of the *Caché System Administration Guide*.

1.2 Monitoring System Performance

System performance metrics are described in the following tables:

- System Usage Table
- Generic (Shared) Memory Heap Usage

1.2.1 System Usage Table

To view the system usage statistics, navigate to the **System Operation** > **System Usage** page.

Table 1–8: System Usage Statistics

Statistic	Definition
Global references (all)	Logical count of accesses to globals, including **Set**s, **Kill**s, **$Data**, **$Order**, **$Increment**, **$Query**, and global references in expressions.
Global update references	Logical count of global references that are **Set**, **Kill**, or **$Increment** operations.
Routine calls	Number of calls to a routine.
Routine buffer loads and saves	Total number of routine loads and saves as a result of **ZLoad**, **ZSave**, and running routines. (In a well-tuned environment, this number increases slowly, since most routine loads are satisfied by the routine cache memory without accessing the disk. Each routine load or save transfers up to 32 KB of data (64 KB for Unicode).)
Logical block requests	Number of database blocks read by the global database code. (In a well-tuned environment, many of these reads are satisfied without disk access.)
Block reads	Number of physical database blocks read from disk for both global and routine references.
Block writes	Number of physical database blocks written to disk for both global and routine references.
WIJ writes	Number of blocks written to the write image journal file.

Statistic	Definition
Journal entries	Number of journal records created—one for each database modification (**Set** , **Kill**, etc.) or transaction event (**TStart**, **TCommit**) or other event that is saved to the journal.
Journal block writes	Number of 64-KB journal blocks written to the journal file.
Routine lines	Number of routine lines executed since system startup.
Last update	Date and time stamp of the displayed statistics.

See the "Gathering Global Activity Statistics with ^GLOSTAT" chapter for an alternative method of monitoring these statistics.

1.2.2 Generic (Shared) Memory Heap Usage

To view the Caché's generic memory heap (gmheap) usage, referred to on this page as shared memory heap (SMH) usage, navigate to the **System Operation** > **System Usage** page, and click the **Shared Memory Heap Usage** link.

Note: To change the size of the generic memory heap or gmheap (sometimes known as the shared memory heap or SMH), navigate to the **Advanced Memory Setting** page (**System Administration** > **Configuration** > **Additional Settings** > **Advanced Memory**); see Advanced Memory Settings in the "Caché Additional Configuration Settings" chapter of the *Caché Additional Configuration Settings Reference* for more information.

The column headings in the table on this page refer to the following:

- **Description** — Purpose for which generic memory is allocated.
- **Allocated SMH/ST** — Total generic (SMH) and string table memory allocated to the purpose.
- **SMH/ST Available** — Generic (SMH) and string table memory allocated to the purpose that is still available.
- **SMH/ST Used** — Generic (SMH) and string table memory allocated to the purpose that is in use.
- **SMT Used** — Static memory table memory in use by the purpose.
- **GST Used** — General string table memory in use by the purpose.
- **All Used** — Total combined memory in use by the purpose.

Table 1–9: Shared Memory Heap Usage

Identifier	Definition
Miscellaneous	Shared memory allocated for the static memory table (SMT) and general string table (GST).
Audit System	Shared memory used for system auditing.
Classes Instantiated	Shared memory allocated/available/used for the class hash table and control blocks.
Database Encryption Key Change	Shared memory allocated/available/used for database encryption key changes.
Semaphore Objects	Shared memory allocated/available/used for semaphore objects.
Event System	Shared memory allocated/available/used for the event system.
Global Mapping	Shared memory allocated/available/used for global mapping and subscript-level mapping (SLM).

Identifier	Definition
License Upgrade	Shared memory allocated/available/used for license upgrades.
Lock Table	Shared memory allocated/available/used for the lock system.
National Language Support	Shared memory allocated/available/used for National Language Support (NLS) tables.
Performance Monitor	Shared memory allocated/available/used for the Caché Performance Monitor (**^PERFMON**).
Process Table	Shared memory allocated/available/used for the Process ID (PID) table.
Routine Buffer in Use Table	Shared memory allocated/available/used for routine buffer-in-use tables.
Security System	Shared memory allocated/available/used for the security system.
Shadowing	Shared memory allocated/available/used for shadowing.
Shared Library	Shared memory allocated/available/used for shared libraries.
TTY Hash Table	Shared memory allocated/available/used for TTY hash tables.
DB Name & Directory	Shared memory allocated/available/used for database names and directories.
iKnow Language Model Data	Shared memory allocated/available/used for iKnow language models.
ECP	Shared memory allocated/available/used for ECP.
Expand Daemon	Shared memory allocated/available/used for expanding daemons.
Total	Total memory for each column. **Note:** Hover over the column headings for a description of each column.
Available SMT & GST	Available memory in the static memory table (SMT) and general string table (GST).
Total SMT & GST Allocated	Total used and available memory in the static memory table (SMT) and general string table (GST).
Total SMH Pages Allocated	Total directly allocated shared memory heap (SMH) and string table allocated Shared memory, together with the total used/available memory in the static memory table (SMT) and the general string table (GST); the number of 64-KB pages is displayed parenthetically.

1.3 Monitoring Locks

Caché locks are created when a Caché process issues a LOCK command on a Caché local variable or global variable, as long as the entity is not already locked by another process. Entities need not exist in the database to lock them. The article *Locking and Concurrency Control* discusses Caché locks in detail.

To display locks system-wide, navigate to the **System Operation** > **Locks** > **View Locks** page. To delete selected locks system-wide, navigate to the **System Operation** > **Locks** > **Manage Locks** page. In both cases, the displayed lock table lists one row for each held lock and for each waiting lock request, identifying the owner. A single row may identify multiple locks held

by an owner on the same entity. For example, holding an incremented lock or holding both a Shared lock and an Exclusive lock. If more than one process holds a lock on the same entity, each owner has its own row.

The **Lock Table** has the following column entries.

Column Heading	Definition
Owner	The process ID of the process holding or waiting for the lock. Contains the client system name if it is a remote lock.
ModeCount	Lock mode and lock increment count. If the lock count is 1 the count is not displayed. For a list of ModeCount values, refer to the "Lock Management" chapter of *Using Caché ObjectScript*.
Reference	Lock reference string of the lock item (does not include the database name).
Directory	The database location of the lock item.
System	The system name of where the lock is located, if it is the local system the column is blank.
Routine	The routine line currently being executed by the process holding or waiting for the lock.
Remove	Manage Locks only: If this lock is removable, this option along with the **Remove all locks for process** option (for local locks) or the **Remove all locks from remote client** option (for remote locks) appears in the row. Click the appropriate option to remove the lock. remove all locks for the process, or remove all locks from the remote client. If a lock you are removing is part of an open transaction, you are warned before confirming the removal.

In most cases, the only time you need to remove locks is as a result of an application problem.

For a more in-depth description of the **LOCK** command and its features, see the LOCK entry of the *Caché ObjectScript Reference*.

You may need to enlarge the size of the lock table if your system uses a large number of locks. You can do this using the Management Portal:

1. Navigate to the **System Administration** > **Configuration** > **Additional Settings** > **Advanced Memory** page.
2. In the locksiz (locksiz) row, click **Edit**.
3. In the **locksiz** box, update the amount of memory allocated on your system for locks (in bytes), and click **OK**.

   The minimum is `65536`; the maximum value depends on the value of **gmheap** (Generic Memory Heap Size). Increase the heap size if you need more room for the lock table. Caché rounds up the value to the next multiple of 64 KB. The default range is from `65536` – `1769472`.
4. Click **Save** and restart Caché for this information to take effect.

For more detailed information and alternative ways to manage locks, see the "Lock Management" chapter of *Using Caché ObjectScript*.

1.4 Monitoring Caché Logs

Caché provides the following logs for monitoring various aspects of its operation:

- Several log files are available in the system manager directory; two can be viewed using the management portal.

- You can view the application error log or ODBC error log using the management portal
- The contents of the Caché system error log, or syslog, can be reviewed using one of several methods.

1.4.1 System Manager Directory Log Files

The following log files are available in the system manager directory, typically *install-dir*\mgr. The console log and System Monitor log can be viewed using the management portal.

console log

Caché reports general messages, startup/shutdown, license, and network errors, certain operating system errors, and the success or failure of jobs started remotely from other systems through an operator console facility, which writes them to the console log, *install-dir*\mgr\cconsole.log by default. Caché System Monitor also writes notifications to the console log.

On Windows-based platforms, all console messages are sent to the console log file, the name of which is configurable. On UNIX®/Linux platforms, you can configure console messages to be sent to the console log file, the console terminal, or both. See the console parameter in the *Caché Parameter File Reference* and Advanced Memory Settings in the *Caché Additional Configuration Setting Reference* for information about console log configuration.

The size of the cconsole.log file is monitored by System Monitor. The file grows until it reaches the configured maximum size (default 5 MB), at which point it is renamed to console.log.old_*yyyymmdd*, any existing console.log.old_*yyyymmdd* file is deleted, and a new console.log is created. The maximum number of megabytes used by the console log is therefore twice the configured maximum. To configure the maximum console log size, navigate to the **System Administration** > **Configuration** > **Additional Settings** > **Startup** page of the Management Portal and update the MaxConsoleLogSize setting.

If the console log or System Monitor log is larger than 1 MB, only the most recent 1 MB portion is displayed by the Management Portal. Click the **Show entire file** link to display the entire file, which may require some time if the file is very large.

Note: If you have trouble starting Caché, use any text editor or text viewer to view the console log.

alerts log

Caché Monitor scans the console log at regular intervals for entries of the configured minimum severity and generates corresponding notifications, which it writes to the alerts log, *install-dir*\mgr\alerts.log, by default. Caché Monitor can be configured to send email notifications instead; see the "Using Caché Monitor" chapter of this guide for more information.

System Monitor log

Status messages about the functioning of Caché System Monitor (see the chapter "Using Caché System Monitor" in this guide) are written to the System Monitor log, *install-dir*\mgr\SystemMonitor.log.

The size of the SystemMonitor.log file is monitored by System Monitor. The file grows until it reaches the maximum size of 5 MB, at which point it is renamed to SystemMonitor.log.old, overwriting any existing SystemMonitor.log.old file, and a new SystemMonitor.log is created. The maximum number of megabytes used by the System Monitor log is therefore 10 MB.

If the System Monitor log is larger than 1 MB, only the most recent 1 MB portion is displayed by the Management Portal. Click the **Show entire file** link to display the entire file, which may require some time if the file is very large.

initialization log

The initialization log, cboot.log, contains information about the initialization of the Caché instance.

journal history log

The journal history log, journal.log, contains a list of all journal files maintained by the Caché instance and is used by all journal-related functions, utilities, and APIs to locate journal files. See the "Journaling" chapter of the *Caché Data Integrity Guide* for information about journaling.

These .log files are plain text files and can be viewed using any text editor or viewer.

To view the console log or the System Monitor log using the Management Portal, select **System Operation** > **System Logs** > **Console Log** or **System Operation** > **System Logs** > **System Monitor Log**.

1.4.2 Application and ODBC Error Logs

To view the application error log, select **System Operation** > **System Logs** > **Application Error Log**.

To view the xDBC error log, select **System Operation** > **System Logs** > **xDBC Error Log**.

1.4.3 Caché System Error Log

Caché sets aside a small portion of its shared memory to log items of interest. This table, which can contain important diagnostic information, is referred to by several different names, including the Caché system error log, errlog, SYSLOG, and the syslog table.

By default, the system error log contains the 500 most recent log items. For information about configuring the number of items in the system error log, see errlog in the "Advanced Memory Settings" section of the *Caché Additional Configuration Settings Reference*.

The view the system error log, choose one of the following methods:

- Open Terminal, enter **set $namespace = %SYS"** to switch to the %SYS namespace, and enter **do ^SYSLOG**. You can also enter **do FILTER^SYSLOG**, which has options to limit the output based on specific error codes or process ID.
- Run a diagnostic report, as described in the "Using the Caché Diagnostic Report" chapter of this guide.
- Run the **cstat** command with the **–e1** option, as described in Running cstat with Options in the "Monitoring Caché Using the cstat Utility" appendix of this guide.
- Run the **CacheHung** script, as described in CacheHung Script in the "Monitoring Caché Using the cstat Utility" appendix.
- Configure Caché to write the system error log to the console log during shutdown by taking the following steps; you can then review the console log as previously described.
  - Go to the **Compatibility Settings** page (**System Administration** -> **Configuration** -> **Additional Settings** -> **Compatibility**).
  - In the **ShutDownLogErrors** row, select **edit**.
  - On the **Edit ShutDownLogErrors** page, select the **ShutDownLogErrors** check box and click **Save**.

For a guide to using the Caché system error log, including sample output and a detailed description of the output fields, see SYSLOG - what it really is and what it means by a member of the InterSystems Product Specialist group on InterSystems Developer Community.

2

Using the Caché Diagnostic Report

InterSystems provides a mechanism to run a diagnostic report on your Caché instance and send the results to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems.

You configure and run the Diagnostic Report as a task from the Management Portal. The following sections describe the tasks, settings, and contents of the **Diagnostic Report** option:

- Running the Diagnostic Report Task
- Configuring Diagnostic Report Settings
- Diagnostic Report Contents

For more information on this task see the %SYS.Task.DiagnosticReport entry in the *InterSystems Class Reference*.

2.1 Running the Diagnostic Report Task

The most direct way to generate the report is by going to the **Diagnostic Report** page (**System Operation** > **Diagnostic Reports**) of the Management Portal and entering the appropriate information for the Diagnostic Report task. You can edit this information at any time by returning to this page. If you do not wish to edit any of the fields, click **Run** to generate the report using the current settings.

If you do not enter any information and click **Run**, the task generates a detailed report and places it in the manager's directory of the Caché instance (*install-dir*\mgr) as an HTML file. The file name is in *CustomerNameYYYYMMDDHHMM*.html format.

For example, on September 24, 2011 at 8:46 p.m., running the Diagnostic Report task with a license key issued to MyCompany on an instance installed in C:\MyCache generates a report file named:

```
C:\MyCache\mgr\MyCompany201109242046.html
```

There are several fields on the page you can set that affect when the task runs, where the file is saved, and whether or not to send the file to the WRC. The *Configuring Diagnostic Report Settings* section describes these settings. If you click **Close**, your changes are discarded and the report task does not run.

Viewing the Diagnostic Report Task History

Click **Task History** at the top of the **Diagnostic Report** page to display the history for the Diagnostic Report task. (See Using the Task Manager in the "Managing Caché" chapter of the *Caché System Administration Guide* for information about tasks and task history.).

2.2 Configuring Diagnostic Report Settings

The Caché installation contains a predefined on-demand Diagnostic Report task. The first time you go to the **Diagnostic Report** page, fill in the pertinent information to update the settings for this task. Depending on which fields you enter, you have the following choices of what to do with the Diagnostic Report:

1. To save the report to a specific archive directory other than the manager's directory, enter a directory name.
2. To send the report to the WRC, enter information in the outgoing mail fields.
3. To both save and send the report, enter the information from the two previous options.
4. To run the report automatically on a regular schedule, enable WRC HealthCheck.

The following list contains the settings for the Diagnostic Report and a description of each:

- **Directory for archived reports** — location to store the reports. Defaults to the manager's directory, *install-dir*\mgr, if you do not enter any information on the page. If you leave this setting blank and enter outgoing mail settings the report is not saved in the manager's directory. Click **Browse** to select an existing directory.

Information required to send the report directly to the WRC — if you enter the outgoing mail settings, the report is sent to *WRCHealthCheck@InterSystems.com*.

- **Existing WRC issue number** — WRC problem number (6 digits) related to this run of the Diagnostic Report. To enter a new problem, contact the WRC or enter your problem into WRC Direct.

  The task runs with the WRC issue number only once and then clears this setting.

- **Name of IP address of server for outgoing mail** — address of your outgoing SMTP (Simple Mail Transfer Protocol) mail server.
- **Username for authenticated SMTP** and **Password** — only required for SMTP authentication with the SMTP server. See RFC 2554 for details.
- **Address for the "From:" field in outgoing mail** — email address to appear in the sender field. Required if you enter SMTP server information; defaults to *DefaultDiagnosticReport@InterSystems.com*.
- **Address for the "Reply-To:" field in outgoing mail** — a valid email address at your company able to receive automated configuration messages from InterSystems.
- **Addresses for the "CC:" field in outgoing mail** — additional email addresses to receive the report.
- **Enable automatic WRC HealthCheck updates** — select this check box to send periodic reports to the WRC. InterSystems highly recommends that you enable the WRC HealthCheck feature. If selected, the Diagnostic Report task runs at regular intervals and sends the report to the WRC. These regular reports allow the WRC to better assist you. Selecting this feature requires you to enter the SMTP server information.

  Important: The report task does not send any private application information, and InterSystems keeps all configuration data strictly confidential.

- **Run the automatic WRC HealthCheck updates every** number of **days at this time** — if you enable WRC HealthCheck, the task manager saves the frequency (defaults to 7 days) and time (defaults to the Caché installation time) information for when to run the Diagnostic Report.

Additional information for the WRC:

- **Primary purpose of this instance** — choose whether you use this instance of Caché for development, testing, quality assurance, or production.

- **Any Ad Hoc content applied that is not in $ZV** — enter ad hoc content you have applied that does not appear in the *$ZVersion* special variable.
- **The type and number of CPUs present**
- **The total amount of physical memory** — enter the amount of physical memory on the machine.
- **Other details of the hardware this system uses**
- **Method used to back up this system (InterSystems, OS, External, other)** — enter the methods you use to back up your system.
- **Other relevant information about this instance** — enter any special notes you want to include with the report.

The Diagnostic Report task retains the information you enter in all but one of the settings; the task runs with the WRC issue number only once and then clears it. You cannot edit task settings while the report is running.

2.3 Diagnostic Report Contents

When the Diagnostic Report task runs, it creates an HTML log file containing both basic and advanced information, which is used by the WRC to resolve issues. The following sections describe the sections of the report:

- Basic Information
- Advanced Information

Note: On Microsoft Windows 32-bit systems the report uses the following third-party utilities developed by SysInternals Software:

- PsInfo.Exe — Displays extended system information
- PsList.Exe — Displays process information at the operating system level

2.3.1 Basic Information

The basic information includes the following categories:

General

Displays the following information:

- Full host name (with domain)
- IP address
- User name
- Date and time report was created
- Caché version string (*$ZVersion*)
- Caché objects version string
- Caché ODBC/JDBC server version information
- Caché Direct server version information
- Caché WebLink version information

- National Language Support (NLS) information
- Free block count information
- Operating system version (**uname -a** on UNIX® systems)
- Extended system information (only on Windows systems if the PsInfo.Exe utility is in the Caché Bin directory).

Key File

Displays active license information including the location of the license key file, the contents of the license key, and license availability (**$System.License.CKEY()** output).

License Counts

Displays license usage information (**$System.License.ShowCounts()** output).

%SS

Displays system status information (**^%SS** output — two snapshots taken thirty seconds apart).

Operating System Processes List

Displays operating system process information (only on Windows systems if the PsList.Exe utility is in the Caché Bin directory).

Spin Counts

Displays spin count information.

CPF File

Displays the contents of the active Caché configuration file (cache.cpf).

SysLog

Displays the contents of the Caché system error log; see Caché System Error Log in the "Monitoring Caché Using the Management Portal" chapter for more information.

Security

Displays a listing of the following security information:

- Security parameters
- Services
- Resources
- Roles
- Applications
- System users
- Current login failures
- Domains
- SSL configurations

Audit

Displays audit information including a listing of events and the contents of the audit log database.

Shadowing

Displays the contents of the shadowing globals for this instance both as a shadow source and a shadow destination.

cconsole

Displays the contents of the cconsole.log (if its size does not exceed 5MB).

Note: To produce a report that contains only the basic information:

1. Navigate to the **View Task Schedule** page (**System Operation** > **Task Manager** > **View Task Schedule**).
2. In the Diagnostic Report row, click **Details**.
3. On the **Task Details** page (**System Operation** > **Task Manager** > **View Task Schedule** > **Task Details**), click **Edit**.
4. On the **Task Scheduler Wizard** page, clear the **AdvancedReport** check box, and click **Finish**.
5. On the **Task Details** page (**System Operation** > **Task Manager** > **View Task Schedule** > **Task Details**), in the Diagnostic Report row, click **Run**.
6. On the **Run Task** page, click **Perform Action Now**.
7. Click **Close**.

2.3.2 Advanced Information

The advanced information includes the following categories:

cstat Snapshot #1

Displays output of the Caché statistics utility (**cstat**) run with the following options:

```
cstat -e2 -m-1 -n3 -j5 -g1 -m3 -L1 -u-1 -v1 -p-1 -c-1 -q1 -w2 -S-1 -E-1 -N65535 -s<mgr_dir>
```

For more information about the **cstat** utility, see the Monitoring Caché Using the cstat Utility appendix of this guide.

cstat Snapshot #2

Displays the output of the **cstat** utility run with the same options as the first snapshot one minute later.

If the **cstat** output files are too large, they are saved to a separate file and not sent with the report. If separate files were created, a message similar to the following is posted in the cstat section of the Diagnostic Report:

```
File /cache/cachetestsys/mgr/cstat201103151102.html is too big to be appended to
the Log File. A copy has been left in the Directory.
```

Although these files have an html extension, they are plain text and should be viewed in a text editor rather than a browser.

Network Status

Displays network information — output of the following utilities:

- **ipconfig /all** (only Windows systems)
- **netstat -an**

- **netstat -s**

Dump License

Displays local license table entries and key information (**$System.License.DumpLocalInUse()** and **$System.License.DumpKeys()** output).

Dump Files in Caché Manager's Directory

Displays a list of core or *.dmp files, if any.

GloStat

Displays global statistic information (**^GLOSTAT** output —ten snapshots taken every ten seconds).

3

Using Caché Monitor

Caché Monitor monitors the Caché instance's console log for errors and traps reported by Caché daemons and user processes and generates corresponding notifications, including email if configured. This chapter discusses the following topics:

- Caché System Monitoring Tools
- Caché Monitor Overview
- Using the ^MONMGR Utility
- Caché Monitor Errors and Traps

3.1 Caché System Monitoring Tools

Caché provides three sets of tools for general monitoring of Caché instances, as follows:

- The Management Portal provides several pages and log files that let you monitor a variety of system indicators, system performance, Caché locks, and errors and traps, as described in the "Monitoring Caché Using the Management Portal" chapter of this guide. Of these, the console log (*install-dir*\mgr\cconsole.log by default) is the most comprehensive, containing general messages, startup/shutdown, license, and network errors, certain operating system errors, and indicators of the success or failure of jobs started remotely from other systems, as well as alerts, warnings and messages from Caché System Monitor.
- Caché Monitor, as described in this chapter, generates notifications for console log entries of a configured minimum severity and either writes them to the alerts log or emails them to specified recipients. This allows console log alerts of all types to be extracted and brought to the attention of system operators.
- Caché System Monitor generates alerts and warnings related to important system status and resource usage indicators and also incorporates Caché Application Monitor and Caché System Health Monitor, which monitor system and user-defined metrics and generate alerts and warnings when abnormal values are encountered. System Monitor and Health Monitor alerts and warnings are written to the console log; Application Monitor alerts can be sent by email or passed to a specified notification method. System Monitor (including Application Monitor and Health Monitor) is managed using the **^%SYSMONMGR** utility. See the "Caché System Monitor" chapter of this guide for detailed information about using System Monitor, Application Monitor and Health Monitor.

3.2 Caché Monitor Overview

Caché Monitor scans the console log at regular intervals for entries of the configured severity level and generates corresponding notifications. These notifications are either written to the alerts log or sent by email to specified recipients.

Caché writes general messages, errors and traps, and the success or failure of jobs started remotely from other systems to the console log; see Monitoring Log Files in the "Monitoring Caché Using the Management Portal" chapter of this guide for more information. In addition, Caché System Monitor write alerts and warnings to the console log. By generating notifications based on console log contents, Caché Monitor bring alerts to the attention of system operators.

Note: Caché Monitor does not generate a notification for every console log entry of the configured severity. When there is a series of entries from a given process within less than about an hour of each other, a notification is generated for the first entry only. For this reason, you should immediately consult the console log (and view System Monitor alerts, if applicable) on receiving a single notification from Caché Monitor. However, the console log entries listed in Caché Monitor Errors and Traps always generate notifications.

Caché Monitor operates with the following settings by default:

- Caché Monitor is continuously running when the instance is running.
- The console log is scanned every 10 seconds.
- Notifications are generated for console log entries of severity 2 (severe) and 3 (fatal).
- Notifications are written to the alerts log.

  Note: You can view the alerts log in the Management Portal by navigating to the **System Logs** page (**System Operation** > **System Logs**) and selecting **System Monitor Log**, then using the **Browse** button to select the alerts.log file.

  The alerts log is not created until Caché Monitor writes its first notification to the log.

You can configure and manage Caché Monitor, including changing its default settings and configuring email notifications, using the interactive **^MONMGR** utility.

3.3 Using the ^MONMGR Utility

The Caché Monitor Manager (**^MONMGR**) utility must be executed in the %SYS namespace (the name is case-sensitive).

1. To start the Caché Monitor Manager, enter the following command in the Terminal:

   ```
   %SYS>do ^MONMGR
   ```

2. The main menu appears. Enter the number of your choice or press **Enter** to exit the Caché Monitor Manager:

   ```
   1) Start/Stop/Update MONITOR
   2) Manage MONITOR Options
   3) Exit

   Option?
   ```

The options in the main menu let you manage Caché Monitor as described in the following table:

Option	Description
`1) Start / Stop / Update Monitor`	Displays the Start/Stop/Update Monitor submenu which lets you manage Caché Monitor and the alerts log.
`2) Manage MONITOR Options`	Displays the Manage Monitor Options submenu which lets you manage Caché Monitor notification options (sampling interval, severity level, email).
`3) Exit`	Exits from the Caché Monitor Manager.

3.3.1 Start/Stop/Update Monitor

This submenu lets you manage the operation of the Caché Monitor Manager. Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 1

1) Update MONITOR
2) Halt MONITOR
3) Start MONITOR
4) Reset Alerts
5) Exit

Option?
```

The options in this submenu let you manage the operation of Caché Monitor as described in the following table:

Option	Description
`1) Update MONITOR`	Dynamically restarts Caché Monitor based on the current settings (interval, severity level, email) in Manage Monitor Options.
`2) Halt MONITOR`	Stops Caché Monitor. The console log is not scanned until Caché Monitor is started.
`3) Start MONITOR`	Starts Caché Monitor. The console log is monitored based on the current settings (interval, severity level, email) in Manage Monitor Options.
`4) Reset ALERTS`	Deletes the alerts log (if it exists).
`5) Exit`	Returns to the main menu.

3.3.2 Manage Monitor Options

This submenu lets you manage Caché Monitor's scanning and notification options. Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 2

1) Set Monitor Interval
2) Set Alert Level
3) Manage Email Options
4) Exit

Option?
```

The options in this submenu let you manage the operation of Caché Monitor as described in the following table:

Option	Description
`1) Set Monitor Interval`	Lets you change the interval at which the console log is scanned. InterSystems recommends an interval no longer than the default of 10 seconds.

Option	Description
`2) Set Alert Level`	Lets you set the severity level of console log entries generating notifications, as follows: • `1` – warning, severe and fatal • `2` – severe and fatal • `3` – fatal only
`3) Manage Email Options`	Lets you configure Caché Monitor email notifications using the Manage Email Options submenu.
`4) Exit`	Returns to the main menu.

Note: Because Caché Monitor generates a notification only for the first in a series of console log entries from a given process within about an hour, setting the alert level to 1 could mean that when a warning has generated an alerts log entry or email message, a subsequent severity 2 alert from the same process does not generate a notification. For example, a license expiration warning from Caché System Monitor could prevent a more serious shadow server disconnection alert 15 minutes later from generating an alerts log entry or email message.

3.3.3 Manage Email Options

The options in this submenu let you configure and enable/disable email. When email is enabled, Caché Monitor sends notifications by email; when it is disabled, notifications are written to the alerts log. Enter the number of your choice or press **Enter** to return to the Manage Monitor Options submenu:

```
Option? 3

1) Enable/Disable Email
2) Set Sender
3) Set Server
4) Manage Recipients
5) Set Authentication
6) Test Email
7) Exit

Option?
```

The options in this submenu let you manage the email notifications for Caché Monitor as described in the following table:

Option	Description
`1) Enable / Disable Email`	Enabling email causes Caché Monitor to: • send an email notification for each item currently in the alerts log, if any • delete the alerts.log file (if it exists) • send email notifications for console log entry of the configured severity from that point forward Disabling email causes Caché Monitor to write entries to the alerts log. **Note:** Enabling/disabling email does not affect other email settings; that is, it is not necessary to reconfigure email options when you enable/disable email.

Option	Description
`2) Set Sender`	Select this option to enter text that indicating the sender of the email, for example `Cache Monitor`. The text you enter does not have to represent a valid email account. You can set this field to NULL by entering `-` (dash).
`3) Set Server`	Select this menu item to enter the name and port number (default 25) of the email server that handles email for your site. Consult your IT staff to obtain this information. You can set this field to NULL by entering `-` (dash).
`4) Manage Recipients`	This option displays a submenu that lets you list, add, or remove the email addresses to which each notification is sent: **Note:** Each valid email address must be added individually; when you select `2) Add Recipient`, do not enter more than one address when responding to the `Email Address?` prompt.
`5) Set Authentication`	Lets you specify the authentication username and password if required by your email server. Consult your IT staff to obtain this information. If you do not provide entries, the authentication username and password are set to NULL. You can set the **User** field to NULL by entering `-` (dash).
`6) Test Email`	Sends a test message to the specified recipients using the specified email server.
`7) Exit`	Returns to the Manage Monitor Options submenu.

3.4 Caché Monitor Errors and Traps

The following console log errors always generate Caché Monitor notifications:

- Process halt due to segment violation (access violation).
- <FILEFULL>in database %
- AUDIT: ERROR: FAILED to change audit database to '%. Still auditing to '%.
- AUDIT: ERROR: FAILED to set audit database to '%.
- Sync failed during expansion of sfn #, new map not added
- Sync failed during expansion of sfn #, not all blocks added
- WRTDMN failed to allocate wdqlist...freezing system
- WRTDMN: CP has exited - freezing system
- Write Daemon encountered serious error - System Frozen
- Insufficient global buffers - WRTDMN in panic mode
- WRTDMN Panic: SFN x Block y written directly to database
- Unexpected Write Error: dkvolblk returned %d for block #%d in %
- Unexpected Write Error: dkswrite returned %d for block #%d in %
- Unexpected Write Error: %d for block #%d in %.
- Cluster crash - All Cache systems are suspended

- System is shutting down poorly, because there are open transactions, or ECP failed to preserve its state
- SERIOUS JOURNALING ERROR: JRNSTOP cannot open %.* Stopping journaling as cleanly as possible, but you should assume that some journaling data has been lost.
- Unable to allocate memory for journal translation table
- Journal file has reached its maximum size of %u bytes and automatic rollover has failed
- Write to journal file has failed
- Failed to open the latest journal file
- Sync of journal file failed
- Journaling will be disabled in %d seconds OR when journal buffers are completely filled, whichever comes first. To avoid potential loss of journal data, resolve the cause of the error (consult the Caché system error log, as described in Caché System Error Log in the "Monitoring Caché Using the Management Portal" chapter) or switch journaling to a new device.
- Error logging in journal
- Journaling Error x reading attributes after expansion
- ECP client daemon/connection is hung
- Cluster Failsoft failed, couldn't determine locksysid for failed system - all cluster systems are suspended
- enqpijstop failed, declaring a cluster crash
- enqpijchange failed, declaring a cluster crash
- Failure during WIJ processing - Declaring a crash
- Failure during PIJ processing - Declaring a crash
- Error reading block – recovery read error
- Error writing block – recovery write error
- WIJ expansion failure: System Frozen - The system has been frozen because WIJ expansion has failed for too long. If space is created for the WIJ, the system will resume otherwise you need to shut it down with cforce
- CP: Failed to create monitor for daemon termination
- CP: WRTDMN has been on pass %d for %d seconds - freezing system. System will resume if WRTDMN completes a pass
- WRTDMN: CP has died before we opened its handle - Freezing system
- WRTDMN: Error code %d getting handle for CP monitor - CP not being monitored
- WRTDMN: Control Process died with exit code %d - Freezing system
- CP: Daemon died with exit code %d - Freezing system
- Performing emergency Cache shutdown due to Operating System shutdown
- CP: All processes have died - freezing system
- cforce failed to terminate all processes
- Failed to start slave write daemon
- ENQDMN exiting due to reason #
- Becoming primary mirror server

4

Using Caché System Monitor

Caché System Monitor is a flexible, user-extensible utility used to monitor a Caché instance and generate notifications when the values of one or more of a wide range of metrics indicate a potential problem. As provided, System Monitor incorporates the following Caché instance monitoring tools:

- System Monitor monitors system status and resources, generating notifications (alerts and warnings) based on fixed parameters and tracking overall system health.
- Caché Health Monitor (Health Monitor) samples key system and user-defined metrics and compares them to user-configurable parameters and established normal values, generating notifications when samples exceed applicable thresholds.
- Caché Application Monitor (Application Monitor) samples significant system metrics, stores the values in the local namespace, and evaluates them using user-created alert definitions. When an alert is triggered, it can either generate an email notification or call a specified class method.

All three tools run in the %SYS namespace by default. System Monitor and Application Monitor can optionally be run in other namespaces under namespace-specific configurations and settings. You can define and configure your own components to extend the capabilities of System Monitor in each namespace as your needs require.

See Caché System Monitoring Tools in the "Using Caché Monitor" chapter of this guide for an overview of general Caché instance monitoring tools and Manage Email Options in that chapter for information about configuring Caché Monitor to generate email messages from notifications in the console log, including those generated by System Monitor. See Monitoring Log Files in the "Monitoring Caché Using the Management Portal" chapter for information about the log files discussed in this chapter.

This chapter contains the following sections:

4.1 Caché System Monitor

System Monitor samples important system status and resource usage indicators, such as the status of ECP connections and the percentage of the lock table in use, and generates notifications—alerts, warnings, and "status OK" messages—based on fixed statuses and thresholds. These notifications are written to the console log, allowing Caché Monitor to generate email messages from them if configured to do so. System Monitor also maintains a single overall system health state.

System Monitor is managed using the **^%SYSMONMGR** utility.

The remainder of this section discusses the following topics:

4.1.1 The System Monitor Process

In each namespace in which it is configured to run, System Monitor gathers and delivers system metric information in three stages using three types of classes (or System Monitor *components*) in sequence:

1. Obtain metric information

   Sensor classes incorporate methods for obtaining the values of system or application metrics. For example, the system sensor class SYS.Monitor.SystemSensors includes the **GetProcessCount()** method, which returns the number of active processes for the Caché instance, and the **GetLockTable()** method, which returns the percentage of the instance's lock table that is in use.

   At a fixed interval, System Monitor calls the **GetSensors()** method of each configured sensor class. A sensor class may do one of the following:

   - Return an array of sensor name/value pairs to be passed by System Monitor to subscriber classes (described in stage 2)
   - Evaluate the sensor values it obtains and return notifications to be posted by System monitor to notifier classes (described in stage 3)

   One of the sensor classes provided with System Monitor, SYS.Monitor.SystemSensors, returns a name/value array. The other, **%SYS.Monitor.AppMonSensor**, performs its own evaluations and generates its own notifications.

2. Evaluate metric information

   Subscriber classes incorporate methods for evaluating sensor values and generating notifications. After calling each sensor class that returns a name/value array, System Monitor calls the **Receive()** method of each subscriber class, populating the SensorReading property with the array. For each sensor name/value pair provided to its **Receive()** method, the subscriber class evaluates the value and if appropriate returns a notification containing text and a severity code.

   For example, when System Monitor passes the name/value array returned from SYS.Monitor.SystemSensors.GetSensors() to subscriber classes,

   - the system subscriber, SYS.Monitor.SystemSubscriber, may discover that the **LockTablePercentFull** value is over 85, its warning threshold for that sensor, and generate a containing with a severity code of 1 and appropriate text.
   - the Health Monitor subscriber, SYS.Monitor.Health.Control, may determine that the **ProcessCount** value is too high, based on that sensor's configured parameters and established normal values, and return a notification containing a severity code of 2 and appropriate text.

3. Generate notifications

Notifier classes incorporate methods for passing notifications to one or more alerting systems. After calling each sensor class and subscriber class, System Monitor calls the **Post()** method of each notifier class, populating the Notifications property with the notifications returned by sensor or subscriber classes. The notifier class then passes each notification to the desired alerting method; for example, when the system notifier receives the notifications returned by the system subscriber for **LockTablePercentFull** and the Health Monitor subscriber for **ProcessCount**, it writes the severity code and text to the console log. This approach allows notifications to be passed to independent alerting systems such as those in Ensemble and TrakCare, as well as user-defined alerting systems.

System Monitor starts automatically when the instance starts and begins calling the configured sensor classes in each of the configured startup namespaces, passing sensor values to configured subscriber classes and notifications to configured notifier classes in turn. You can define and configure your own System Monitor sensor, subscriber and notifier classes on a per-namespace basis.

Note: In an emergency, System Monitor may need to be shut down. The classmethod **%SYS.Monitor.Enabled([***flag***])** sets, clears, and reports the status of System Monitor. If *flag* is **0**, System Monitor will not start.

4.1.2 Tracking System Monitor Notifications

Typically, any System Monitor alert (notification of severity 2) or sequence of System Monitor warnings (severity 1) should be investigated. Health Monitor can also generate System Monitor alerts and warnings

System Monitor alerts and warnings, including those generated by Health Monitor, and System Monitor status messages (severity 0) are written to the console log (*install-dir*\mgr\cconsole.log). (All System Monitor and Health Monitor status messages are written to the System Monitor log, *install-dir*\mgr\SystemMonitor.log. Application Monitor alerts are not written to logs, but can be sent by email or passed to a specified notification method.)

To track System Monitor alerts and warnings, you can do the following:

- View System Monitor alerts using the **^%SYSMONMGR** utility. This option lets you display alerts for all sensors or for a specific sensor and view all recorded alerts or only those occurring during a specified time period, but it does not display warnings.
- Monitor the console log (see Monitoring Log Files in the "Monitoring Caché Using the Management Portal" chapter). Bear in mind that when a sequence of System Monitor alerts is generated for a given sensor within a short period of time, only the first is written to the console log.

  Note: In the console log, System Monitor status notifications are labeled with initial capitals, for example `[System Monitor] started in %SYS`, whereas warnings, alerts and OK messages are labeled in uppercase, such as `[SYSTEM MONITOR] CPUusage Warning: CPUusage = 90 ( Warnvalue is 85)`.

- Configure Caché Monitor to send email notifications of alerts (and optionally warnings) appearing in the console log (instead of writing them to the alerts log, the default). When relying on this method, keep in mind that Caché Monitor does not generate a notification for every console log entry of the configured severity; when there is a series of entries from a given process (such as System Monitor) within about one hour, a notification is generated for the first entry only. For example, if a network problem causes System Monitor alerts concerning ECP connections, open transactions, and shadow server connection to be generated over a 15 minute period, Caché Monitor generates only one notification (for whichever alert was first). For this reason, on receiving a single System Monitor notification from Caché Monitor you should immediately view System Monitor alerts and consult the console log.

4.1.3 System Monitor Status and Resource Metrics

The following table lists the system status and resource usage metrics sampled by System Monitor and the notification thresholds and rules for each that result in warnings (severity 1), alerts (severity 2), and "status OK" severity 0 notifications.

Table 4–1: System Monitor Status and Resource Notifications

Metric	Description	Notification Rules
Disk Space	Available space in a database directory	• < 250MB — warning • < 50MB — alert • > 250 after warning/alert — OK
Journal Space	Available space in the journal directory	• < 250MB — warning • < 50MB — alert • > 250 after warning/alert — OK
Paging	Percentage of physical memory and paging space used	• paging space > 30% — warning • physical memory > 96% + paging space > 50% — alert
Lock Table	Percentage of the lock table in use	• > 85% — warning • > 95% — alert • < 85% after warning/alert — OK
Write Daemon	Status of the write daemon	• write daemon is awake and processing its (non-empty) queue but has been on one cycle at least 10 seconds longer than the configured write daemon cycle time (default 80 seconds) — alert • write daemon completes a pass after alert — OK
ECP Connections	State of connections to ECP application servers or ECP data servers	• state is *Trouble* for at least five (5) seconds — alert
Shadow Server	Status of connection to shadow sources	• trouble — warning • disconnected — alert
Shared Memory Heap (Generic Memory Heap)	Status of shared memory heap (SMH), also known as generic memory heap (gmheap)	• SMH (gmheap) status 1 — warning • SMH (gmheap) status 2 — alert
Open Transactions	Duration of longest open local or remote (ECP) transactions	• > 10 minutes — warning • > 20 minutes — alert
License Expiration	Days until license expires	• 7 days — warning • 5 days or fewer — alert (daily)

Metric	Description	Notification Rules
SSL/TLS Certificate Expiration	Days until certificate expires	• individual certificate expires within 30 days — warning (repeated daily) • one or more daily expiring certificate warnings — alert (summary of warnings, one per day)
ISCAgent (mirror members only)	ISCAgent status	• Unresponsive for <1 minute — warning • Unresponsive for >1 minute — alert

4.1.4 System Monitor Health State

Based on notifications posted to the console log (see Monitoring Log Files in the "Monitoring Caché Using the Management Portal" chapter of this guide), including both system alerts generated directly by the Caché instance and alerts and warnings generated by System Monitor and its Health Monitor component, System Monitor maintains a single value summarizing overall system health in a register in shared memory.

At startup, the system health state is set based on the number of system (not System Monitor) alerts posted to the console log during the startup process. Once System Monitor is running, the health state can be elevated by either system alerts or System Monitor alerts or warnings. Status is cleared to the next lower level when 30 minutes have elapsed since the last system alert or System Monitor alert or warning was posted. The following table shows how the system health state is determined.

Table 4–2: System Monitor Health State

State	Set at startup when ...	Set following startup when ...	Cleared to ...
GREEN/ok(0)	no system alerts are posted during startup	30 minutes (if state was **YELLOW**) or 60 minutes (if state was **RED**) have elapsed since the last system alert or System Monitor alert or warning was posted	n/a
YELLOW/warning(1)	up to four system alerts are posted during startup	state is **GREEN** and • one system alert is posted OR • one or more System Monitor alerts and/or warnings are posted, but not alerts sufficient to set **RED**, as below	**GREEN** when 30 minutes have elapsed since the last system alert or System Monitor alert or warning was posted
RED/alert(2)	five or more system alerts are posted during startup	• state is **YELLOW** and one system alert is posted OR • state is **GREEN** or **YELLOW** and during a 30 minute period, System Monitor alerts from at least five different sensors or three System Monitor alerts from a single sensor are posted	**YELLOW** when 30 minutes have elapsed since the last system alert or System Monitor alert or warning was posted

Note: A fourth state, **HUNG**, can occur when global updates are blocked. Specifically, the following events change the state to **HUNG**:

- The journal daemon is paused for more than 5 seconds or frozen (see Journal I/O Errors in the "Journaling" chapter of the *Caché Data Integrity Guide*).
- Any of switches 10, 11, 13, or 14 are set (see Using Switches in the "Managing Caché Remotely" chapter of *Caché Specialized System Tools and Utilities*).
- The write daemon is stopped for any reason or sets the **updates locked** flag for more than 3 seconds.
- The number of available buffers falls into the critical region and remains there for more than 5 seconds.

When the health state changes to **HUNG**, the reason is written to the console log.

The System Monitor health state can be viewed using

- the **View System Health** option on the View System Data menu of **^%SYSMONMGR** (does not report **HUNG**)
- the $SYSTEM.Monitor API, which lets you access the system status directly. Use **$SYSTEM.Monitor.State()** to return the system status; see also the **SetState**, **Clear**, **Alert**, and **ClearAlerts** methods.
- the **ccontrol list** and **ccontrol qlist** commands (do not include health state on Windows)

Note: When System Monitor is not running, the System Monitor health state is always **GREEN**.

4.1.5 System Monitor Defaults

System Monitor calls a provided set of classes that can be augmented, runs in the %SYS namespace, and operates under three default settings that can be changed.

4.1.5.1 Default System Monitor Components

Five classes are provided with Caché and configured in System Monitor in the %SYS namespace by default.

Sensor classes:

- SYS.Monitor.SystemSensors

  System sensor class obtaining sensor values to be passed to configured subscriber classes, including the System Monitor subscriber (SYS.Monitor.SystemSubscriber) and Caché Health Monitor subscriber (SYS.Monitor.Health.Control).

- **%SYS.Monitor.AppMonSensor**

  Class providing sensor, subscriber and notification services for Caché Application Monitor; obtains sensor values and stores them in the local namespace, evaluates the values based on user-defined alerts and either generates an email message or calls a user-specified method when an alert is triggered, based on the alert definition.

Subscriber classes:

- SYS.Monitor.Health.Control

  Subscriber class for Health Monitor; receives and evaluates statistical sensor values from SYS.Monitor.SystemSensors and posts notifications to the system notifier.

- SYS.Monitor.SystemSubscriber

  System Monitor subscriber available to all sensor classes; contains all code required to monitor and analyze the sensors in SYS.Monitor.SystemSensors. Generates System Monitor notifications and Health Monitor notifications for some sensors.

Notifier class:

- SYS.Monitor.SystemNotify

  System notifier available to all subscriber classes. On receiving a notification from the system subscriber (SYS.Monitor.SystemSubscriber) or Health Monitor subscriber (SYS.Monitor.Health.Control), writes it to the System Monitor log, and to the console log if it is of severity 2 (alert). (See the chapter "Monitoring Caché Using the Management Portal" in this guide for information on these log files.)

  The system notifier also generates a single overall evaluation of system status that can be obtained using the **SYS.Monitor.State()** method, which returns `0` (OK), `1` (warning), or `2` (alert).

User-defined classes can be configured using **^%SYSMONMGR**.

4.1.5.2 Default System Monitor Namespace

All System Monitor and Application Monitor configurations and settings are namespace-specific. By default, System Monitor starts and runs only in the %SYS namespace. Additional startup namespaces for System Monitor and Application Monitor can be configured using **^%SYSMONMGR**. Following any change you make to the System Monitor or Application Monitor configuration for a namespace, you must restart System Monitor in the namespace for the change to take effect.

Health Monitor runs only in the %SYS namespace.

4.1.5.3 Default System Monitor Settings

By default, the System Monitor is always running when the instance is running; it can be stopped using **^%SYSMONMGR** but will start automatically again when the instance next starts.

By default, the System Monitor

- calls the **GetSensors()** method of each configured sensor class every 30 seconds
- writes only alerts, warnings and messages to the System Monitor log, and does not write sensor readings
- does not save sensor readings

These settings can be changed using **^%SYSMONMGR**.

4.1.6 Using the ^%SYSMONMGR Utility

The **^%SYSMONMGR** utility lets you manage and configure the System Monitor. The utility can be executed in any namespace, and changes made with it affect only the namespace in which it is started. You must maintain a separate System Monitor configuration for each startup namespace you configure by starting **^%SYSMONMGR** in that namespace. Following any change you make to the System Monitor configuration for a namespace, you must restart System Monitor in the namespace for the change to take effect.

To manage the System Monitor, enter the following command in the Terminal:

```
%SYS>do ^%SYSMONMGR
```

The main menu appears.

```
1) Start/Stop System Monitor
2) Set System Monitor Options
3) Configure System Monitor Classes
4) View System Monitor State
5) Manage Application Monitor
6) Manage Health Monitor
7) View System Data
8) Exit

Option?
```

Enter the number of your choice or press **Enter** to exit the utility.

The options in the main menu let you perform System Monitor tasks as described in the following table:

Option	Description
1) Start/Stop System Monitor	• Start System Monitor • Stop System Monitor
2) Set System Monitor Options	• Set the sampling interval for configured sensor classes • Set the debugging level of information written to the System Monitor log • Enable saving of sensor readings and set number of days to save • Return sampling interval, debugging level, and sensor reading saving to their defaults
3) Configure System Monitor Components	• Configure or remove user-defined sensor, subscriber and notifier classes • Configure startup namespaces
4) View System Monitor State	• Display the operating status of System Monitor and its configured components
5) Manage Application Monitor	• Display the Application Monitor submenu
6) Manage Health Monitor	• Display the Health Monitor submenu (available only if **^%SYSMONMGR** is run in the %SYS namespace)
7) View System Data	• View saved sensor readings • View the System Monitor health state • View past or current System Monitor alerts

4.1.6.1 Start/Stop System Monitor

When a Caché instance starts, System Monitor starts automatically and begins calling configured classes in each configured startup namespace; this cannot be changed. While the instance is running, however, you can stop System Monitor, and must do so in order to change the configuration of Caché Health Monitor. In addition, following any change you make to the System Monitor configuration for a namespace, you must restart System Monitor in the namespace for the change to take effect.

When you enter 1 at the main menu, the following menu is displayed:

```
1) Start System Monitor
2) Stop System Monitor
3) Exit
```

Enter 2 to stop System Monitor when it is running, and 1 to start it when it is stopped.

Note: System Monitor monitors the size of the console log and rolls it over when required, thereby limiting the space it uses to twice the MaxConsoleLogSize configuration setting, which is 5 MB by default. When System Monitor is stopped, therefore, the console log may grow beyond this limit until the instance is restarted or the PurgeErrorsAndLogs task is run. See Monitoring Log Files in the "Monitoring Caché Using the Management Portal" chapter for information about the console log.

4.1.6.2 Set System Monitor Options

To change global System Monitor settings or to return them to their default values, stop System Monitor if it is running and then enter 2 at the main menu:

```
1) Set Sample Interval
2) Set Debugging Level
3) Reset Defaults
4) Manage Debug Data
5) Exit
```

Enter 1 to set the interval at which System Monitor calls each configured sensor class; the default is 30 seconds.

Enter 2 to set the debugging level. The default is `0 (base)` which writes System Monitor and Health monitor status and error messages to the System Monitor log, and does not save sensor readings. Debugging level `1 (log all sensors)` writes sensor readings to the System Monitor log along with messages and saves sensor readings, which can then be viewed using the View Sensor Data option of the View System Data menu.

Enter 3 to reset the sample interval, debugging level, and saving of sensor readings to their default values.

Enter 4 to set the number of days for which sensor readings are saved; the default is 5.

Your changes take effect when you next start or restart System Monitor.

4.1.6.3 Configure System Monitor Components

As described in Caché System Monitor, you can create your own sensor, subscriber and notifier classes by extending %SYS.Monitor.AbstractSensor, %SYS.Monitor.AbstractSubscriber, and %SYS.Monitor.AbstractNotification, respectively, and configure them in System Monitor to extend the capabilities of the provided classes described in Default System Monitor Components. You can also add namespaces other than %SYS for System Monitor to start and run in.

Configure System Monitor Classes

When you enter 3 at the main menu, the following menu is displayed:

```
1) Configure Components
2) Configure Startup Namespaces
3) Exit
```

Enter 1 to display the options for configuring classes:

```
1) List Classes
2) Add Class
3) Delete Class
4) Exit
```

Enter 1 to list the currently configured classes for the namespace in which you started **^%SYSMONMGR**, including provided system classes and those you have configured.

Enter 2 to configure a user-defined class for the namespace in which you started **^%SYSMONMGR**. The class you specify must exist in the namespace and be recognized by System Monitor as a valid sensor, subscriber or notifier class. You can also enter a description of the class.

Enter 3 to delete a user-defined class you have configured.

Note: Configuring or deleting a class affects only the namespace in which you started **^%SYSMONMGR**.

Configure System Monitor Namespaces

When an instance starts, System Monitor automatically starts as a separate process in each configured startup namespace (by default %SYS only). All System Monitor configurations and settings are namespace-specific. When you make changes using **^%SYSMONMGR**, the changes *affect only the namespace in which you started the utility*.

Note: All instances of **^%SYSMONMGR** write messages to the same System Monitor log. Startup namespaces can be configured from any namespace.

When you enter 3 at the main menu, the following menu is displayed:

```
1) Configure Components
2) Configure Startup Namespaces
3) Exit
```

Enter 2 to display the options for configuring namespaces:

```
1) List Startup Namespaces
2) Add Namespace
3) Delete Namespace
4) Exit
```

Enter 1 to list the currently configured startup namespaces.

Enter 2 to add a startup namespace.

Enter 3 to delete a startup namespace. (You cannot delete %SYS.)

4.1.6.4 View System Monitor State

Enter 4 at the main menu to display the status of System Monitor and its components in the namespace in which you started **^%SYSMONMGR**, for example:

```
       Component                  State
System Monitor                    OK
%SYS.Monitor.AppMonSensor         None
SYS.Monitor.SystemSensors         OK
SYS.Monitor.Health.Control        Running: Period is Thursday 09:00 - 11:30
SYS.Monitor.SystemSubscriber      OK
SYS.Monitor.SystemNotifier        OK
```

In this example, System Monitor and its system sensor, subscriber and notifier classes are running normally, as is Health Monitor's subscriber class. However, none of Application Monitor's classes are activated (see Manage Monitor Classes), so it is not evaluating sensor samples or generating alerts.

4.1.6.5 Manage Application Monitor

See Using ^%SYSMONMGR to Manage Application Monitor.

4.1.6.6 Manage Health Monitor

See Using ^%SYSMONMGR to Manage Health Monitor.

4.1.6.7 View System Data

Enter 7 at the main menu to display options for viewing System Monitor information about the system.

```
1) View Sensor Data
2) View System Health
3) View Alerts
4) Exit
```

Enter 1 to view saved sensor readings, if you have enabled saving of sensor data using the **Manage Sensor Readings** option on the Set System Monitor Options menu. You can display saved readings for all sensors or for a specific sensor, and you can view all saved sensor readings or only those for a time period you specify.

Enter 2 to view the System Monitor health state, including all alerts between the previous **GREEN** state and the current state, if not **GREEN**.

Enter 3 to view System Monitor alerts. You can display alerts for all sensors or for a specific sensor, and you can view all alerts within the period you specified using the **Manage Sensor Readings** option on the Set System Monitor Options menu, or only those for a time period you specify.

4.1.7 Defining System Monitor Components

The SYS.Monitor API lets define your own sensor, subscriber, and notifier classes.

4.1.7.1 Sensor Classes

Sensor classes extend %SYS.Monitor.AbstractSensor. The System Monitor controller initially calls each sensor class's **Start()** method; thereafter, on each cycle, it calls the **GetSensors()** method. The **SetSensor()** method is used within the sensor class to set sensor name/value pairs in the SensorReading property, which is returned by **GetSensors()** and passed to all subscriber classes.

A sensor class may also evaluate sensor readings and, as a result of its evaluation, call the %SYS.Monitor.Email class for generating email messages from notifications or any user-defined alerting method.

4.1.7.2 Subscriber Classes

Subscriber classes extend %SYS.Monitor.AbstractSubscriber. The System Monitor controller initially calls each subscriber class's **Start()** method; thereafter, on each cycle, it calls the **Receive()** method once for each sensor class called in the cycle, passing the SensorReading property with the sensor name/value pairs received from that sensor class. The subscriber class may evaluate one or more of the name/value pairs and set notifications using the **Notify()** method, which populates the Notifications property.

A subscriber class may also, as a result of its sensor evaluation, call the %SYS.Monitor.Email class for generating email messages from notifications, or any user-defined alerting method.

%SYS.Monitor.SampleSubscriber is provided as a sample subscriber class.

4.1.7.3 Notifier Classes

Notifier classes extend %SYS.Monitor.AbstractNotification. The System Monitor controller initially calls each notifier class's **Start()** method; thereafter, on each cycle, it calls the **Post()** method once for each subscriber class called in the cycle, passing the Notifications property with the notifications received from that subscriber. The notifier class calls then passes the notifications to its alerting method(s), which may include the %SYS.Monitor.Email class for generating email messages from notifications or any user-defined alerting method.

4.2 Caché Health Monitor

Caché Health Monitor monitors a running Caché instance by sampling the values of a broad set of key metrics during specific periods and comparing them to configured parameters for the metric and established normal values for those periods; if sampled values are too high, Health Monitor generates an alert (notification of severity 2) or warning (severity 1). For example, if CPU usage values sampled by Health Monitor at 10:15 AM on a Monday are too high based on the configured maximum value for CPU usage or normal CPU usage samples taken during the Monday 9:00 AM to 11:30 AM period, Health Monitor generates a notification.

This section covers the following topics:

- Caché Health Monitor Overview
- Using ^%SYSMONMGR to Manage Health Monitor

4.2.1 Caché Health Monitor Overview

Health Monitor uses a fixed set of rules to evaluate sampled values and identify those that are abnormally high. This design is based on the approach to monitoring manufacturing processes described in the "Process or Product Monitoring and Control" section of the *NIST/SEMATECH e-Handbook of Statistical Methods*, with deviation from normal values determined using rules based on the WECO statistical probability rules (Western Electric Rules), both adapted specifically for Caché monitoring purposes.

Health Monitor alerts (severity 2) and warnings (severity 1) are written to the console log (*install-dir*\mgr\cconsole.log). See Tracking System Monitor Notifications for information about ways to make sure you are aware of these notifications.

Health Monitor status messages (severity 0) are written to the System Monitor log (*install-dir*\mgr\SystemMonitor.log).

Note: Unlike System Monitor and Application Monitor, Health Monitor runs only in the %SYS namespace.

This section contains the following subsections:

- Health Monitor Process Description
- Health Monitor Elements and Extensions

4.2.1.1 Health Monitor Process Description

By default, Health Monitor does not start automatically when the instance starts; for this to happen, you must enable Health Monitor within Caché System Monitor using the **^%SYSMONMGR** utility. (You can specify an interval to wait after Caché starts before starting Health Monitor when it is enabled, allowing the instance to reach normal operating conditions before sampling begins.) You can always use the utility to see the current status of Health Monitor. For more information, see Using ^%SYSMONMGR to Manage Health Monitor, later in this chapter.

The basic elements of the Health Monitor process are described in the following:

- Health Monitor samples 41 system sensors defined in SYS.Monitor.SystemSensors (see Default System Monitor Components).

  Some sensors represent an overall metric for the Caché instance; for example, the **LicensePercentUsed** sensor samples the percentage of the instance's authorized license units that are currently in use, while the **JournalGrowthRate** sensor samples the amount of data (in KB per minute) written to the instance's journal files.

  Other sensors apply to a particular *sensor item*, such as a database or mirror; for example, **DBLatency** sensors sample the time (in milliseconds) required to complete a random read on each mounted database, while **DBReads** sensors sample the number of reads per minute from each mounted database (the databases are specified by directory).

- Each sensor is represented by a *sensor object* within Health Monitor that sets at least one and possibly three parameters, as follows:

  – a required base (minimum) value for sensor samples

  – optionally, either a maximum value and warning value, or a multiplier and warning multiplier

  For example, by default the **DBLatency** sensor object specifies a base of **1000**, a maximum value of **3000**, and a warning value of **1000**, while the **DBReads** sensor object specifies a base of **1024**, a multiplier of **2**, and a warning multiplier of **1.6**.

- Each sensor is sampled every 30 seconds during specified weekly, monthly, quarterly or yearly periods; samples below the base specified by the sensor object are discarded.

  By default there are 63 weekly periods each of which represents one of nine specified intervals during a particular day of the week, for example 9:00 AM to 11:30 AM on Mondays, but you can configure your own periods.

- To evaluate sensor samples, Health Monitor uses the sensor object parameters and, if necessary, a *chart* for each sensor during each period, containing previously collected samples and their mean value and the standard deviation from the mean, or *sigma*.

  If a sensor object has maximum and warning values set, a chart is not required to evaluate samples for sensors using that object, because notifications are generated by comparing samples to those values (see Notification Rules). Under the default settings, therefore, charts are not required for **DBLatency** sensors.

  For each sensor object that instead has multiplier values set, a chart is required. Under the default settings, therefore, charts are required for **DBReads** sensors. The chart for the **DBReads c:\InterSystems\Cache\mgr\docbook** sensor during the Monday 09:00 - 11:30 period, for example, might indicate the mean reads per minute from the **DOCBOOK** database during this period to be **2145**, with a sigma of **141** and a highest single value of **2327**.

  If a chart for a sensor during a particular period is required but does not yet exist, it must be generated before samples taken during that period can be evaluated. When Health Monitor is active, therefore, each sensor is in one of two modes during any given period:

  – If a chart is required but does not exist, that sensor is automatically in *analysis mode*.

    In analysis mode, Health Monitor simply records the samples it collects, then at the end of the period generates the required chart for the sensor. To ensure that the chart is reliable, a minimum of 13 samples must have been taken in analysis mode. Until 13 valid samples are taken within a single recurrence of a period, the sensor remains in analysis mode and no chart is generated for that period

    Note: Charts should always be generated from samples taken during normal, stable operation of the Caché instance. For example, when a Monday 09:00 - 11:30 chart does not exist, it should not be generated on a Monday holiday or while a technical problem is affecting the operation of the Caché instance.

  – If a required chart exists, or no chart is required, that sensor is in *monitoring mode*.

    In monitoring mode, Health Monitor collects samples to evaluate against the values in the sensor object or the existing chart. To ensure that notifications are not triggered by transient abnormal samples, every six sample values are averaged together to generate one reading every three minutes, and it is these readings that are evaluated.

- Sensor readings are evaluated by the appropriate subscriber class (see The System Monitor Process). When a sequence of readings meets the criteria for a notification when compared to the sensor object settings and the appropriate chart (if required), the subscriber class generates an alert or a warning by passing a notification containing text and a severity code to the system notifier, SYS.Monitor.SystemNotify.

  Specifically, when three (3) consecutive readings exceed the maximum value for the sensor object, an alert (notification of severity 2) is generated; when five (5) consecutive readings exceed the warning value for the sensor object, a warning (notification of severity 1) is generated. Complete information about how the maximum and warning values are determined for each sensor object appears in the Notification Rules section, but examples are as follows:

  – The **DBLatency** sensor object has maximum and warning values set by default. Therefore, for the **DBLatency c:\InterSystems\Cache\mgr\docbook** sensor during the Monday 09:00 - 11:30 period, an alert is generated if three consecutive readings are greater than the sensor object maximum value (**3000** by default).

  – The **DBReads** sensor object, on the other hand, has multipliers set by default, which means the maximum value is the multiplier times the greater of:

    - the mean in the chart plus three times the sigma in the chart
    - the highest value in the chart plus one sigma

    So for the **DBReads c:\InterSystems\Cache\mgr\docbook** sensor during the same period, an alert is generated if three consecutive readings are greater than **5136**—the default sensor object multiplier of **2** times **2568 (**the chart mean of **2145** plus three times the sigma of **141**), which is greater than **2468** (the high chart value of **2327** plus one sigma).

– If the **DBReads** sensor object were edited to remove the multipliers, leaving it with only a base, an alert would be generated for **DBReads c:\InterSystems\Cache\mgr\docbook** if three consecutive readings were greater than **2568**, which is the greater of

  • the mean in the chart plus three times the sigma in the chart

  • the highest value in the chart (**2327**)

Note: Because no chart is required to evaluate readings from sensors whose sensor objects have maximum and warning values specified, evaluation of these sensor readings and posting of any resulting notifications is handled by the SYS.Monitor.SystemSubscriber subscriber class, rather than the SYS.Monitor.Health.Control subscriber class (see Default System Monitor Components). As a result, notifications for these sensors are generated even when Health Monitor is not enabled (see Using ^%SYSMONMGR to Manage Health Monitor), as long as System Monitor is running.

If you want to generate notifications using values for some sensors represented by a given sensor object but using multipliers for others—for example, using values for **DBLatency** sensors for some databases but multipliers for others—you can do so by setting multipliers in the sensor object and manually creating charts for those for which you want to use absolute values; see Charts for more information.

- When a period has recurred five times since a chart was generated for a sensor or sensor/item during that period, not including those during which an alert was generated, the readings from these five normal period recurrences are evaluated to detect a rising or shifted mean for the sensor. If the mean is rising or has shifted with 95% certainty, the chart is recalibrated—the existing chart for the sensor during that period is replaced with a chart generated from the samples taken during the most recent recurrence of the period. For example, if the number of users accessing a database is growing slowly but steadily, the mean **DBReads** value for that database is likely to also rise slowly but steadily, resulting in regular chart recalibration every five periods, which avoids unwarranted alerts.

  Note that sensor object maximum and multiplier values cannot be automatically recalibrated in the same way, and should be adjusted manually because automatic chart recalibration does not apply to such sensors. For example, if the number of users accessing a database grows, the base, maximum value, and warning value for the **DBLatency** sensor object may require manual adjustment.

4.2.1.2 Health Monitor Elements and Extensions

Health Monitor is provided with a set of default elements that you can reconfigure and extend in various ways, as described in the following subsections:

- Sensors and Sensor Objects
- Notification Rules
- Periods
- Charts

Sensors and Sensor Objects

A Health Monitor sensor object represents one of the sensors in SYS.Monitor.SystemSensors. Each sensor object must provide a base value, and can optionally provide either a maximum value and warning value, or a multiplier and a warning multiplier; see Health Monitor Process Description and Notification Rules for information about how these values are used in evaluating sensor readings. The Health Monitor sensor objects are shown with their default parameters in the following table.

Where a sensor item is shown, the sensor object represents multiple sensors, one for each applicable item (job type, CSP server, database, or mirror); where there is no sensor item listed, the sensor object represents just one instance-wide sensor.

Sensor objects can be listed and edited (but not deleted) using the **^%SYSMONMGR** utility as described in Configure Health Monitor Classes. Editing a sensor object allows you to modify one or all of its values. You can enter a base, maximum value, and warning value; a base, multiplier, and warning multiplier; or a base only.

Table 4–3: Caché Health Monitor Sensor Objects

Sensor Object	Sensor Item	Description	Base	Max	Mult	Warn	Warn Mult
CPUUsage		System CPU usage (percent).	50	85	—	75	—
CSPSessions	*IP_address:port*	Number of active CSP sessions on the listed CSP gateway server.	100	—	2	—	1.6
CSPActivity	*IP_address:port*	Requests per minute to the listed CSP gateway server.	100	—	2	—	1.6
CSPActualConnections	*IP_address:port*	Number of connections created on the listed CSP gateway server.	100	—	2	—	1.6
CSPInUseConnections	*IP_address:port*	Number of currently active connections to the listed CSP gateway server.	100	—	2	—	1.6
CSPPrivateConnections	*IP_address:port*	Number of private connections to the listed CSP gateway server.	100	—	2	—	1.6
CSPUrlLatency	*IP_address:port*	Time (milliseconds) required to obtain a response from *IP_address:port*/csp/sys/UtilHome.csp.	1000	5000	—	3000	—
CSPGatewayLatency	*IP_address:port*	Time (milliseconds) required to obtain a response from the listed CSP gateway server when fetching the metrics represented by the CSP sensor objects.	1000	2000	—	1000	—
DBLatency	*database_directory*	Milliseconds to complete a random read from the listed mounted database.	1000	3000	—	1000	—
DBReads	*database_directory*	Reads per minute from the listed mounted database.	1024	—	2	—	1.6
DBWrites	*database_directory*	Writes per minute to the listed mounted database.	1024	—	2	—	1.6
ECPAppServerKBPerMinute		KB per minute sent to the ECP data server.	1024	—	2	—	1.6
ECPConnections		Number of active ECP connections.	100	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max	Mult	Warn	Warn Mult
ECPDataServerKBPer-Minute		KB per minute received as ECP data server.	1024	—	2	—	1.6
ECPLatency		Network latency (milliseconds) between the ECP data server and this ECP application server.	1000	3000	—	3000	—
ECPTransOpenCount		Number of open ECP transactions	100	—	2	—	1.6
ECPTransOpenSecsMax		Duration (seconds) of longest currently open ECP transaction	60	—	2	—	1.6
GlobalRefsPerMin		Global references per minute.	1024	—	2	—	1.6
GlobalSetKillPerMin		Global sets/kills per minute.	1024	—	2	—	1.6
JournalEntriesPerMin		Number of journal entries written per minute.	1024	—	2	—	1.6
JournalGrowthRate		Number of KB per minute written to journal files.	1024	—	2	—	1.6
LicensePercentUsed		Percentage of authorized license units currently in use.	50	—	1.5	—	—
LicenseUsedRate		License acquisitions per minute.	20	—	1.5	—	—
LockTablePercentFull		Percentage of the lock table in use.	50	99	—	85	—
LogicalBlockRe-questsPerMin		Number of logical block requests per minute.	1024	—	2	—	1.6
MirrorDatabaseLatency-Bytes	*mirror_name*	On the backup failover member of a mirror, number of bytes of journal data received from the primary but not yet applied to mirrored databases on the backup (measure of how far behind the backup's databases are).	$2*10^7$	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max	Mult	Warn	Warn Mult
MirrorDatabaseLatencyFiles	*mirror_name*	On the backup failover member of a mirror, number of journal files received from the primary but not yet fully applied to mirrored databases on the backup (measure of how far behind the backup's databases are).	3	—	2	—	1.6
MirrorDatabaseLatency-Time	*mirror_name*	On the backup failover member of a mirror, time (in milliseconds) between when the last journal file was received from the primary and when it was fully applied to the mirrored databases on the backup (measure of how far behind the backup's databases are).	1000	4000	—	3000	—
MirrorJournalLatency-Bytes	*mirror_name*	On the backup failover member of a mirror, number of bytes of journal data received from the primary but not yet written to the journal directory on the backup (measure of how far behind the backup is).	$2*10^7$	—	2	—	1.6
MirrorJournalLatency-Files	*mirror_name*	On the backup failover member of a mirror, number of journal files received from the primary but not yet written to the journal directory on the backup (measure of how far behind the backup is).	3	—	2	—	1.6
MirrorJournalLatency-Time	*mirror_name*	On the backup failover member of a mirror, time (in milliseconds) between when the last journal file was received from the primary and when it was fully written to the journal directory on the backup (measure of how far behind the backup is).	1000	4000	—	3000	—
PhysicalBlock-ReadsPerMin		Number of physical block reads per minute.	1024	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max	Mult	Warn	Warn Mult
PhysicalBlockWrites-PerMin		Number of physical block writes per minute.	1024	—	2	—	1.6
ProcessCount		Number of active processes for the Caché instance.	100	—	2	—	1.6
RoutineCommandsPerMin		Number of routine commands per minute.	1024	—	2	—	1.6
RoutineLoadsPerMin		Number of routine loads per minute.	1024	—	2	—	1.6
RoutineRefsPerMin		Number of routine references per minute.	1024	—	2	—	1.6
SMHPercentFull		Percentage of the shared memory heap (generic memory heap) in use.	50	98	—	85	—
ShadowConnectionsLatency		Network latency (milliseconds) of shadow server connections to this data source.	1000	—	2	—	1.6
ShadowLatency		Network latency (milliseconds) of this shadow server's connection to data source.	1000	—	2	—	1.6
TransOpenCount		Number of open local transactions (local and remote).	100	—	2	—	1.6
TransOpenSecondsMax		Duration (seconds) of longest currently open local transaction.	60	—	2	—	1.6
WDBuffers		Average number of database buffers updated per write daemon cycle.	1024	—	2	—	1.6
WDCycleTime		Average number of seconds required to complete a write daemon cycle.	60	—	2	—	1.6
WDWIJTime		Average number of seconds spent updating the write image journal (WIJ) per cycle.	60	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max	Mult	Warn	Warn Mult
WDWriteSize		Average number of bytes written per write daemon cycle.	1024	—	2	—	1.6

Note: Some sensors are not sampled for all Caché instances. For example, the four `ECP...` sensors are sampled only on ECP data and application servers.

When you are monitoring a mirror member (see the "Mirroring" chapter of the *Caché High Availability Guide*), the following special conditions apply to Health Monitor:

- No sensors are sampled while the mirror is restarting (for example, just after the backup failover member has taken over as primary) or if the member's status in the mirror is indeterminate.
- If a sensor is in analysis mode for a period and the member's status in the mirror changes during the period, no chart is created and the sensor remains in analysis mode.
- Only the **MirrorDatabaseLatency*** and **MirrorJournalLatency*** sensors are sampled on the backup failover mirror member.
- All sensors *except* the **MirrorDatabaseLatency*** and **MirrorJournalLatency*** sensors are sampled on the primary failover mirror member.

Notification Rules

Health Monitor generates an alert (notification of severity 2) if three (3) consecutive readings of a sensor during a period are greater than the sensor maximum value, and a warning (notification of severity 1) if five (5) consecutive readings of a sensor during a period are greater than the sensor warning value. The maximum and warning values depend on the settings in the sensor object and whether the applicable chart was generated by Health Monitor or created by a user, as shown in the following table.

Note also that, as described in Health Monitor Process Description:

- When a sensor object has maximum value and warning value set, no chart is required and therefore no chart is generated, and notifications are generated even when Health Monitor is disabled.
- When a sensor object has multiplier and warning multiplier set, or base only, a chart is required; until sufficient samples have been collected in analysis mode to generate the chart, no notifications are generated.
- When a user-created chart exists, it does not matter what the sensor object settings are.

Sensor Object Settings	Chart Type	Sensor Maximum Value	Sensor Warning Value	Active When
base, maximum value, warning value	none	sensor object maximum value	sensor object warning value	System Monitor running
base, multiplier, warning multiplier	generated	sensor object multiplier times greater of: • chart mean plus three sigma • highest chart value plus one sigma	sensor object warning multiplier times greatest of: • base • chart mean plus two sigma • highest chart value	System Monitor running, Health Monitor enabled
base only	generated	greater of: • chart mean plus three sigma • highest chart value	greater of: • chart mean plus two sigma • highest chart value	
(n/a if user-created chart exists)	user-created	chart alert value	chart warning value	

Periods

By default there are 63 recurring weekly periods during which sensors are sampled. Each of these periods represents one of the following specified intervals during a particular day of the week:

Table 4–4: Default Health Monitor Periods

00:15 a.m. – 02:45 a.m.	03:00 a.m. – 06:00 a.m.	06:15 a.m. – 08:45 a.m.
09:00 a.m. – 11:30 a.m.	11:45 a.m. – 01:15 p.m.	01:30 p.m. – 04:00 p.m.
04:15 p.m. – 06:00 p.m.	06:15 p.m. – 08:45 p.m.	09:00 p.m. – 11:59 p.m.

You can list, add and delete periods using the **Configure Periods** option in the Configure Health Monitor Classes submenu of the **^%SYSMONMGR** utility. You can add monthly, quarterly or yearly periods as well as weekly periods.

Note: Quarterly periods are listed in three-month increments beginning with the month specified as the start month; for example, if you specify `5` (May) as the starting month, the quarterly cycle repeats in August (`8`), November (`11`) and February (`2`).

Descriptions are optional for user-defined periods.

Charts

Health Monitor generates charts containing the readings taken for each sensor during analysis mode for each period, and the mean and sigma calculated from those readings. The mean, sigma and highest single value in the chart are used to evaluate some sensor readings, as described in Notification Rules.

The **Configure Charts** option of the **^%SYSMONMGR** utility Configure Health Monitor Classes submenu lets you display a list of all current charts. including the mean and sigma of each, and also display the details of a particular chart, including the individual readings and highest reading.

The **Configure Charts** option also provides two ways to customize alerting by customizing charts.

- By editing an existing chart, you can change the mean and/or sigma to whatever values you wish. The standard notification rules apply, but using the values you have entered.
- You can create a chart, specifying an alert value and a warning value. When you do this, the sensor object settings no longer apply; alerts and warnings are generated based solely on the values you supply for the chart.

Note: When listing, examining, editing or creating charts, the **Item** heading or prompt refers to a job type, a database directory path specifying a database, an IP address specifying a CSP gateway server, or a mirror name specifying a mirror; see Sensors and Sensor Objects for more information.

You can also programmatically build chart statistics based on a list of values with the following SYS.Monitor.Health.Chart class methods:

- **CreateChart()** — Creates a chart for a specific period/sensor, evaluates the list of values, and sets the resulting mean and sigma values.
- **SetChartStats()** — Evaluates the list of values and sets the resulting mean and sigma values for a specified period/sensor.

For more information, see the SYS.Monitor.Health.Chart class documentation.

Note: A chart generated by Health Monitor, including one you have edited, can be automatically recalibrated as described in the final step of Health Monitor Process Description. In addition, all charts generated by Health Monitor, including those that have been edited, are deleted when a Caché instance is upgraded. A chart *created* using the **Configure Charts** submenu or the **CreateChart()** class method, however, is never automatically recalibrated or deleted on upgrade. A user-created chart is therefore permanently associated with the selected sensor/period combination until you select the **Reset Charts** option within the **Reset Defaults** option of the Configure Health Monitor Classes submenu or select **Recalibrate Charts** within the **Configure Charts** option.

4.2.2 Using ^%SYSMONMGR to Manage Health Monitor

As described in Using the ^%SYSMONMGR Utility, the **^%SYSMONMGR** utility lets you manage and configure System Monitor, including Health Monitor. To manage Health Monitor, change to the %SYS namespace in the Terminal, then enter the following command:

```
%SYS>do ^%SYSMONMGR

1) Start/Stop System Monitor
2) Set System Monitor Options
3) Configure System Monitor Classes
4) View System Monitor State
5) Manage Application Monitor
6) Manage Health Monitor
7) View System Data
8) Exit

Option?
```

Enter **6** for **Manage Health Monitor**. The following menu displays:

```
1) Enable/Disable Health Monitor
2) View Alerts Records
3) Configure Health Monitor Classes
4) Set Health Monitor Options
5) Exit

Option?
```

Enter the number of your choice or press **Enter** to exit the Health Monitor utility.

Note: Health Monitor runs only in the %SYS namespace. When you start **^%SYSMONMGR** in another namespace, option 6 (**Manage Health Monitor**) does not appear.

The options in the main menu let you perform Health Monitor tasks as described in the following table:

Option	Description
`1) Enable/Disable Health Monitor`	• Enable Health Monitor (if it is disabled, as by default), so that it starts when System Monitor starts. Health Monitor does not begin collecting sensor reading until after the configured startup wait time is complete. • Disable Health Monitor (if it is enabled), so that it does not start when System Monitor starts.
`2) View Alert Records`	• View alert records for one or all sensors objects over a specified date range.
`3) Configure Health Monitor Classes`	• List notification rules. • List and delete existing periods and add new ones. • List, examine, edit, create and recalibrate charts. • List sensor objects and edit their settings. • Reset Health Monitor elements to their defaults.
`4) Set Health Monitor Options`	• Set startup wait time. • Specify when alert records should be purged.

Note: When the utility asks you to specify a single element such as a sensor, rule, period or chart, you can enter **?** (question mark) at the prompt for a numbered list, then enter the number of the element you want.

All output from the utility can be displayed on the Terminal or sent to a specified device.

4.2.2.1 View Alerts Records

Choose this option to view recently generated alerts for a specific sensor, or for all sensors. You can examine the details of individual alerts and warnings, including the mean and sigma of the chart and the readings that triggered the notification. (Alert records are purged after a configurable number of days; see the Set Health Monitor Options for more information.).

4.2.2.2 Configure Health Monitor Classes

The options in this submenu let you customize Health Monitor, as described in the following table.

Note: You cannot use these options to customize Health Monitor while System Monitor is running; you must first stop System Monitor, and then restart it after you have made your changes.

Option	Description
`1) Activate/ Deactivate Rules`	(not in use in this release)
`2) Configure Periods`	List the currently configured periods and add and delete periods.
`3) Configure Charts`	Lets you • List the mean and sigma of all existing charts, organized by period. • Examine individual charts in detail, including the readings on which the mean and sigma are based, with the highest reading called out. • Change the mean and sigma of an existing chart using the Edit Charts option. • Create a chart, specifying alert and warning thresholds. • Manually recalibrate all charts (including user-created charts) or an individual chart from the most recent data.
`4) Edit Sensor Objects`	List the sensor objects representing the sensors in the SYS.Monitor.SystemSensors class and modify their base, maximum, warning, multiplier, and warning multiplier values.
`5) Reset Defaults`	Lets you • Reset to the default period configuration and remove all existing charts, returning every period to analysis mode (see Health Monitor Process Description). • Remove all existing charts (including user-created charts), returning every period to analysis mode, without removing any user-defined period configuration. • Reset all sensor objects to their default values. • Reset the health monitor options (startup wait time and alert purge time) to their defaults

4.2.2.3 Set Health Monitor Options

This submenu lets you set several Health Monitor options, as shown in the following table:

Option	Description
`1) Set Startup Wait Time`	Configure the number of minutes System Monitor waits after starting, when Health Monitor is enabled, before passing sensor readings to the Health Monitor subscriber, SYS.Health.Monitor.Control. This allows Caché to reach normal operating conditions before Health Monitor begins creating charts or evaluating readings.
`2) Set Alert Purge Time`	Specify when an alert record should be purged (deleted); the default is five days after the alert is generated.

4.3 Caché Application Monitor

Caché Application Monitor monitors a user-extensible set of metrics, maintains a persistent repository of the data it collects, and triggers user-configured alerts.

This section covers the following topics:

- Application Monitor Overview
- Using ^%SYSMONMGR to Manage Application Monitor
- Application Monitor Metrics
- Writing User-defined Application Monitor Classes

4.3.1 Application Monitor Overview

Caché Application Monitor (Application Monitor) is an extensible utility that monitors a user-selected set of system and user-defined metrics in each startup namespace configured in System Monitor. As described in Default System Monitor Components, when **%SYS.Monitor.AppMonSensor**, the Application Monitor sensor class, is called by System Monitor, it samples metrics, evaluates the samples, and generates its own notifications. (Unlike System Monitor and Health Monitor notifications, these are not written to the console log.) Specifically, Application Monitor does the following in each System Monitor startup namespace:

1. Starts when System Monitor starts.
2. Lets you register the provided system monitor classes (they are registered in %SYS by default).
3. Lets you activate the system and user-defined classes you want to monitor. You can activate any registered system class; you can activate any user-defined class that is present in the local namespace. For example, if you have created a user-defined class only in the USER namespace, you can activate that class only in the USER namespace.
4. Monitors each active class by sampling the metrics specified by the class. These metrics represent the properties returned by the sample class called by the **GetSample()** method of the monitor class. For example, the %Monitor.System.LockTable class calls %Monitor.System.Sample.LockTable which returns (among others) the properties TotalSpace, containing the total size of the lock table, and UsedSpace, containing the size of the lock table space in use. The sampled data, along with monitor and class metadata, is stored in the local namespace and can be accessed by all object and SQL methods.
5. If an alert is configured for a class and the class returns a property value satisfying the evaluation expression configured in it, generates an email message or calls a class method, if one of these actions is configured in the alert. For example, you can first configure email notifications to a list of recipients, then configure an alert for the %Monitor.System.LockTable class, specifying that an email be sent when the ratio of the UsedSpace property of %Monitor.System.Sample.LockTable to the TotalSpace property is greater than .9 (90% full).

Note: The %Monitor.System.HistorySys and %Monitor.System.HistoryPerf classes provided with Application Monitor, when activated, create and maintain a historical database of system usage and performance metrics to help you analyze system usage and performance issues over time. These classes and %Monitor.System.HistoryUser run only in %SYS and cannot be registered in other namespaces. See the "Caché History Monitor" chapter of this guide for more information about these classes and the historical database.

4.3.2 Using ^%SYSMONMGR to Manage Application Monitor

As described in Using the ^%SYSMONMGR Utility, the **^%SYSMONMGR** utility lets you manage and configure System Monitor, including Application Monitor. The utility can be executed in any namespace, and changes made with it affect only the namespace in which it is started. You must maintain a separate Application Monitor configuration for each startup namespace you configure by starting **^%SYSMONMGR** in that namespace.

Note: Following any change you make to the Application Monitor configuration, such as activating a class, you must restart System Monitor in the namespace in which you made the change for the change to take effect.

To manage Application Monitor, enter the following command in the Terminal:

```
%SYS>do ^%SYSMONMGR
```

then enter **5** for **Manage Application Monitor**. The following menu displays:

```
1) Set Sample Interval
2) Manage Monitor Classes
3) Change Default Notification Method
4) Manage Email Options
5) Manage Alerts
6) Debug Monitor Classes
7) Exit

Option?
```

Enter the number of your choice or press **Enter** to exit the Application Monitor utility.

4.3.2.1 Manage Application Monitor

The options in the main menu let you manage Application Monitor as described in the following table:

Option	Description
`1) Set Sample Interval`	Sets the interval at which metrics are sampled; the default is 30 seconds. This setting can be overridden for an individual class by setting a class-specific interval using the `5) Set Class Sample Interval` option on the Manage Monitor Classes submenu. **Note:** As described in Set System Monitor Options, System Monitor calls each configured sensor class, including **%SYS.Monitor.AppMonSensor**, every 30 seconds by default, but this setting can also be changed. If the Application Monitor sampling interval or a class-specific interval is different from the System Monitor interval, whichever interval is longer is in effect. For example, if the System Monitor interval is 30 and the Application Monitor interval is 120, all active Application Monitor classes are sampled every 120 seconds; if the System Monitor interval is 60 and the %Monitor.System.LockTable class interval is 20, the class is sampled every 60 seconds.
`2) Manage Monitor Classes`	Displays the Manage Monitor Classes submenu which lets you manage system- and user-defined monitor classes in the namespace in which you are running the Application Monitor Manager.
`3) Change Default Notification Method`	Lets you specify the default action for alerts when triggered. Any alerts you create use this action unless you specify otherwise.
`4) Manage Email Options`	Displays the Monitor Email Options submenu which lets you enable and configure email notifications so you can specify this action in alerts.
`5) Manage Alerts`	Displays Manage Alerts submenu which lets you create alerts for system and user-defined monitor classes.
6) Debug Monitor Classes	Displays Debug Monitor Classes menu which lets you enable and disable debugging as well as lists errors.

4.3.2.2 Manage Monitor Classes

This submenu lets you manage system and user-defined monitor classes. Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 2

1) Activate/Deactivate Monitor Class
2) List Monitor Classes
3) Register Monitor System Classes
4) Remove/Purge Monitor Class
5) Set Class Sample Interval
6) Exit

Option?
```

This submenu displays a list of menu items that let you manage the system- and user-defined classes as described in the following table:

Option	Description
`1) Activate / Deactivate Monitor Class`	Application Monitor samples active classes only. This option lets you activate an inactivate class, or deactivate an active one. You can display a numbered list of the system and user-defined classes registered in the local namespace, including the activation state of each, by entering **?** at the `Class?` prompt, then enter either the number or class name.
`2) List Monitor Classes`	Displays a list of the system and user-defined classes registered in the local namespace, including the activation state of each.
`3) Register Monitor System Classes`	Registers all system monitor classes (except the %Monitor.System.HistorySys, %Monitor.System.HistoryPerf, and %Monitor.System.HistoryUser classes) and stores them in the local namespace. System classes must still be activated using option `1) Activate/Deactivate Monitor Class` on this menu for sampling to begin.
`4) Remove/Purge Class`	Removes a monitor class from the list of classes in the local namespace. You can display a numbered list of the system and user-defined classes registered in the local namespace, including the activation state of each, by entering **?** at the `Class?` prompt, then enter either the number or class name. **Note:** This option does not remove the class, but simply removes the name of the class from the list of registered classes that can be activated. To reset the list, choose option `3) Register Monitor System Classes` on this menu.
`5) Set Class Sample Interval`	Lets you override the default Application Monitor sampling interval, specified by the `1) Set Sample Interval` option of the Manage Application Monitor menu, for a single class. The default is `0`, which means the class does not have a class-specific sample interval. **Note:** See the description of the `Set Sample Interval` option for an explanation of precedence between this setting, the `Set Sample Interval` setting, and the System Monitor sample interval discussed in Set System Monitor Options.

4.3.2.3 Change Default Notification Method

When you create an alert, you specify an action to be taken when it is triggered; the default choice for this action is the default notification method, set using this option. Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 3
Notify Action (0=none,1=email,2=method)? 0 =>
```

The choice you make with this option is used when you configure an alert to use the default notification method, as described in the following table:

Input Field	Description
`0`	Do not take action when an alert is triggered.
`1`	Send an email message to the configured recipients when an alert is triggered. For information about configuring email, see Manage Email Options.

Input Field	Description
2	Call a notification method when an alert is triggered. If you select this action, the method is called with two arguments – the application name specified in the alert and a %List object containing the properties returned to the monitor class by the sample class (as described in Application Monitor Overview). When prompted, enter the full class name and method, that is*packagename.classname.method*. This method must exist in the local namespace.

4.3.2.4 Manage Email Options

The options in this submenu let you configure and enable email. When email is enabled, Application Monitor sends email notifications when an alert configured for them is triggered. Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 4

1) Enable/Disable Email
2) Set Sender
3) Set Server
4) Manage Recipients
5) Set Authorization
6) Test Email
7) Exit

Option?
```

The options in this submenu let you manage the email notifications for the Application Monitor as described in the following table:

Option	Description
1) Enable / Disable Email	Enabling email makes it possible for Application Monitor to send email notifications when alerts are triggered, if configured. Disabling email prevents Application Monitor from sending email notifications when an alert is triggered. **Note:** It is not necessary to reconfigure email options when you disable and then reenable email.
2) Set Sender	This option is required. Enter text identifying the sender of the email. Depending on the specified outgoing mail server, this may or may not have to be a valid email account.
3) Set Server	This option is required. Enter the name of the server that handles outgoing email for your site. If you are not sure, your IT staff should be able to provide this information.

4) Manage Recipients	This option displays a submenu that lets you list, add, or remove valid email addresses of recipients:
	
	`1) List Recipients`
	`2) Add Recipient`
	`3) Remove Recipient`
	`4) Exit`
	
	When adding or removing recipients, email addresses must be entered individually, one per line. Addresses of invalid format are rejected.

Option	Description
`5) Set Authorization`	Lets you specify the authorization username and password if required by your email server. Consult your IT staff to obtain this information. If you do not provide entries, the authorization username and password are set to NULL.
`6) Test Email`	Sends a test message to the specified recipients using the specified email server. If the attempt fails, the resulting error message may help you fix the problem.

4.3.2.5 Manage Alerts

An alert specifies

- a condition within the namespace that is of concern to you, defined by the values of properties sampled by a monitor class
- an action to be taken to notify you when that condition occurs

To return to the previous example, you might create an alert specifying

- condition: the lock table is over 90% full, defined by the UsedSpace property returned when the %Monitor.System.LockTable class calls %Monitor.System.Sample.LockTable being more than 90% of the TotalSpace property
- action: send an email notification

The definition of a condition based on properties is called an *evaluation expression*; after specifying the properties of the sample class you want to use, you specify the evaluation expression. Properties are indicated in the expression by place-holders corresponding to the order in which you provide them; for example, if when creating the lock table alert you specify the UsedSpace property first and then the TotalSpace property, you would enter the evaluation expression as **`%1 / %2 > .9`**, but if you enter the properties in the reverse order, the expression would be **`%2 / %1 > .9`**.

When the alert menu displays, enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 2

1) Create Alert
2) Edit Alert
3) List Alerts
4) Delete Alert
5) Enable/Disable Alert
6) Exit

Option?
```

The options in this submenu let you manage alerts for the Application Monitor as described in the following table:

Input Field	Description
`1) Create Alert`	Lets you define a new alert. For a description of the prompts and responses, see the Responses to Alert Prompts. The newly created alert is enabled by default.
`2) Edit Alert`	Lets you modify an existing alert. Enter the name of the alert to edit, or enter **?** for a list of existing alerts and then enter a number or name. **Note:** You must respond to all prompts including those that you do not want to modify; that is, you must re-enter information for fields that you do not want to modify as well as the revised information for the fields you are modifying. For a description of the prompts and responses, see the Responses to Alert Prompts.

Input Field	Description

`3) List Alerts`	Displays the definitions of all alerts in the local namespace; for example:
	`Alert: LockTable90 USER`
	`Action: email`
	`Class: %Monitor.System.LockTable`
	`Property: UsedSpace,TotalSpace`
	`Expression: %1/%2>.9`
	`Notify Once: True`
	`Enabled: Yes`

`4) Delete Alert`	Lets you delete an existing alert. Enter the name of the alert to edit, or enter **?** for a list of existing alerts and then enter a number or name. **Note:** Each alert must be entered individually; that is, you cannot specify a series or range of alerts to delete.
`5) Enable / Disable Alert`	Enabling an alert activates it. Disabling an alert deactivates it. **Note:** It is not necessary to reconfigure alert options when you disable and then reenable an alert.

The following table describes the valid responses to Alert prompts:

Table 4–5: Responses to Alert Prompts

Input Field	Description
`Alert Name?`	Enter an alphanumeric name. To display a numbered list of alert names already defined in the local namespace, enter **?** at the `Alert Name?` prompt.
`Application?`	Enter descriptive text to be passed to the email message or notification method. This text can include references to the properties you specify at the `Property?` prompt later in the procedure in the form %*N*, where %1 refers to the first property in the list of properties, %2 the second property, and so on.

Input Field	Description
`Action (0=default, 1=email, 2=method)?`	Specifies the action to take when the alert is triggered. Enter one of the following options: • **`0`** – Use the notification method identified as the default method (none, email, or class method). See Change Default Notification Method. • **`1`** – Send an email message containing your descriptive text and the names and values of the properties returned to the monitor class by the sample class (as described in Application Monitor Overview) to the configured email recipients when an alert is triggered. For information about configuring email, see Manage Email Options. • **`2`** – Call a specified notification method with two arguments – your descriptive text and a %List object containing the properties returned to the monitor class by the sample class (as described in Application Monitor Overview). When prompted, enter the full class name and method, that is*packagename.classname.method*. This method must exist in the local namespace.
`Raise this alert during sampling?` or `Define a trigger for this alert?`	The first prompt is displayed when are creating an alert; the send prompt is displayed when you are editing an alert for which you entered `No` at the first prompt when creating it. Enter one of the following: • **`Yes`** – Continues prompting for required information. • **`No`** – Skips to the end, bypassing `Class`, `Property` and `Evaluation expression` prompts.
`Class?`	Enter the name of a system or user-defined monitor class registered in the local namespace. To display a numbered list of registered classes in the local namespace, including its activation state, enter **`?`** at the `Class?` prompt, then enter a number or name. **Note:** You can create an alert for an inactive class. An alert is not removed when the class it is configured for is removed.
`Property?`	Enter the name of a property defined in the class specified in the preceding prompt that you are using in the evaluation expression, in the descriptive text, or in both.. To display a numbered list of properties defined in the named class, enter **`?`** at the `Property?` prompt, then enter a number or name. Each property must be entered individually. When you are done, press **Enter** at an empty prompt to display the list of properties in the order in which you specified them.
`Evaluation expression?`	Expression used to evaluate the properties specified at the `Property?` prompt. For example, in **`(%1 = "User") && (%2 < 100)`**, %1 refers to the first property in the list of properties, %2 the second property, and so on.
`Notify once only?`	Enter one of the following: • **`Yes`** – Notify users only the first time an alert is triggered. • **`No`** – Notify users every time an alert is triggered.

4.3.2.6 Debugging Monitor Classes

This submenu lets you manage system debugging.

Debugging monitor classes adds the capability to capture any errors generated during the collection of sample values by user-defined Application Monitor classes.

Enter the number of your choice or press **Enter** to return to the main menu:

```
Option? 6

1) Enable Debug
2) Disable Debug
3) List Errors
4) Exit

Option?
```

The options in this submenu let you manage debugging for Application Monitor as described in the following table:

Input Field	Description
`1) Enable Debug`	Lets you enable debugging. If the class is not creating sample values, then you can check to see if errors are preventing the sample values from being saved.
`2) Disable Debug`	Lets you disable debugging.
`3) List Errors`	Displays the definitions of all errors in the local namespace; for example: **%Save()**, **%New()**, **Initialize()** and **GetSample()**. Enable debugging for the class using **^%SYSMONMGR**, and the System Monitor will save the last error caught for specific methods within the class.

4.3.3 Application Monitor Metrics

The system monitor classes included with Application Monitor call various sample classes, as shown in the following table:

Sample Classes	Application Monitor System Classes
Audit metrics	%Monitor.System.Sample.AuditCount and %Monitor.System.Sample.AuditEvents
Client metrics	%Monitor.System.Sample.Clients
CSP Gateway metrics	%Monitor.System.Sample.CSPGateway
Disk space metrics	%Monitor.System.Sample.Diskspace
Free space metrics	%Monitor.System.Sample.Freespace
Global metrics	%Monitor.System.Sample.Globals
History database metrics (see the "Caché History Monitor" chapter of this guide)	%Monitor.System.Sample.HistoryPerf, %Monitor.System.Sample.HistorySys, %Monitor.System.Sample.HistoryUser
Journal metrics	%Monitor.System.Sample.Journals
License metrics	%Monitor.System.Sample.License
Lock table metrics	%Monitor.System.Sample.LockTable

Sample Classes	Application Monitor System Classes
Process metrics	%Monitor.System.Sample.Processes
Routine metrics	%Monitor.System.Sample.Routines
Server metrics	%Monitor.System.Sample.Servers
System activity metrics	%Monitor.System.Sample.SystemMetrics

For a list of properties corresponding to the sample metrics in each category, see the *InterSystems Class Reference*.

Similar functions that control the **MONITOR** facility are available through the classes in the %Monitor.System package, which also allows you to save the data as a named collection in a persistent object format. See the %Monitor.System.Sample package classes and the %Monitor.System.SystemMetrics class documentation in the *InterSystems Class Reference* for more details.

4.3.3.1 Generating Metrics

The %Monitor.SampleAgent class does the actual sampling, invoking the **Initialize()** and **GetSample()** methods of the metrics classes.

The **%Monitor.SampleAgent.%New(n)** constructor takes one argument, the name of the metrics class it is to run. It creates an instance of that class, and invokes the **Startup()** method on that class. Then, each time the **%Monitor.SampleAgent.Collect()** method is invoked, the Sample Agent invokes the **Initialize()** method for the class, then repeatedly invokes the **GetSample()** method for the class. On each call to **GetSample()**, %Monitor.SampleAgent creates a sample class for the metrics class; the pseudocode for these operations is:

```
set sampler = ##class(%Monitor.SampleAgent).%New("MyMetrics.Freespace")
/* at this point, the sampler has created an instance of MyMetrics.Freespace,
and invoked its Startup method */
for I=1:1:10 { do sampler.Collect() hang 10 }
/* at each iteration, sampler calls MyMetrics.Freespace.Initialize(), then loops
on GetSample().  Whenever GetSample() returns $$$OK, sampler creates a new
MyMetrics.Sample.Freespace instance, with the sample data. When GetSample()
returns an error value, no sample is created, and sampler.Collect() returns. */
```

4.3.3.2 Viewing Metrics Data

All metrics classes are CSP-enabled; the CSP code is generated automatically when the sample class is generated. Therefore, the simplest way to view metrics is using a web browser; for example, based on the example in Generating Metrics and assuming a superserver port of 57772, the CSP URL is: http://localhost:57772/csp/user/MyMetrics.Sample.Freespace.cls, which displays output similar to:

```
Monitor - Freespace c:\cache51\
            Name of dataset:  c:\cache51\
Current amount of Freespace:  8.2MB

Monitor - Freespace c:\cache51\mgr\
            Name of dataset:  c:\cache51\mgr\
Current amount of Freespace:  6.4MB
```

Alternatively, you can use the **Display(***metric_class***)** method of the %Monitor.View class; for example:

```
%SYS>set mclass="Monitor.Test.Freespace"

%SYS>set col=##class(%Monitor.SampleAgent).%New(mclass)

%SYS>write col.Collect()
1
%SYS>write ##class(%Monitor.View).Display(mclass)

Monitor - Freespace    c:\cache51\
                Name of dataset:  c:\cache51\
    Current amount of Freespace:  8.2MB

Monitor - Freespace    c:\cache51\mgr\
                Name of dataset:  c:\cache51\mgr\
    Current amount of Freespace:  6.4MB
```

Note: The URL for a class with % (percent sign) in the name must use %25 in its place. For example, for the %Monitor.System.Freespace class, use

http://localhost:57772/csp/sys/%25Monitor.System.Freespace.cls

4.3.4 Writing User-Defined Application Monitor Classes

In addition to the provided system classes, you can write your monitor and sample classes to monitor user application data and counters.

A monitor class is any class that inherits from the abstract Monitor class, %Monitor.Adaptor; the %Monitor.System classes are examples of such classes. To create your own user-defined monitor classes:

1. Run **^%MONAPPMGR** in the namespace where you want to monitor data. Use option 2 to list monitor classes, and within that menu, use option 3 to register monitor system classes.

```
SAMPLES>d ^%MONAPPMGR


1) Set Sample Interval
2) Manage Monitor Classes
3) Change Default Notification Method
4) Manage Email Options
5) Manage Alerts
6) Exit

Option? 2

1) Activate/Deactivate Monitor Class
2) List Monitor Classes
3) Register Monitor System Classes
4) Remove/Purge Monitor Class
5) Set Class Sample Interval
6) Exit

Option? 3
Exporting to XML started on 06/21/2022 12:52:36
Exporting class: Monitor.Sample
Export finished successfully.

Load started on 06/21/2022 12:52:36
Loading file C:\InterSystems\SRCCTRL\mgr\Temp\t0jFhPqLkZoYAA.stream as xml
Imported class: Monitor.Sample
Compiling class Monitor.Sample
Compiling table Monitor.Sample
Compiling routine Monitor.Sample.1
Load finished successfully.


1) Activate/Deactivate Monitor Class
2) List Monitor Classes
3) Register Monitor System Classes
4) Remove/Purge Monitor Class
5) Set Class Sample Interval
6) Exit

Option?
```

2. Write a class that inherits from %Monitor.Adaptor. The inheritance provides persistence, parameters, properties, code generation, and a projection that generates the monitor metadata from your class definition. See the %Monitor.Adaptor class documentation for full details on this class, as well as the code you must write.
3. Compile your class. Compiling classes that inherit from %Monitor.Adaptor generates new sample classes in a subpackage of the users class called Sample. For example, if you compile A.B.MyMetric, a new class is generated in A.B.Sample.MyMetric. You do not need to do anything with the generated class.

   Important: When deleting application monitor classes, only the monitor class should be deleted; that is, do not delete generated sample classes. Use the Management Portal to delete only the monitor class (for example, A.B.MyMetric) from which the sample class (for example, A.B.Sample.MyMetric) is generated; this automatically deletes both the monitor class and generated sample class.

All sample classes are automatically CSP-enabled, so that sample data for the user's metrics may be viewed by pointing to A.B.Sample.MyMetric.cls. Application Monitor automatically invokes this class and generates data and alerts, if the class has been activated; for information about activating monitor classes, see Manage Monitor Classes.

Important: The **SECURITYRESOURCE** parameter is empty in %Monitor.Adaptor, and therefore in user classes inheriting from %Monitor.Adaptor unless explicitly modified. Code generation copies the **SECURITYRESOURCE** value from the user-defined class to the generated sample class. See %CSP.Page Class Parameters in *Using Caché Server Pages (CSP)* for information about the **SECURITYRESOURCE** parameter.

The following simple example retrieves the free space for each dataset in a Caché instance. For detailed instructions for creating a user-defined Application Monitor class and alert to send email notifications when an application error occurs, written by an InterSystems senior technical trainer and accompanied by downloadable code, see Creating a Custom Application Monitor Class and an Alert on InterSystems Developer Community.

Each sampling requests *n* instances of sample data objects, each instance corresponding to a dataset. In this example, each instance has only a single property, the free space available in that dataset when the sample was collected.

1. Create a class that inherits from %Monitor.Adaptor:

   ### Class Definition

   ```
   Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
   {
   }
   ```

2. Add properties that you want to be part of the sample data. They must be of %Monitor types:

   - Gauge
   - Integer
   - Numeric
   - String

   For example:

   ### Class Definition

   ```
   Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
   {
       /// Name of dataset
       Property DBName As %Monitor.String(CAPTION = "Database Name");

       /// Current amount of Freespace
       Property FreeSpace As %Monitor.String;
   }
   ```

3. Add an INDEX parameter that tells which fields form a unique key among the instances of the samples:

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
    Parameter INDEX = "DBName";
}
```

4. Add control properties as needed, marking them `[Internal]` so they do not become part of the storage definition in the generated class.

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
    /// Result Set
    Property Rspec As %Library.ResultSet [Internal];
}
```

5. Override a method named **Initialize()**. Initialize is called at the start of each metrics gathering run.

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
    /// Initialize the list of datasets and freespace.
    Method Initialize() As %Status
    {
        set ..Rspec = ##class(%Library.ResultSet).%New("SYS.Database:FreeSpace")
        do ..Rspec.Execute("*",0)
        return $$$OK
    }
}
```

6. Override a method named **GetSample()**. **GetSample()** is called repeatedly until a status of `0` is returned. You write code to populate the metrics data for each sample instance.

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
    /// Get dataset metric sample.
    /// A return code of $$$OK indicates there is a new sample instance.
    /// A return code of 0 indicates there is no sample instance.
    Method GetSample() As %Status
    {
        // Get freespace data
        set stat = ..Rspec.Next(.sc)

        // Quit if we have done all the datasets
        if 'stat {
            Quit 0
        }

        // populate this instance
        set ..DBName = ..Rspec.Get("Directory")
        set ..FreeSpace = ..Rspec.Get("Available")

        // quit with return value indicating the sample data is ready
        return $$$OK
    }
}
```

7. Compile the class. The class is shown below:

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor
{
    Parameter INDEX = "DBName";
```

```
    /// Name of dataset
    Property DBName As %Monitor.String;

    /// Current amount of Freespace
    Property FreeSpace As %Monitor.String;

    /// Result Set
    Property Rspec As %Library.ResultSet [Internal];

    /// Initialize routine metrics.
    Method Initialize() As %Status
    {
        set ..Rspec = ##class(%Library.ResultSet).%New("SYS.Database:FreeSpace")
        do ..Rspec.Execute("*",0)
        return $$$OK
    }

    /// Get routine metric sample.
    /// A return code of $$$OK indicates there is a new sample instance.
    /// Any other return code indicates there is no sample instance.
    Method GetSample() As %Status
    {
        // Get freespace data
        Set stat = ..Rspec.Next(.sc)

        // Quit if we have done all the datasets
        if 'stat {
            Quit 0
        }

        // populate this instance
        set ..DBName = ..Rspec.Get("Directory")
        set ..FreeSpace = ..Rspec.Get("Available")

        // quit with return value indicating the sample data is ready
        return $$$OK
    }
}
```

8. Additionally, you can override the **Startup()** and **Shutdown()** methods. These methods are called once when sampling begins, so you can open channels or perform other one-time-only initialization:

Class Definition

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
    /// Open a tcp/ip device to send warnings
    Property io As %Status;
    Method Startup() As %Status
    {
        set ..io="|TCP|2"
        set host="127.0.0.1"
        open ..io:(host:^serverport:"M"):200
    }
    Method Shutdown() As %Status
    {
        close ..io
    }
}
```

9. Compiling the class creates a new class, MyMetric.Sample.Freespace in the MyMetric.Sample package :

Class Definition

```
/// Persistent sample class for MyMetric.Freespace
Class MyMetric.Sample.Freespace Extends Monitor.Sample
{
  Parameter INDEX = "DBName";

  Property Application As %String [ InitialExpression = "MyMetric" ];

  /// Name of dataset
  Property DBName As %Monitor.String(CAPTION = "");

  /// Current amount of Freespace
  Property FreeSpace As %Monitor.String(CAPTION = "");

  Property GroupName As %String [ InitialExpression = "Freespace" ];

  Property MetricsClass As %String [ InitialExpression = "MyMetric.Freespace" ];

}
```

Note: You should not modify this class. You may, however, inherit from it to write custom queries against your sample data.

Important: If you do modify and recompile an active user-defined Application monitor class, the class is deactivated and the class-specific sample interval override, if any, is removed; to restor it, you must activate it, reset the sample interval if desired, and restart of System Monitor. If System Monitor is running when you modify and recompile a user-defined class, all alerts based on the class are deleted.

5

Gathering Global Activity Statistics Using ^GLOSTAT

Caché provides the **^GLOSTAT** utility, which gathers global activity statistics and displays a variety of information about disk I/O operations. This chapter describes how to use the routine; it covers the following topics:

- Running ^GLOSTAT
- Overview of ^GLOSTAT Statistics
- Examples of ^GLOSTAT Output

You can also use the InterSystems Management Portal to view the statistics reported by **^GLOSTAT**. Logon to the portal application for the system you are monitoring and navigate to the **System, System Usage** page.

5.1 Running ^GLOSTAT

To run the **^GLOSTAT** routine you must be in the %SYS namespace. The name of the routine is case-sensitive.

1. Enter the following command:

   ObjectScript

   ```
   Do ^GLOSTAT
   ```

2. Press **Enter** to show summary block totals (Example A).
3. The **^GLOSTAT** routine displays statistics according to your request. Each time Caché starts, it initializes the **^GLOSTAT** statistical counters; therefore, the output of the *first* run reflects operations since Caché has started.

   After the first and subsequent displays of the **^GLOSTAT** report, the following prompt appears:

   ```
   Continue (c), Timed Stats (# sec > 0), Quit (q)?
   ```

   You may enter one of the following:

Response	Action
c	Displays the report again with updated cumulative statistics since the last initialization.
q	Quits the **^GLOSTAT** routine.
# (a positive integer indicating number of seconds)	Initializes statistics, counts statistics for the indicated number of seconds, and reports statistics as an average per second (Example B).

5.2 Overview of ^GLOSTAT Statistics

Each **^GLOSTAT** statistic represents the number of times a type of event has occurred since Caché has started since the counters have been initialized, or per second during a defined interval. You may run **^GLOSTAT** at any time from the system manager's namespace. In most cases, it is significant to run the utility on an active system, not an idle one.

If the Caché instance is a stand-alone configuration or an ECP data server, then the report displays only the "Total" column. If it is an ECP application server (that is, it connects to a remote database) then three columns are shown: "Local," "Remote," and "Total" (Example C).

The following table defines the **^GLOSTAT** statistics.

Table 5–1: Statistics Produced by ^GLOSTAT

Statistic	Definition
Global references (all)	Logical count of accesses to globals, including **Set**s, **Kill**s, **$Data**, **$Order**, **$Increment**, **$Query**, and global references in expressions.
Global update references	Logical count of global references that are **Set**s, **Kill**s, or **$Increment**s.
Private global references	The count of all process private global accesses.
Private update references	The count of process private global references that are **SET**s or **KILL**s, etc.
Routine calls	Number of calls to a routine.
Routine buffer loads and saves	Total number of routine loads and saves as a result of **ZLoad**, **ZSave**, and running routines. (In a well-tuned environment, this number increases slowly, since most routine loads are satisfied by the routine cache memory without accessing the disk. Each routine load or save transfers up to 32 KB of data (64 KB for Unicode).)
Routine commands	Number of routine commands executed since system startup.
WIJ writes	Number of writes to the write image journal file.
Routine not cached	Number of routines not cached in memory. This information help you determine whether or not the routine buffer cache is adequately sized.
Cache Efficiency	Number of all global references divided by the number of physical block reads and writes. *Not a percentage*.

Statistic	Definition
Journal Entries	Number of journal records created—one for each database modification (**Set** , **Kill**, etc.) or transaction event (**TStart**, **TCommit**) or other event that is saved to the journal.
Journal Block Writes	Number of 64-KB journal blocks written to the journal file.
Logical Block Requests	Number of database blocks read by the global database code. (In a well-tuned environment, many of these reads are satisfied without disk access.)
Physical Block Reads	Number of physical database blocks read from disk for both global and routine references.
Database Physical Block Writes	Number of physical database blocks written to disk for both global and routine references.
WIJ Physical Block Writes	Number of physical Write Image Journaling (WIJ) blocks written to disk for both global and routine references.
Blocks Queued to be Written	Number of database blocks that have been queued to be written to disk.

5.3 Examples of ^GLOSTAT Output

The following output samples show the various options when running the **^GLOSTAT** utility routine:

- Example A — Initial running on a stand-alone or server configuration.
- Example B — Subsequent running at a timed interval.
- Example C — Initial running on a client configuration.

5.3.1 Example A

The following is sample output of the initial running of the **^GLOSTAT** routine. The Caché instance is either a stand-alone configuration or a server:

```
%SYS>Do ^GLOSTAT

Statistics                                                      Total
--------------                                                ---------
Global references (all):                                        530,801
Global update references:                                       175,073
Private global references:                                      160,267
Private update references:                                       76,739
Routine calls:                                                  650,085
Routine buffer loads & saves:                                       570
Routine commands:                                            17,747,411
Routine not cached:                                                 710
Logical block requests:                                         289,166
Block reads:                                                      2,179
Block writes:                                                       680
WIJ writes:                                                         903
Cache Efficiency:                                                   186
Journal Entries:                                                  1,356
Journal Block Writes:                                                 6

Continue (c), Timed Stats (# sec > 0), Quit (q)?
```

5.3.2 Example B

The following example shows **^GLOSTAT** statistics per second for a 30-second timed interval. The Caché instance is either a stand-alone configuration or a server:

```
Continue (c), Timed Stats (# sec > 0), Quit (q)?  30

Counts per Second for 30 Seconds...

Statistics (per second)                                          Total
--------------                                                 ---------
Global references (all):                                           4.0
Global update references:                                          2.0
Private global references:                                         2.0
Private update references:                                         0.9
Routine calls:                                                     8.8
Routine buffer loads & saves:                                        0
Routine commands:                                                222.2
Routine not cached:                                                  0
Logical block requests:                                            2.3
Block reads:                                                         0
Block writes:                                                        0
WIJ writes:                                                          0
Cache Efficiency:                                               no i/o
Journal Entries:                                                     0
Journal Block Writes:                                                0

Continue (c), Timed Stats (# sec > 0), Quit (q)?
```

5.3.3 Example C

The following is sample output of the initial running of the **^GLOSTAT** routine. The Caché instance is a client:

```
%SYS>Do ^GLOSTAT

 Statistics                            Local           Remote            Total
--------------                      ---------       ---------        ---------
Global references (all):              123,783               3          123,786
Global update references:               6,628               0            6,628
Private global references:              3,558             n/a            3,558
Private update references:              1,644             n/a            1,644
Routine calls:                         55,275               0           55,275
Routine buffer loads & saves:             759               0              759
Routine commands:                                                   1,304,213
Routine not cached:                                                       167
Logical block requests:                83,959             n/a           83,959
Block reads:                            2,125               0            2,125
Block writes:                             217             n/a              217
WIJ writes:                               126             n/a              126
Cache Efficiency:                          53          no gets
Journal Entries:                          511             n/a              511
Journal Block Writes:                       3             n/a                3

Continue (c), Timed Stats (# sec > 0), Quit (q)?
```

6

Monitoring System Performance Using ^PERFMON

^PERFMON is a Caché utility that controls the **MONITOR** facility.

The **MONITOR** facility provides performance data for the Caché system by collecting counts of events at the system level and sorting the metrics by process, routine, global, and network nodes. Since there is some overhead involved in collecting this data, you must specifically enable the collection of counters and collect data for a specific number of processes, globals, routines, and network nodes. Caché allocates memory at **MONITOR** startup to create slots for the number of processes, routines, globals, and nodes specified. The first process to trigger an event counter allocates the first slot and continues to add to that set of counters. Once the facility allocates all the available slots to processes, it includes any subsequent process counts in the *Other* slot. It follows the same procedure for globals, routines, and nodes.

You can review reports of the data while collection is in progress. When you stop collection, memory is de-allocated and the counter slots are gone. So, any retention of the numbers needs to be handled by writing the reports to a file (or a global). Data is given as rates per second by default, although there is also an option for gathering the raw totals. There are also functions which allow you to pause/resume the collection, and zero the counters.

The menu items available by running **^PERFMON** correspond directly to functions available in the **^PERFMON** routine, and the input collected is used to directly supply the parameters of these functions.

Similar functions that control the same **MONITOR** facility are available through the classes in the %Monitor.System package. For more information see Caché Application Monitor in the "Using Caché System Monitor" chapter of this guide and the "Examining Routine Performance Using ^%SYS.MONLBL" chapter of this guide.

6.1 Using ^PERFMON

You can use the **^PERFMON** routine in two ways: running it interactively or calling its functions from your own routines. The menu items available from running **^PERFMON** correspond directly to callable functions in the **^PERFMON** routine; it uses the input it collects to directly supply the parameters of these functions. Each function returns a success or failure status (`1` for success and a string consisting of a negative number followed by a comma and a brief message for failure).

The following is an example of running the **^PERFMON** routine interactively from the terminal:

1. Enter the following command:

```
DO ^PERFMON
```

2. The following menu appears. Enter the number of your choice. Press **Enter** to exit the routine.

```
1. Start Monitor
2. Stop Monitor
3. Pause Monitor
4. Resume Monitor
5. Sample Counters
6. Clear Counters
7. Report Statistics
8. Timed Collect and Report

Monitor is Stopped

Enter the number of your choice:
```

Each of these menu options corresponds to a callable function in the routine. The following functions are available:

- Start
- Stop
- Pause
- Resume
- Sample Counters
- Clear
- Report
- Collect

Because **^PERFMON** and the line-by-line monitor routine **^%SYS.MONLBL** share the same memory allocation, you can only run one of them at a time on a Caché instance. You see the following message if you try to run **^PERFMON** and **^%SYS.MONLBL** has started monitoring:

```
The Line-by-line Monitor is already enabled.
This must be stopped before ^PERFMON can be used.
```

6.2 Start

Turns on collection of the statistics.

Format:

```
status = $$Start^PERFMON(process,routine,global,database,network)
```

Parameters:

- *process* — number of process slots to reserve (default = **$$pcount** (the number of processes in the process table))
- *routine* — number of routine slots to reserve (default = 200)
- *global* — number of global slots to reserve (default = 100)
- *database* — number of database slots to reserve (default = 10)
- *network* — number of network node slots to reserve (default = 5)

If you are running ^PERFMON interactively, it prompts you for each parameter value.

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-2	Monitor is already running
-3	Memory allocation failed
-4	Could not enable statistics collection

6.3 Stop

Stops collection of statistics.

Format:

```
status = $$Stop^PERFMON()
```

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-2	Monitor is not running

6.4 Pause

Momentarily pauses the collection of statistics to allow a consistent state for viewing data.

Format:

```
status = $$Pause^PERFMON()
```

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-2	Monitor is not running
-3	Monitor is already paused

6.5 Resume

Resumes collection of statistics that you previously paused.

Format:

```
status = $$Resume^PERFMON()
```

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-2	Monitor is not running
-3	Monitor is already running

6.6 Sample Counters

Starts a job to continuously Pause and Resume a collection, creating a periodic sampling of metrics. If wait_time = 0, the background job is stopped and collection is Paused.

Format:

```
status = $$Sample^PERFMON(wait_time,sample_time)
```

Parameters:

- *wait_time* — number of seconds between collections (default = 10)
- *sample_time* — number of seconds a collection should last (default = 1)

Status Codes:

Status code	Description
1	Successful
-2	Monitor is not running
-8	Sample job already running

6.7 Clear

Clears all metric counters.

Format:

```
status = $$Clear^PERFMON()
```

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-2	Monitor is not running

6.8 Report

The report function gathers and outputs a report of metrics.

Format:

```
status = $$Report^PERFMON(report,sort,format,output,[list],[data])
```

Parameters:

- *report* — type of report to output; valid values are:
  - `G` – for global activity
  - `R` – for routine activity
  - `N` – for network activity
  - `C` – for a custom report where you select the metrics to report
- *sort* — grouping and sort order of report; valid values are:
  - `P` – to organize the report by Process
  - `R` – to organize the report by Routine
  - `G` – to organize the report by Global
  - `D` – to organize the report by Database
  - `I` – to organize the report by Incoming node
  - `O` – to organize the report by Outgoing node
- *format* — output format; valid values are:
  - `P` – for a printable/viewable report (.txt file, no pagination)
  - `D` – for comma delimited data (.csv file) which can be read into a spreadsheet
  - `X` – for Microsoft Excel XML markup suitable for import into Excel (.xml file)
  - `H` – for an HTML page (.html file)
- *output* — enter a file name, **Return** to accept the default file name displayed, or `0` (zero) for output to the screen
- *list* — (*for Custom report only*) comma-separated list of metric numbers which specify what columns to include in the report; the following table lists the metrics available.

Table 6–1: Metrics for Custom ^PERFMON Report

Metric Number	Column Title	Description
1	GloRef	global references
2	GloSet	global sets
3	GloKill	global kills
4	TotBlkRd	total physical block reads (sum of next seven counters)
5	DirBlkRd	directory block reads
6	UpntBlkRd	upper pointer block reads
7	BpntBlkRd	bottom pointer block reads
8	DataBlkRd	data block reads
9	BdataBlkRd	big data block reads
10	MapBlkRd	map block reads
11	OthBlkRd	other block reads
12	DirBlkWt	directory block writes
13	UpntBlkWt	upper pointer block writes
14	BpntBlkWt	bottom pointer block write
15	DataBlkWt	data block writes
16	BdataBlkWt	big data block writes
17	MapBlkWt	map block writes
18	OthBlkWt	other block writes
19	DirBlkBuf	directory block requests satisfied from a global
20	UpntBlkBuf	upper pointer block requests satisfied from a global buffer
21	BpntBlkBuf	bottom pointer block requests satisfied from a global buffer
22	DataBlkBuf	data block requests satisfied from a global buffer
23	BdataBlkBuf	big data block requests satisfied from a global buffer
24	MapBlkBuf	map block requests satisfied from a global buffer
25	OthBlkBuf	other block requests satisfied from a global buffer
26	JrnEntry	journal entries
27	BlkAlloc	blocks allocated
28	NetGloRef	network global refs
29	NetGloSet	network sets
30	NetGloKill	network kills
31	NetReqSent	network requests sent

Metric Number	Column Title	Description
32	NCacheHit	network cache hits **Note:** This counter is no longer available in the current network protocol. However, network cache hits can be calculated using the formula NetGloRef - NetGloSet - NetGloKill - NCacheMiss.
33	NCacheMiss	network cache misses
34	NetLock	network locks
35	RtnLine	ObjectScript lines
36	RtnLoad	routine loads
37	RtnFetch	routine fetches
38	LockCom	lock commands
39	LockSucc	successful lock commands
40	LockFail	failed lock commands
41	TermRead	terminal reads
42	TermWrite	terminal writes
43	TermChRd	terminal read chars
44	TermChWrt	terminal write chars
45	SeqRead	sequential reads
46	SeqWrt	sequential writes
47	IJCMsgRd	local IJC messages read
48	IJCMsgWt	local IJC messages written
49	IJCNetMsg	network IJC messages written
50	Retransmit	network retransmits
51	BuffSent	network buffers sent

The global, routine, and network activity reports (indicated by the *report* parameter) display a predefined subset of this list.

- *data* — type of data to report; valid values are:
  - 1 – for standard rates/second
  - 2 – for raw totals

Status Codes:

Status code	Description
1	Successful
-1	Monitor is not running
-2	Missing input parameter
-3	Invalid report category
-4	Invalid report organization
-5	Invalid report format
-6	Invalid list for custom report
-7	Invalid data format

The Report Examples section shows how to enter different values for the input parameters.

6.9 Collect

The timed collect and report function provides a fast automated snapshot of system performance by collecting metrics for a specified period (30 seconds by default), creating five basic reports and a process count, and formatting them together as either an Excel spreadsheet or an HTML page.

Format:

```
status = $$Collect^PERFMON(time,format,output)
```

Parameters:

- *time* — number of seconds for data collection (default 30)
- *format* — output format; valid values are:
  - `XML` – for Microsoft Excel XML markup suitable for import into Excel (.xml file)
  - `HTML` – for an HTML page (.html file)
  - `CSV`
- *output* — enter a file name, **Return** to accept the default file name displayed, or `0` (zero) for output to the screen

Status Codes:

Status code	Description
1	Successful
-1	Somebody else is using Monitor
-3	Monitor is already running

6.10 Report Examples

The following is an example of running a report of global statistics, gathered and sorted by global name and output to a file in the manager's directory called perfmon.txt.

```
%SYS>Do ^PERFMON


1. Start Monitor
2. Stop Monitor
3. Pause Monitor
4. Resume Monitor
5. Sample Counters
6. Clear Counters
7. Report Statistics
8. Timed Collect & Report

Enter the number of your choice: 7


Category may be: G=Global, R=Routine, N=Network or C=Custom
Category ('G', 'R', 'N' or 'C'): g
Sort may be: P=Process, R=Routine, G=Global, D=Database, I=Incoming or O=Outgoing node
Sort ('P', 'R', 'G', 'D', 'I' or 'O'): g
Format may be: P=Print, D=Delimited data, X=Excel XML, H=HTML
Format ('P', 'D', 'X', 'H'): p
File name: perfmon.txt



Press RETURN to continue ...
```

The following is an example of running a custom report of statistics that correspond to metrics with the following numbers: `5,10,15,20,25,30,35,40,45,50`. The counts are gathered and sorted by process ID and output to a file in the manager's directory called perfmonC.txt.

```
1. Start Monitor
2. Stop Monitor
3. Pause Monitor
4. Resume Monitor
5. Sample Counters
6. Clear Counters
7. Report Statistics
8. Timed Collect & Report

Enter the number of your choice: 7


Category may be: G=Global, R=Routine, N=Network or C=Custom
Category ('G', 'R', 'N' or 'C'): c
List of field numbers: 5,10,15,20,25,30,35,40,45,50
Sort may be: P=Process, R=Routine, G=Global, D=Database, I=Incoming or O=Outgoing node
Sort ('P', 'R', 'G', 'D', 'I' or 'O'): p
Format may be: P=Print, D=Delimited data, X=Excel XML, H=HTML
Format ('P', 'D', 'X', 'H'): p
File name: perfmonC.txt
```

7

Monitoring Routine Performance Using ^PROFILE

The **^PROFILE** utility helps programmers analyze the performance of their application routines and classes. It accomplishes this task in two phases:

1. It gathers data, sorted at the routine level, to help you identify which routines do the most "work."
2. It lets you select routines for which you want to gather and display data (subroutines, procedures, and individual lines) at a detail level so that you can "drill down" into the individual routines that may be causing performance issues.

By default, **^PROFILE** captures metrics for up to 5000 routines; if there is not enough shared memory available for the maximum number of routines, it displays an informational message before it starts to capture the metrics, then captures metrics for as many routines as possible.

After you select the routines to monitor and the metrics to gather, if there is not enough space to collect metrics on all routines running in Caché, the utility displays a message about the number of pages of memory required to monitor this collection and the number of pages available.

This chapter includes the following sections:

7.1 Using ^PROFILE

Invoke the (**^PROFILE**) utility from the %SYS namespace:

```
%SYS>do ^PROFILE
```

When you are prompted to start the collection of data, press **Enter**.

Note: When you are prompted for a response (other than `Yes` or `No`) you can enter `?` to display online help.

By default, the profile displays a numbered list of routines with the following metrics; initially, the list is sorted by the **RtnLine** metrics:

Column Title (Metric)	Description
RtnLine	Number of routine lines of code executed. By default, it lists the value as a percentage of all lines of code executed.
Time	Elapsed time used to execute the routine. By default, the time is listed as a percentage of the total time used by all routines.
CPU	CPU time used to execute the routine. By default, the entry is listed as a percentage of the total CPU time used by all routines.
RtnLoad	Number of times the routine is loaded. By default, the entry is listed as a percentage of all routine loads.
GloRef	Number of global references by the routine. By default, the entry is listed as a percentage of global references by all routines.
GloSet	Number of global sets by the routine. By default, the entry is listed as a percentage of global sets by all routines.

The name of the routine (INT or MVI file) and the namespace where it is executing is displayed on the second line of the entry.

Follow the instructions that are displayed in the Terminal:

- When the list of routines is displayed at the profile level, you can specify any of the following:

Option	Description
#	Flag the specified line(s) for detailed profile-level data collection. **Note:** On each displayed page, you can enter single line numbers (#), a comma-separated list (#,#,#), a range (#-#), or a combination (#-#,#,#-#,#). After you select the routines on any page, you can move to the next or previous page to select other routines. After you select all the routines you want to analyze, enter **Q** to start the detail level profile collection.
B	Display the previous page of the list.
E	Export the displayed collection of metrics.
N	Display the next page of the list.
O	Re-sort the page based on different metrics (the selected metric is displayed in the first column).
Q	Exit from the **^PROFILE** utility. **Note:** If you flagged routines that you want to analyze, this option lets you choose between collecting subroutine- and line-level metrics or exiting.

Option	Description
R	Refresh the list with the most recent metrics.
X	Clear all flags of selected routines (including those selected on other pages) and refresh the collection of metrics.

- When the list of routines is displayed at the *detailed* profiling level, you can specify any of the following:

Option	Description
#	The line number of the routine you want to analyze in more detail. After you press **Enter**, the subroutine labels in then selected routine is displayed.
B	Display the previous page of the list.
N	Display the next page of the list.
O	Re-sort the page based on different metrics (the selected metric is displayed in the first column).
Q	Exit from the **^PROFILE** utility.
R	Refresh the list with the most recent metrics.

- When the list of subroutine labels (and metrics for each label) are displayed, you can specify any of the following:

Option	Description
#	The line number of the subroutine label (in the code) you want to analyze in more detail. After you press **Enter**, the code for the specified label is displayed.
B	Display the previous page of the list.
L	Switch to the line level display of the subroutine.
N	Display the next page of the list.
Q	Exit the list, return to the previous level.
R	Refresh the list with the most recent metrics. **Note:** If `*Unknown*` is displayed in the listing, enter **R**.

- When lines of code are displayed, you are prompted to specify what you want to do next. Your options are:

Option	Description
#	The line number in the code you want to analyze in more detail. After you press **Enter**, the code for the specified label is displayed.
B	Display the previous page of the list.
C	Switch code display between source code and intermediate (INT/MVI) code.

Option	Description
M	Change the page margin and length.
N	Display the next page of the list.
O	Re-sort the page based on different metrics.
Q	Exit the list, returning to the previous level.
R	Refresh the list with the most recent metrics.
S	Switch to the subroutine level display of the routine.

7.2 ^PROFILE Example

Following is an example of running the **^PROFILE** utility interactively (from the %SYS namespace) in the Terminal:

1. Enter the following command:

```
DO ^PROFILE
```

2. The following message appears.

```
This routine will start a system-wide collection of data
on routine activity and then display the results. There
may be some overhead associated with this collection,
but it should not significantly impact the system.

Are you ready to start the collection?  Yes =>
```

3. Press **Enter** to start collecting metrics. Metrics similar to the following are displayed:

```
Waiting for initial data collection ...

      RtnLine     Time        CPU         RtnLoad     GloRef      GloSet
1.    38.02%      0.03%       10.49%      0.83%       0.05%       0.08%
      SYS.Database.1.INT (CACHESYS)
2.    21.00%      0.08%       49.97%      0.12%       68.73%      7.55%
      DocBook.chapter.CLS (DOCBOOK)
3.    19.68%      0.01%       11.55%      7.16%       28.17%      88.34%
      DocBook.para.CLS (DOCBOOK)
4.    10.93%      0.00%       1.98%       19.64%      0.00%       0.00%
      %cspParser.INT (CACHELIB)
5.    1.99%       0.00%       2.05%       5.14%       0.07%       0.00%
      DocBook.Renderer.CLS (DOCBOOK)
6.    1.75%       0.00%       2.53%       22.46%      0.00%       0.00%
      %CSP.TokenStream.1.INT (CACHELIB)
7.    1.05%       0.00%       1.50%       6.00%       0.69%       0.83%
      DocBook.listitem.CLS (DOCBOOK)
8.    0.84%       0.00%       0.00%       0.01%       0.17%       0.78%
      PROFILE.INT (CACHESYS)
9.    0.76%       0.00%       0.48%       7.91%       0.24%       0.88%
      %Library.GlobalCharacterStream.1.INT (CACHELIB)
10.   0.47%       0.00%       0.48%       4.78%       0.12%       0.00%
      DocBook.block.CLS (DOCBOOK)
11.   0.33%       0.00%       0.51%       1.22%       0.15%       0.13%
      Security.Resources.1.INT (CACHESYS)
Select routine(s) or '?' for more options  N =>
```

4. Flag the routines you want to analyze in more detail. For example, enter `2-3,5,7,10`, then enter `N` or `B` to display other pages so that you can select additional routines.

5. After you select all the routines you want to analyze, enter `Q` to display a message similar to the following:

```
There are 5 routines selected for detailed profiling. You may now
end the routine level collection and start a detailed profiler collection.

WARNING !!

This will have each process on the system gather subroutine level and line
level activity on these routines. Note that this part of the collection may
have a significant effect on performance and should only be run in a test
or development instance of Cache.

Are you ready to start the detailed collection?  Yes =>
```

6. After you press **Enter**, a page similar to the following is displayed:

```
Stopping the routine level Profile collection ...

Loading ^DocBook.chapter.1 in c:\intersystems\cache\mgr\docbook\
Loading ^DocBook.para.1 in c:\intersystems\cache\mgr\docbook\
Loading ^DocBook.Renderer.1 in c:\intersystems\cache\mgr\docbook\
Loading ^DocBook.listitem.1 in c:\intersystems\cache\mgr\docbook\
Loading ^DocBook.block.1 in c:\intersystems\cache\mgr\docbook\

Detail level Profile collection started.

    RtnLine     Routine Name  (Database)
1.  0%          DocBook.Renderer.CLS (DOCBOOK)
2.  0%          DocBook.block.CLS (DOCBOOK)
3.  0%          DocBook.chapter.CLS (DOCBOOK)
4.  0%          DocBook.para.CLS (DOCBOOK)
5.  0%          DocBook.listitem.CLS (DOCBOOK)

Select routine to see details or '?' for more options  R =>
```

7. After you select the routine whose code you want to analyze, a page similar to the following is displayed:

```
Line  RtnLine                Code
1.    0           ;DocBook.chapter.1
2.    0           ;(C)InterSystems, generated for class DocBook.chapter.  Do NOT
3.    0           ;;0032F4FE18715E65;DocBook.chapter
4.    0           ;
5.    0          %1Check(id="",lockonly=0) public {
6.    0              Set exists=($select(id="":0,(+##class(DocBook.chapter).%OnDe
7.    0          %AcquireLock(%this,locktype="") public {
8.    0              Quit ..%LockId(($listget($zobjval(,0,,,,3))),$s($e(locktype)
9.    0          %BMEBuilt(bmeName)
10.   0              Set bmeName = "$chapter"
11.   0              Quit ''$d(^DocBook.blockI("$chapter"))
12.   0          %BindExport(%this,dev,Seen,RegisterOref,AllowedDepth,AllowedCapa
13.   0              i $d(Seen(""_%this)) q 1
14.   0              Set Seen(""_%this)=%this
15.   0              s sc = 1
16.   0              s proporef=$PROPERTY(%this,"book")
17.   0              s proporef=$PROPERTY(%this,"component")
18.   0              s proporef=$PROPERTY(%this,"container")
19.   0              d:RegisterOref InitObjVar^%SYS.BINDSRV(%this)
20.   0              i dev'="" s t=$io u dev i $s($P(dev,":",1)="|TRM|":$$debugPu
21.   0              i AllowedDepth>0 s AllowedDepth = AllowedDepth - 1

Routine DocBook.chapter.1 in DOCBOOK - '?' for options  N =>
```

8

Examining Routine Performance Using ^%SYS.MONLBL

The routine **^%SYS.MONLBL** provides a user interface to the Caché **MONITOR** facility. This utility provides a way to diagnose where time is spent executing selected code in ObjectScript and Caché Basic routines, helping to identify lines of code that are particularly resource intensive. It is an extension of the existing **MONITOR** utility accessed through **^PERFMON** and the %Monitor.System package classes. Because these utilities share the same memory allocation, you can only run one of them at a time on a Caché instance.

This chapter contains the following sections:

8.1 Invoking the Line-by-line Monitoring Routine

If the monitor is not running when you invoke **^%SYS.MONLBL**, the routine displays a warning message and gives you the option to start the monitor or to check memory requirements. For example:

```
%SYS>Do ^%SYS.MONLBL

WARNING ! Starting the line-by-line monitor will enable the
  collection of statistics for *every* line of code executed by
  the selected routines and processes. This can have a significant
  impact on the performance of a system, and it is recommended
  that you do this on a 'test' system.

  The line-by-line monitor also allocates shared memory to track
  these statistics for each line of each routine selected. This is
  taken from the general shared memory already allocated by Cache and
  should be considered if you are using '*' wildcards and trying to
  analyze a large number of routines. If the monitor fails to start due
  to a problem with memory allocation, you may need to increase the
  GenericHeapSize parameter in the system configuration. You may use
  the 'Memory Requirements' option to see how much memory a collection
  would need (without starting the collection).

1.) Start Monitor
```

```
2.) Memory Requirements

Enter the number of your choice:
```

- Enter `1` to begin the dialog to provide the appropriate information to *Start Monitoring*.
- Enter `2` to calculate an estimate of how much memory a collection needs before actually starting the monitor. See the *Estimate Memory Requirements* section for details.

8.1.1 Start Monitoring

You can select which routines and processes to monitor and which metrics to collect. These characteristics of the collection remain until you stop the monitor. You provide monitoring collection information to the routine in the following order:

1. `Routine Names` – Enter a list of routine names to monitor. You can only select routines accessible from your current namespace. Do not use the leading ^ when entering the routine name; the names are case-sensitive. You may use asterisk (*) wild cards to select multiple routines. Press **Enter** twice after entering the last routine name to end the list.
2. `Select Metrics to monitor` – Enter the number of your choice of what type of metrics. The default is `1` for minimal metrics.

   ```
   Select Metrics to monitor
     1) Monitor Minimal Metrics
     2) Monitor Lines (Coverage)
     3) Monitor Global Metrics
     4) Monitor All Metrics
     5) Customize Monitor Metrics

   Enter the number of your choice: <1>
   ```

   A description of what metrics are included for each option follows:

   - *Minimal metrics* — Monitors the metrics described in the following table.

Metric	Description
`Metric#: 34 - RtnLine`	Number of times a routine line is executed
`Metric#: 51 - Time`	Clock time spent in executing that line
`Metric#: 52 - TotalTime`	Total clock time for that line including time spent in subroutines called by that line

     Note: Total Time for Recursive Code
     When a routine contains recursive code, the `TotalTime` counter for the line which calls back into the same subroutine only records the time of the outermost call, which should be, in most cases, the actual time to run the recursive loop. Prior Caché releases accumulated the time for multiple iterations of the same code reporting times that may have seemed too large.

   - *Lines* — Monitors the number of times a routine line is executed (`Metric#: 34 - RtnLine`).
   - *Global metrics* — Monitors several global metrics (`Metric# 1-26,34-36,51,52`).
   - *All metrics* — Monitors all available metrics.

- *Customize metrics* — Allows you to create a customized list of metrics to monitor. You can select any of the standard performance metrics supported by the %Monitor.System package classes. Enter a question mark (?) when prompted for the metric item number to see a list of available metrics. For example:

```
Enter the number of your choice: <1> 5

Enter metrics item number (press 'Enter' to terminate, ? for list)

Metric#: ?
1.) GloRef: global refs
2.) GloSet: global sets
.
.
.
34.) RtnLine: lines of Cache Object Script
.
.
.
51.) Time: elapsed time on wall clock
52.) TotalTime: total time used (including sub-routines)
Metric#:
```

  This example does not show the full list; it is best for you to retrieve the current list when you run the routine. See the Line-by-line Monitor Programming Interface section for a method of retrieving the list.

  Note: For all collections, the number of routine lines and time (minimal metrics) are *always* collected.

3. `Select Processes to monitor` – Enter the number of your choice as it appears in the menu. The default is `1` for all processes.

```
Select Processes to monitor
  1.) Monitor All Processes
  2.) Monitor Current Process Only
  3.) Enter list of PIDs

Enter the number of your choice: <1>
```

  ^%SYS.MONLBL does not currently provide a list or a way to select PIDs; however, you can use the **^%SS** utility or the **System** > **Processes** page of the Management Portal to find specific process ID numbers.

```
Enter the number of your choice: <1> 3

Enter PID (press 'Enter' to terminate)

PID: 1640
PID: 2452
PID:
```

  Press **Enter** twice after entering the last process ID to end the list.

Once you provide the necessary information, **^%SYS.MONLBL** allocates a special section of shared memory for counters for each line per routine, and notifies the selected processes that monitoring is activated.

Note: Since shared counters may be updated simultaneously by multiple processes and/or running processes may not start counting at exactly the same moment, there may be a slight loss of precision in the counters, resulting in counts being lower than expected.

```
Monitor started.

Press RETURN to continue ...
```

After starting the line-by-line monitor, the routine displays a more extensive menu. The Line-by-line Monitoring Options section describes each option on this extended menu.

8.1.2 Estimate Memory Requirements

Before starting the monitoring process you can use this utility to estimate how much memory a collection requires. Typically, there is sufficient shared memory available for monitoring a few routines. However, if you want to monitor hundreds or more routines, use this option to help determine memory needs.

The routine and metrics prompts are identical to those for the `Start Monitor` choice. After you select the routines to monitor and the metrics to gather, the utility displays the number of pages of memory required to monitor this collection and the number of pages available. It also tells you to increase the size of the `GenericHeapSize` parameter if necessary.

You can maintain the **gmheap** (GenericHeapSize) setting from the **System** > **Configuration** > **Advanced Memory Settings** page of the Management Portal.

The following is an example that estimates the memory requirements for monitoring eight selected metrics for all routines that begin with `JRN`:

```
Enter the number of your choice: 2

Enter routine names to be monitored on a line by line basis.
Patterns using '*' are allowed.
Enter '?L' to see a list of routines already selected.
Press 'Enter' to terminate input.

Routine Name: JRN*                      (22 routines added to selection.)
Routine Name:

Select Metrics to monitor
  1) Monitor Minimal Metrics
  2) Monitor Lines (Coverage)
  3) Monitor Global Metrics
  4) Monitor All Metrics
  5) Customize Monitor Metrics

Enter the number of your choice: <1> 5

Enter metrics item number (press 'Enter' to terminate, ? for list)

Metric#: 1 - GloRef
Metric#: 2 - GloSet
Metric#: 3 - GloKill
Metric#: 25 - JrnEntry
Metric#: 34 - RtnLine
Metric#: 35 - RtnLoad
Metric#: 51 - Time
Metric#: 52 - TotalTime
Metric#:

9 page(s) of memory required.
82 page(s) of memory available.

The GenericHeapSize parameter can be increased if more memory is needed.
Pages are each 64kb of memory.

Press RETURN to continue ...
```

You may adjust your memory if that is required for your selected collection and then choose to *Start Monitoring* from the original menu.

8.2 Line-by-line Monitoring Options

If you invoke **^%SYS.MONLBL** while the monitor is running you have the following menu options:

```
Line-by-Line Monitor

1.) Stop Monitor
2.) Pause Monitor / Resume Monitor
3.) Clear Counters
4.) Report - Detail
5.) Report - Summary
6.) Report - Delimited (CSV) Output
7.) Report - Procedure Level

Enter the number of your choice:
```

The first three options are fairly self-explanatory:

- `Stop Monitor` — Stops all **^%SYS.MONLBL** monitoring; deallocates the counter memory and deletes collected data.
- `Pause Monitor` — Pauses the collection and maintains any collected data. This may be useful when viewing collected data to ensure that counts are not changing as the report is displayed. This option only appears if the monitor is running.

  `Resume Monitor` — Resumes collection after a pause. This option only appears if you paused the monitor.
- `Clear Counters` — Clears any collected data, but continues monitoring and collecting new data.

The *Report Line-by-line Statistics* section explains the four report options in more detail.

8.2.1 Report Line-by-line Statistics

When you choose to report the statistics of the metrics that have been collecting (options 4–7), you then provide information about how you want the routine to report the statistics.

You have four types of reports to choose from:

- `Detail` — Generates a report of the selected metrics for each line in the selected routines. The report accumulates and displays totals for each of the performance columns.
- `Summary` — Generates a report of summary information for each selected routine including coverage and time. The report orders the routines by coverage percentage.
- `Delimited (CSV) Output` — Generates the same report information as the detail report, but presents it as comma-delimited output facilitating its import into a spreadsheet.
- `Procedure Level` — Generates a report of selected metrics at a subroutine level within the selected routines. Caché allows you to profile usage at the level of individual subroutines, procedures, and functions. You can quickly see which subroutines are taking up the most time to run to analyze and improve performance.

Depending on which type of report you choose, you select how you want to display the information:

1. If you choose the detail or summary report, you can also choose if you want to include a coverage analysis for the lines executed in each routine you select. For example:

   ```
   Enter the number of your choice: 4
   Include Coverage Analysis summary (Y/N)? y
   ```

2. Next, for all but the summary report, select one or more routines from the list of monitored routines that have statistics available; enter an asterisk (*) for all available routines. For example:

```
The following routines have been executed during the run,
and have detail statistics available for them.
1) JRNDUMP
2) JRNOPTS
3) JRNSTART
4) JRNSWTCH
5) JRNUTIL
6) JRNUTIL2

Enter list of routines, or * for all
Routine number (*=All)? * - All
```

3. If you are entering routine names, after entering the last routine, press **Enter** again to end the list. For example:

```
Enter list of routines, or * for all
Routine number (*=All)? 1 - JRNDUMP
Routine number (*=All)? 2 - JRNOPTS
Routine number (*=All)? 5 - JRNUTIL
Routine number (*=All)?
FileName:
```

4. You can enter a file name or a full directory path for the output. You can instead enter nothing and press **Enter** to display the report on your terminal.

   If you enter a file name but not a path, **%SYS.MONLBL** creates the file in the directory of the current namespace's default database for globals. For example, if running **%SYS.MONLBL** in the USER namespace:

```
FileName: monlbl_JRN_dtl.txt
```

   Creates a file for the report in *install-dir**mgr**user* named monlbl_JRN_dtl.txt.

5. Press **Enter** to initiate the reporting of the metrics you are collecting in the format you have chosen.

The *Sample Line-by-line Monitor Reports* section shows examples of each reporting option.

8.3 Sample Line-by-line Monitor Reports

This section contains samples of the various reports the **^%SYS.MONLBL** routine generates:

- Line-by-line Detail Report
- Line-by-line Summary Report
- Line-by-line Delimited Output Report
- Line-by-line Procedure Level Report

8.3.1 Line-by-line Detail Report

The following is an example of reporting the detail of the minimal metrics of selected journal utilities including the coverage analysis. The report is sent to the monlbl_JRN_dtl.txt file, a portion of which is displayed.

```
Line-by-Line Monitor

1.) Stop Monitor
2.) Pause Monitor
3.) Clear Counters
4.) Report - Detail
5.) Report - Summary
6.) Report - Delimited (CSV) Output
```

```
7.) Report - Procedure Level

Enter the number of your choice: 4
Include Coverage Analysis summary (Y/N)? y

The following routines have been executed during the run,
and have detail statistics available for them.
1) JRNDUMP
2) JRNOPTS
3) JRNSTART
4) JRNSWTCH
5) JRNUTIL
6) JRNUTIL2

Enter list of routines, or * for all
Routine number (*=All)? 1 - JRNDUMP
Routine number (*=All)? 2 - JRNOPTS
Routine number (*=All)? 5 - JRNUTIL
Routine number (*=All)?
FileName: monlbl_JRN_dtl.txt


Press RETURN to continue ...
```

For each line of the selected routine(s), the report displays a line number, the counts for each metric, and the text of that line of code (if source code is available). If you requested coverage analysis, it displays after each selected routine.

```
Routine ^JRNDUMP ...

Line    RtnLine       Time  TotalTime
1             0   0          0        JRNDUMP ;dump the contents...
2             0   0          0         /*
.
.
.
85            0   0          0         n (l,usecluster)
86            3   0.000016   0.000016  i +$g(usecluster) d showlistclu(.l) q
87            3   0.000008   0.000008  s diroff=((3+12+1)+10+1)
88            3   0.000072   0.000072  s i="" f  s i=$o(l(i)) q:i=""  d
89           11   0.001542   0.001542  . w /cup(i+3,1),?3,$S($F(l(i),";")...
90           11   0.028125   0.028220  . w ?(3+12+1),l(i,"info"),?diroff...
91           11   0.000378   0.000895  . w $$GJrnPrefix(l(i))
92            3   0.000027   0.000027  q
93            0   0          0        listjrn(f,list,n) ;list at most...
.
.
.
Total       582  17.258963

Total Lines = 579
Total Lines Hit = 100
Coverage Percentage = 17.27%
```

This is a partial display of one selected routine.

8.3.2 Line-by-line Summary Report

The following is an example of reporting a summary of the minimal metrics of selected journal utilities including the coverage analysis. The report is sent to the monlbl_JRN_summ.txt file, a portion of which is displayed.

```
Line-by-Line Monitor

1.) Stop Monitor
2.) Pause Monitor
3.) Clear Counters
4.) Report - Detail
5.) Report - Summary
6.) Report - Delimited (CSV) Output
7.) Report - Procedure Level

Enter the number of your choice: 5
Include Coverage Analysis summary (Y/N)? Y
FileName: monlbl_JRN_summ.txt


Press RETURN to continue ...
```

The report shows each selected routine with a summary of lines, coverage, and time. The routines with the highest coverage percentage appear first in the list.

```
Routine              Lines   LinesHit    Percent    RtnLine       Time
JRNOPTS                109         60     55.05%        155  14.172230
JRNSWTCH               249         58     23.29%         69   0.926131
JRNDUMP                579        100     17.27%        582  17.265002
JRNSTART               393         23      5.85%         23   0.005541
JRNUTIL                872         39      4.47%         39   0.116995
JRNUTIL2               276          8      2.90%         56   0.006056
JRNCHECK                18          0      0.00%
JRNCLFOR               416          0      0.00%
JRNCLUREST             193          0      0.00%
JRNCLUREST2            229          0      0.00%
JRNINFO                263          0      0.00%
JRNMARK                195          0      0.00%
JRNRESTB              1315          0      0.00%
JRNRESTC              1245          0      0.00%
JRNRESTC2              540          0      0.00%
JRNRESTCHELP           122          0      0.00%
JRNRESTD               445          0      0.00%
JRNRESTO               859          0      0.00%
JRNROLL                827          0      0.00%
JRNSTAT                 62          0      0.00%
JRNSTOP                119          0      0.00%
JRNWUTL                235          0      0.00%

TOTAL 22 rtns         9561        288      3.01%        924  31.591955
```

This is the complete sample report.

8.3.3 Line-by-line Delimited Output Report

This example reports the delimited detail of the minimal metrics of selected journal utilities. The report is sent to the monlbl_JRN_csv.txt file, a portion of which is displayed:

```
Line-by-Line Monitor

1.) Stop Monitor
2.) Pause Monitor
3.) Clear Counters
4.) Report - Detail
5.) Report - Summary
6.) Report - Delimited (CSV) Output
7.) Report - Procedure Level

Enter the number of your choice: 6

The following routines have been executed during the run,
and have detail statistics available for them.
1) JRNDUMP
2) JRNOPTS
3) JRNSTART
4) JRNSWTCH
5) JRNUTIL
6) JRNUTIL2

Enter list of routines, or * for all
Routine number (*=All)? * - All
FileName: monlbl_JRN_csv.txt


Press RETURN to continue ...
```

For each line of the selected routine(s), the report displays the routine name, line number, the counts for each metric, and the text of that line of code (if source code is available) all delimited by a comma. The source code line is contained within quotes.

```
Routine,Line,RtnLine,Time,TotalTime,Code
JRNDUMP,1,0,0,0,"JRNDUMP ;dump the contents of a journal file ;
,2,0,0,0," /*"
.
.
.
JRNDUMP,85,0,0,0," n (l,usecluster)"
JRNDUMP,86,3,0.000016,0.000016," i +$g(usecluster) d showlistclu(.l) q"
JRNDUMP,87,3,0.000008,0.000008," s diroff=((3+12+1)+10+1)"
JRNDUMP,88,3,0.000072,0.000072," s i="""" f  s i=$o(l(i)) q:i="""  d"
JRNDUMP,89,11,0.001542,0.001542," . w /cup(i+3,1),?3,$S($F(l(i),"";""):$E(l(i),...
JRNDUMP,90,11,0.028125,0.028220," . w ?(3+12+1),l(i,""info""),?diroff...
JRNDUMP,91,11,0.000378,0.000895," . w $$GJrnPrefix(l(i))"
JRNDUMP,92,3,0.000027,0.000027," q"
JRNDUMP,93,0,0,0,"listjrn(f,list,n) ;list at most n journal files...
.
```

This is a partial display of one selected routine.

8.3.4 Line-by-line Procedure Level Report

The following is an example of reporting the detail of the minimal metrics of selected journal utilities by subroutine function. The report is sent to the monlbl_JRN_proc.txt file, a portion of which is displayed.

```
Line-by-Line Monitor

1.) Stop Monitor
2.) Pause Monitor
3.) Clear Counters
4.) Report - Detail
5.) Report - Summary
6.) Report - Delimited (CSV) Output
7.) Report - Procedure Level

Enter the number of your choice: 7

The following routines have been executed during the run,
and have detail statistics available for them.
1) JRNDUMP
2) JRNOPTS
3) JRNSTART
4) JRNSWTCH
5) JRNUTIL
6) JRNUTIL2

Enter list of routines, or * for all
Routine number (*=All)? * - All
FileName: monlbl_JRN_proc.txt


Press RETURN to continue ...
```

For each subroutine of the selected routine(s), the report displays a tag number, the counts for each metric, and the subroutine label (if source code is available).

```
Routine ^JRNDUMP ...

Tag     RtnLine       Time  TotalTime
1             6   0.000154   0.000154            JRNDUMP
2             0   0          0                   INT
3             0   0          0                   getkey1
4             0   0          0                   progress
5             6   0.000050   0.000050            listhdr
6            21   0.000240   0.000322            showlist
7            20   0.136909   0.330301            listjrn
8             7   0.188435   0.188435            getjrninfo
9             0   0          0                   guijrn
.
.
.
```

This is a portion of the report for one selected routine.

8.4 Line-by-line Monitor Programming Interface

Programmers can also interface with the Caché **MONITOR** facility through the %Monitor.System.LineByLine class. Methods are provided for each menu option in **^%SYS.MONLBL**. For example, start monitoring by calling:

```
Set status=##class(%Monitor.System.LineByLine).Start(Routine,Metric,Process)
```

You can select which routines and processes to monitor. You may also select any of the other standard performance metrics supported by the %Monitor.System classes. Use the **Monitor.System.LineByLine.GetMetrics()** method to retrieve a list of metric names:

```
Set metrics=##class(%Monitor.System.LineByLine).GetMetrics(3)
```

Selecting 3 as the parameter prints a list of all available metrics with a short description for each to the current device.

Stop monitoring by calling:

```
Do ##class(%Monitor.System.LineByLine).Stop()
```

You can retrieve the collected counts using the **%Monitor.System.LineByLine:Result** query, where the counters for each line are returned in $LIST format.

See the %Monitor.System.LineByLine class entry in the online *InterSystems Class Reference* for more details.

9

Monitoring Block Collisions Using ^BLKCOL

A block collision occurs when a process is forced to wait for access to a block. Excessive block collisions slow application performance.

9.1 Using ^BLKCOL

The **^BLKCOL** utility samples block collisions over a specified period (10 seconds by default), recording the latest block collision within a specified interval (10 milliseconds by default) during this time. For each recorded collision, **^BLKCOL** identifies not only the block, but the global involved and its first and last references in the block, as well as the routine and line that created the process attempting to access the block.

Note: The **cstat -D** option, as described in Running cstat with Options in the "Monitoring Caché Using the cstat Utility" appendix of this guide, also samples block collisions, but identifies only the blocks involved.

The output of **cstat -D** is included in the reports generated by the **^pButtons** utility, as described in the Monitoring Performance Using ^pButtons chapter of this guide.

When running **^BLKCOL**, you can specify the following:

- The length of the sampling period in seconds
- The interval between samples in milliseconds
- Whether to collect routine details (default is yes)
- Whether to format the output as
  - a list of the blocks with the highest collision counts (default)
  - a list of all blocks involved in collisions
  - comma-separated values from all block collisions detected, sorted and counted by block number and routine
  - comma-separated values from all block collisions detected, unsorted (raw)
  - a list of collision hot spots in routines
- the number of blocks to display (if applicable)
- whether to send output to a file

9.2 ^BLKCOL Ouput

Use of the **^BLKCOL** utility is shown in the following sample terminal session:

```
%SYS>d ^BLKCOL

Block Collision Analysis

How many seconds should we sample: <10>
How long to wait (ms) between each sample: <10>
Collect routine details? <Y>
Format for 'T'op counts, 'D'isplay all, 'S'orted CSV, 'H'ot spot, or 'R'aw CSV: <T>
Number of blocks to display: <10>
Output to file: <0>

Sampling ... (any key to interrupt)


625 block collisions in 735 samples.

Block # (count) - Global refs (first - last in block) - Routine refs (SFN)

767       (395) in c:\mycache\mgr\user\
  ^acctest - ^acctest(10220," 167") (T/BPtr)
    325 at ^AccessTest+156(4)
    25 at ^AccessTest+121(4)
    24 at ^AccessTest+92(4)
    8 at ^AccessTest+109(4)
    8 at ^AccessTest+127(4)
    4 at ^AccessTest+170(4)
    1 at ^AccessTest+163(4)

3890      (11)  in c:\mycache\mgr\user\
  ^acctest(2552,"  371") - ^acctest(2552,"  38") (Data)
    6 at ^AccessTest+164(4)
    3 at ^AccessTest+163(4)
    1 at ^AccessTest+134(4)
    1 at ^AccessTest+156(4)

15572     (9)   in c:\mycache\mgr\user\
  ^acctest(6980," 4795") - ^acctest(6988," 3259") (Data)
    7 at ^AccessTest+134(4)
    1 at ^AccessTest+164(4)
    1 at ^AccessTest+170(4)

15818     (8)   in c:\mycache\mgr\user\
  ^acctest(9124," 173") - ^acctest(9124," 1743") (Data)
    5 at ^AccessTest+164(4)
    3 at ^AccessTest+170(4)

971       (7)   in c:\mycache\mgr\user\
  ^acctest(484," 3927") - ^acctest(484," 3938") (Data)
    5 at ^AccessTest+170(4)
    2 at ^AccessTest+164(4)

1137      (7)   in c:\mycache\mgr\user\
  ^acctest(756," 4063") - ^acctest(756," 4073") (Data)
    3 at ^AccessTest+109(4)
    2 at ^AccessTest+134(4)
    1 at ^AccessTest+156(4)
    1 at ^AccessTest+163(4)

2999      (7)   in c:\mycache\mgr\user\
  ^acctest(2092,"  666") - ^acctest(2092,"  674") (Data)
    3 at ^AccessTest+170(4)
    1 at ^AccessTest+109(4)
    1 at ^AccessTest+121(4)
    1 at ^AccessTest+134(4)
    1 at ^AccessTest+164(4)

6173      (7)   in c:\mycache\mgr\user\
  ^acctest(3684,"  528") - ^acctest(3684,"  536") (Data)
    3 at ^AccessTest+163(4)
    1 at ^AccessTest+109(4)
    1 at ^AccessTest+156(4)
    1 at ^AccessTest+164(4)
    1 at ^AccessTest+170(4)

14617     (7)   in c:\mycache\mgr\user\
  ^acctest(9688,"   18") - ^acctest(9688,"   26") (Data)
```

```
    4 at ^AccessTest+170(4)
    2 at ^AccessTest+164(4)
    1 at ^AccessTest+134(4)

15282     (7)   in c:\mycache\mgr\user\
  ^acctest(8700," 4889") - ^acctest(8760," 1402") (Data)
    4 at ^AccessTest+170(4)
    3 at ^AccessTest+164(4)
%SYS>d ^BLKCOL

Block Collision Analysis

How many seconds should we sample: <10>
How long to wait (ms) between each sample: <10>
Collect routine details? <Y>
Format for 'T'op counts, 'D'isplay all, 'S'orted CSV, 'H'ot spot, or 'R'aw CSV: <T> H
Number of blocks to display: <10>
Output to file: <0>

Sampling ... (any key to interrupt)


571 block collisions in 768 samples.


Sorted by routine/line that waits for block ownership
_____________________________________________________

(571) AccessTest
    (324) +156^AccessTest :  s @G@($J,node)=$$getdata($E(Str,1,$r(1000)))  ;SMLXXX+, AFH
    (54) +164^AccessTest :  k @G@($J,node)
    (43) +134^AccessTest :  .  k @G@($J,node)
    (31) +92^AccessTest :  .  .  k @G@($j)
    (28) +109^AccessTest :  .  s x=$O(@G@($J,x))

Sorted by routine that owns the block
_____________________________________

(472) AccessTest
    (472) +AccessTest
```

10

Monitoring Performance Using ^pButtons

This chapter describes the **^pButtons** utility, a tool for collecting detailed performance data about a Caché instance and the platform on which it is running. You can send the report it produces to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems. **^pButtons** is similar to Diagnostic Reports (see "Using the Caché Diagnostic Report" in this guide), but focuses on performance data. **^pButtons** works with all versions of Caché since release 5.0.

Note: This utility may be updated between releases. The latest version is available on the WRC distribution site under **Tools**.

You can run the profiles in the Terminal (see Running the ^pButtons Utility in this chapter) or schedule runs using the Task Manager in the Management Portal (see Scheduling the ^pButtons Utility with Task Manager in this chapter). In addition, you can add, modify, and delete profiles using the API that is included with the utility.

Note: For information about a Python-based utility that allows you to extract selected sections from a ^pButtons report and create charts based on the data they contain, written by an InterSystems Senior Technology Architect and available for download on GitHub, see Yape - Yet another pButtons extractor (and automatically create charts) on InterSystems Developer Community.

This chapter includes the following:

- Running the ^pButtons Utility
- Generating ^pButtons Performance Reports
- Scheduling the ^pButtons Utility with Task Manager
- Customizing the ^pButtons Utility
- Performance Reports Created by ^pButtons Utility

Note: For a detailed practical look at the generation and use of **^pButtons** reports to evaluate system performance by an InterSystems senior technology architect, see InterSystems Data Platforms and performance – Part 1 and InterSystems Data Platforms and performance – Part 2 on InterSystems Developer Community.

10.1 Running the ^pButtons Utility

The **^pButtons** utility lets you select one or more profiles to run. (The profiles available vary depending on your Caché release and any customization that has been performed.) Based on the selected profile(s), it generates a set of log files, which are placed in the output directory. By default, the output directory is the manager's directory of the Caché instance,

(*install-dir*\mgr); alternatively, you can specify the output directory as described in the Change Output Directory section in this chapter.

By default, **^pButtons** provides the following profiles:

- 12hours — 12-hour run sampling every 10 seconds
- 24hours — 24-hour run sampling every 10 seconds
- 30mins — 30-minute run sampling every 1 second
- 4hours — 4-hour run sampling every 5 seconds
- 8hours — 8-hour run sampling every 10 seconds
- test — 5-minute TEST run sampling every 30 seconds

To run the **^pButtons** utility:

1. Enter the following command, which is case-sensitive and must be run in the %SYS namespace, in the Terminal:

```
%SYS>do ^pButtons
```

2. From the main menu that is displayed, enter the number of the profile you want to run, or press **Enter** to exit the main menu:

```
Current log directory: c:\intersystems\cache20111\mgr\
Windows Perfmon data will be left in raw format.
Available profiles:
     1 12hours - 12-hour run sampling every 10 seconds
     2 24hours - 24-hour run sampling every 10 seconds
     3 30mins  - 30-minute run sampling every 1 second
     4 4hours  - 4-hour run sampling every 5 seconds
     5 8hours  - 8-hour run sampling every 10 seconds
     6 test    - 5-minute TEST run sampling every 30 seconds

select profile number to run:
```

3. After you enter the profile you want to run, the utility displays information about the data it is collecting:

```
select profile number to run: 1
Collection of this sample data will be available in 1920 seconds.
The runid for this data is 20111007_1041_30mins.
```

The generated log files are located in the output directory. The files are identified by the *runid*, which is uniquely named as follows: *YYYYMMDD_HHMM_profile_name*.log, where *YYYYMMDD_HHMM* is the year, month, day, hour, and minute the utility started to collect data; and *profile_name* is the name of the profile you selected.

After the utility finishes collecting data (that is, at the end of the period of time specified in the profile), you can generate a readable performance report; for information, see the Generating the ^pButtons Performance Reports section in this chapter.

10.1.1 Abort ^pButtons

If you want to stop a running profile, you can abort the collection of data and optionally delete all .log files for the profile with the **$$Stop^pButtons(***runid***)** command. For example, to abort the collection of data for a report identified by the *runid*20111220_1327_12hours and delete all .log files written so far, enter the following command in the Terminal in the %SYS namespace:

```
do Stop^pButtons("20111220_1327_12hours")
```

To stop the job without deleting log files, enter:

```
do Stop^pButtons("20111220_1327_12hours",0)
```

For more information about this command, see `$$Stop^pButtons("runid")` in the Run ^pButtons Programmatically subsection.

Note: You must have permission to stop jobs and delete files.

10.1.2 Run ^pButtons Programmatically

You can run the **^pButtons** utility programmatically using entry points for the start, collect, preview, and stop functions as described in the following table:

Note: You can run multiple profiles concurrently.

Command	Description
`$$run^pButtons("profile")`	Starts the specified *profile*. If successful, returns the *runid*; if unsuccessful, returns `0`.
`$$literun^pButtons("profile")`	Same as `$$run^pButtons("profile")`, *except* that it does not include operating-system data. **Note:** This command is intended for servers that are running multiple instances of Caché, where the operating-system data would be duplicated.
`$$Collect^pButtons("runid")`	Produces a readable HTML performance report file for the specified *runid*. If successful, returns `1` and the report filename; if unsuccessful, returns `0` followed by a carat and the reason for the failure.
`$$Preview^pButtons("runid")`	Produces a readable HTML interim (incomplete) performance report file for the specified *runid*. If successful, returns `1`. If unsuccessful, returns `0` followed by a carat and the reason for the failure.
`$$Stop^pButtons("runid",[0])`	Stops (aborts) **^pButtons** from collecting data for a specified *runid* and by default deletes the associated .log files produced by the utility. To stop data collection without deleting the .log files, include the `0` parameter following the *runid*. If successful, returns: *1:2:3:4_1:2:3:4*; if unsuccessful, returns `0` followed by a carat and the reason for the failure. **Note:** The "successful" return status is made up of two parts separated by an underscore: OS-specific and Caché-specific; within each part, colon-separated values specify: 1. Number of jobs successfully stopped 2. Number of jobs that failed to stop 3. Number of files successfully deleted 4. Number of files not deleted

Command	Description
`$$waittime^pButtons("runid")`	Reports the time until the final HTML file for the specified *runid* will be complete. If the *runid* is finished, returns `ready now`, otherwise returns a string of the form *XX* `hours` *YY* `minutes` *ZZ* `seconds`.

In the following example the *runid*, which is created by the **^pButtons** utility, is obtained programmatically, then tested to determine if a full or interim report has been generated. It determines programmatically that a full report has not been created because the profile has not finished ("`0^not ready`" is returned), but an interim report has been created ("`1`" is returned). Based on this information, you know that an HTML file has been generated.

```
%SYS>set runid=$$run^pButtons("30mins")

%SYS>set sc=$$Collect^pButtons(runid)
pButtons run 20111004_1238_30mins is not yet ready for collection.

%SYS>write sc
0^not ready

%SYS>set sc=$$Preview^pButtons(runid)

%SYS>write sc
1
%SYS>
```

10.2 Generating ^pButtons Performance Reports

The **^pButtons** utility automatically generates a full (complete) readable HTML performance report from the log files produced by the **^pButtons** utility. You can also use the **Preview^pButtons** entry point to produces an interim (incomplete) report using the data that is being collected by the profile you selected when you ran the **^pButtons** utility.

The generated report files are located in the output directory which, by default, is the manager's directory of the Caché instance (*install-dir*\mgr). The files are uniquely identified by names, which are in the following format: *hostname_instance_runid*.html, where *hostname* is the hostname of the system on which the instance of Caché is running; *instance* is the name of the instance for which performance data has been collected; and *runid* is the unique identifier generated when the **^pButtons** utility was run. If the report is an interim report, `_P`*n* is appended to the file name, where `P` identifies it as a preliminary report and *n* is the number of the preliminary report.

10.3 Scheduling the ^pButtons Utility with Task Manager

This section provides examples using the **Task Manager** page (**System Operation** > **Task Manager**) in the Management Portal to schedule **^pButtons** to run.

Note: The examples describe only the fields that are required. You can edit other fields as desired.

Example 1: Weekly 24-Hour Run

In this example, a task is created to schedule the **^pButtons** utility to run a profile named 24hours (which collects performance data for 24 hours) every Thursday at 09:00. To schedule this task use the procedure described in Schedule Task Manager in the "Managing Caché" chapter of *Caché System Administration Guide*.

In this example, first create the task to run **^pButtons**:

1. From the **Task Manager** page (**System Operation** > **Task Manager**), choose the **New Task** option to start the **Task Scheduler Wizard**. Then enter the following information in the specified fields:

- **Task name** — enter `24HourRun`.
- **Description** — enter `Start 24-hour ^pButtons Run`.
- **Namespace to run task in** — select **%SYS** from the drop-down list.
- **Task type** — select **RunLegacyTask** from the drop-down list.

  In the **ExecuteCode** textbox, enter the following code:

  ```
  do run^pButtons("24hours")
  ```

- **Output file** — leave blank; the task has no output (see Change Output Directory for information on customizing the **^pButtons** output directory).

2. Click **Next**. Then enter the following information in the specified fields:
   - **How often ...** — choose **Weekly** from the drop-down list.

     Select the **Thursday** check box.
   - **Start Date** — enter the start date in the text box.

     Click **Run once at this time:** and enter 09:00:00 in the text box.
3. Click **Finish**.

Example 2: Daily 2–Minute Run

In this example, a task is created to schedule the **^pButtons** utility to run a profile named 2mins (which collects performance data for two minutes) every day at 12:00. To schedule this task use the procedure described in Schedule Task Manager in the "Managing Caché" chapter of *Caché System Administration Guide*.

In this example, first create the task to run **^pButtons**:

1. From the **Task Manager** page (**System Operation** > **Task Manager**), choose the **New Task** option to start the **Task Scheduler Wizard**. Then enter the following information in the specified fields:
   - **Task name** — enter `2MinRun`.
   - **Description** — enter `Start 2-minute ^pButtons Run`.
   - **Namespace to run task in** — select **%SYS** from the drop-down list.
   - **Task type** — select **RunLegacyTask** from the drop-down list.

     In the **ExecuteCode** textbox, enter the following code:

     ```
     do run^pButtons("2mins")
     ```

   - **Output file** — leave blank; the task has no output (see Change Output Directory for information on customizing the **^pButtons** output directory).
2. Click **Next**. Then enter the following information in the specified fields:
   - **How often ...** — choose **Daily** from the drop-down list.
   - **Start Date** — enter the start date in the text box.

     Click **Run once at this time:** and enter 12:00:00 in the text box.
3. Click **Finish**.

10.4 Customizing the ^pButtons Utility

This section describes the tasks you can accomplish with the API:

- Change Output Directory
- Get Version Information
- Manipulate Profiles

10.4.1 Change Output Directory

The default output directory for both the log files and the resulting HTML report file is the manager's directory of the Caché instance (*install-dir*\mgr) for which you are running the **^pButtons** utility. You can change the default directory using the commands described in the following table.

Note: These commands do not affect currently running profiles, whether or not the HTML report files have been produced; that is, no files associated with currently running profiles are moved to the new output directory.

Command	Description
`do setlogdir^pButtons("directory")`	Sets the pathname of output directory to *directory*; if the does not exist, it is created. **Note:** If you do not specify an absolute pathname (for example, `C:\pButtonReports`), the directory is assumed to be relative to the manager's directory.
`do clrlogdir^pButtons()`	Resets the output directory pathname to the default manager's directory.

10.4.2 Get Version Information

You can find the current version of the **^pButtons** utility using the following commands:

- `write $$version^pButtons()`
- `set ver=$$version^pButtons()`

10.4.3 Manipulate Profiles

You can use the APIs described in the following sections to manipulate the profile definitions.

- Create New Profiles
- Edit Profiles
- Copy Profiles
- Delete Profiles

10.4.3.1 Create New Profiles

You can create a new profile with the following API command:

```
set rc=$$addprofile^pButtons("profilename","description",interval,count)
```

where *profilename*, which is required, must be unique and cannot contain spaces or white-space characters; *description*, which is required, should be meaningful because it displayed in the menu when you run the **^pButtons** utility; *interval*, which is required, is the frequency with which to run each sample, in seconds (in the range of 1 second to 300 seconds); and *count*, which is required, is the number of times to run the profile.

Note: An interval of 1 second is only allowed if the profile duration is an hour or less.

For example, to create a profile named 2minrun that runs a sampling every 10 seconds until it runs 12 samplings (for a total of 120 seconds, or two minutes), enter the following:

```
set rc=$$addprofile^pButtons("2minrun","A 2-minute run sampling every 10 seconds",10,12)
```

The next time you run the **^pButtons** utility, the list of profiles includes the following profile name and description:

```
2minrun     A 2-minute run sampling every 10 seconds
```

Generate Profile

Alternatively, you can quickly generate new profiles (with a meaningful name and description) with the following API command:

```
set rc=$$genprofile^pButtons("duration"[,interval])
```

where *duration* (*interval* * *count*), which is required, must be in the format "*hh:mm*", "*hh:*", or *mm*; and *interval*, which is optional, is the frequency with which to run each sample.

Note: The maximum *duration* is 24 hours (86400 seconds); if you specify a longer duration, **^pButtons** reduces it to 24 hours. The *duration* must be double-quoted only if it contains a colon (:); the colon denotes hours.

The minimum *interval*, if specified, is 2 seconds, unless the duration (that is, *interval* * *count*) is less than one hour, in which case the minimum *interval* is 1 second. If you specify an invalid *interval*, **^pButtons** increases it to the required minimum. If the *interval* is not specified, it defaults to 10 seconds.

For example, to generate a profile named 12hours (with a generated profile name and description) that runs samples every 5 minutes (300 seconds) over 12 hours, enter the following:

```
set rc=$$genprofile^pButtons("12:",300)
```

In addition, to generate a profile named 90mins that runs samples every 10 seconds for 90 minutes, enter the following:

```
set rc=$$genprofile^pButtons(90)
```

The next time you run the **^pButtons** utility, the list of profiles includes the following profile names and descriptions:

```
12hours     A 12 hour run sampling every 300 seconds
90mins      A 90 minute run sampling every 10 seconds
```

10.4.3.2 Edit Profiles

You can edit an existing profile (except for the predefined “test” profile) with the following API command:

```
set rc=$$editprofile^pButtons("profilename","description",[interval],[count])
```

where *profilename*, which is required, must be unique and cannot contain spaces or white-space characters; *description*, which is required, should be meaningful because it displayed in the menu when you run the **^pButtons** utility; *interval*,

which is optional, is the time to run each sample, in seconds (in the range of 2 seconds to 300 seconds); and *count*, which is optional, is the number of times to run the profile.

Note: The arguments are positional; if, for example, to edit the *count* argument (and keep the value specified in the *interval* argument), you must include the comma separator, as follows: `set rc=$$editprofile^pButtons("2minrun","A 5-minute run sampling every 30 seconds",,50)`.

If the duration exceeds 24 hours (86400 seconds), it is automatically reduced to 24 hours.

For example, to modify the 2minrun profile to run a sampling every 30 seconds until it runs 10 samplings (for a total of 300 seconds, or five minutes), enter the following:

```
set rc=$$editprofile^pButtons("2minrun","A 5-minute run sampling every 30 seconds",30,10)
```

The next time you run the **^pButtons** utility, the list of profiles includes the following profile name and description:

```
2minrun     A 5-minute run sampling every 30 seconds
```

10.4.3.3 Copy Profiles

You can copy an existing profile to a file with a different name with the following API command:

`set rc=$$copyprofile^pButtons("`*`sourceprofilename`*`","`*`targetprofilename`*`")`

where *sourceprofilename*, which is required, is the name of an existing profile, and *targetprofilename*, which is required, must be unique and cannot contain spaces or white-space characters.

For example, to make a copy of the 2minrun profile, enter the following:

```
set rc=$$copyprofile^pButtons("2minrun","5minrun")
```

The next time you run the **^pButtons** utility, the list of profiles includes the following profile names and description:s

```
2minrun     A 2-minute run sampling every 30 seconds
5minrun     A 2-minute run sampling every 30 seconds
```

You can now edit the new profile as described in Edit Profiles in this section of the guide.

10.4.3.4 Delete Profiles

You can delete existing profiles (except for the predefined “test” profile) with the following API command:

`set rc=$$delprofile^pButtons("`*`profilename`*`")`

where *profilename*, which is required and must be double-quoted, is the name of the profile you want to delete.

For example, to delete the 2minrun profile, enter the following:

```
set rc=$$delprofile^pButtons("2minrun")
```

The next time you run the **^pButtons** utility, the list of profiles does not include 2minrun profile.

10.5 Performance Reports Created by ^pButtons Utility

The **^pButtons** utility generates platform-specific reports as described in this chapter. The report is divided into sections, as illustrated in the following listing:

```
Configuration

CACHE20161 on machine testsystem

Customer: InterSystems Development
License : 123456

Caché Version String: Cache for Windows (x86-32) 2016.1 (Build 656) Thu Mar 17 2016 17:51:22 EDT
------------------------------------------------------------
Profile

Profile run "test" started at 10:07 on Jun 01 2016.
Run over 10 intervals of 30 seconds.
------------------------------------------------------------
license

Product=Enterprise
License Type=Concurrent User
Server=Multi
Platform=Heterogeneous
Licensed Users=1000
Licensed CPUs=16
.
.
.
------------------------------------------------------------
End of Caché Performance Data Report
```

The tables in this section describe the sections of each platform-specific report. The sections are listed alphabetically in each table to help you find a specific section more easily. Data that is collected only once is flagged with an asterisk (*). The rest of the data is collected throughout the profile run.

For descriptions of the platform-specific data, see the following tables:

- Caché Performance Data Report for Microsoft Windows Platforms
- Caché Performance Data Report for Apple macOS Platforms
- Caché Performance Data Report for IBM AIX® Platforms
- Caché Performance Data Report for Red Hat Linux/SuSE Linux Platforms

Note: In all of the following tables, data marked with * is collected once per run.

Table 10–1: Caché Performance Data Report for Microsoft Windows Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
Configuration *	Caché instance name and hostname from the server, the full Caché version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.

Section	Description
cstat -c	Four samples taken at even intervals over the course of the run using the command **.\bin\cstat -s -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all CACHE.DAT and CACHE.EXT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide.
cstat -D	Eight samples taken at even intervals over the course of the run using the command **ccontrol stat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide. For information about monitoring block collisions using the **^BLKCOL** utility, see the "Monitoring Block Collisions Using ^BLKCOL" chapter of this guide.
license *	Caché license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
mgstat	Caché-specific data taken over the course of the run using the **^mgstat** utility. See the Monitoring Performance Using ^mgstat section of the *Caché Monitoring Guide*.

Section	Description

perfmon	Output from the Microsoft Windows **perfmon** utility.
	
	The default presentation of Microsoft Windows **perfmon** data is raw format. The format can be switched to processed, which removes the repeated server name and splits the datetime column into separate columns, to improve readability.
	
	The following functions allow the querying and updating of the flag that determines whether the **perfmon** data is manipulated or not:
	
	`set rc=$$setperfmonpostproc^pButtons(<onoroff>)`
	
	where `onoroff` can be `1` (on) or `0` (off), or the non-case-sensitive words `"on"` or `"off"`.
	
	A return code of `1` indicates successful update of the flag, `0` indicates a failed update, and `-1` indicates a non-Windows platform.
	
	To determine the current format (raw or processed):
	
	`set status=$$getperfmonpostproc^pButtons()`
	
	A return code of `1` indicates processed format, `0` indicates raw format.
	
	In addition, the current status of the flag is reported prior to the profile menu display in the interactive run of **^pButtons**.
	
	By default, **perfmon** monitors the counter definitions specified in the default pbctrs.txt file. To monitor previously defined **perfmon** counters, import the definition into **^pButtons** using:
	
	`write $$importctrs^pButtons(`*`WindowsCtrName`* `[,`*`pButtonsCtrName`*
	`[,`*`pButtonsFileName`*`]])`
	
	A return code of `0` indicates success and a negative number followed by a reason string indicates failure. Duplicate pButtons counter names are not allowed. If necessary, **^pButtons** generates both the internal counter name and file name.
	
	To change the default pButtons counter definition to an exiting definition, use:
	
	`write $$setctrdefault(`*`pButtonsCtrName`*`)`
	
	Return code of `1` indicates success and `0` followed by a reason string indicates failure. If an invalid counter is specified, the builtin defaualt is set.
	
	To reset the default pButtons counter definition, use:
	
	`do clrctrdefault^pButtons()`
	
	To associate a specific pButtons counter definition with an existing profile, use:
	
	`write $$addctrtoprofile(`*`ProfileName`*`,`*`pButtonsCtrName`*`)`
	
	Return code of `1` indicates success and `0` followed by a reason string indicates failure. If either the profile or the counter definition do not exist, the command is not run.

Profile *	Information about the **^pButtons** profile that created this log.

Section	Description
tasklist *	Output from the **tasklisk -V** command; a list of all processes running on the system at the start of the profile run.
Windows info *	Output from the **systeminfo** command, including the Windows version (excluding hotfix information) and hardware information; for example, processor count, memory installed, and memory used.

Table 10–2: Caché Performance Data Report for Apple macOS Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
Configuration *	Caché instance name and hostname from the server, the full Caché version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.
cstat -c	Four samples taken at even intervals over the course of the run using the command **ccontrol stat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all CACHE.DAT and CACHE.EXT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide.
cstat -D	Eight samples taken at even intervals over the course of the run using the command **ccontrol stat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide. For information about monitoring block collisions using the **^BLKCOL** utility, see the "Monitoring Block Collisions Using ^BLKCOL" chapter of this guide.

Section	Description
ipcs *	Interprocess communication configuration information, including shared memory, semaphores, and message queues; output from **ipcs -a** command.
license *	Caché license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
MacOS Info *	OS version and hardware information. Output from the **sw_vers**, **uname -a**, **mount**, and **netstat** commands.
mgstat	Caché-specific data taken over the course of the run using the **^mgstat** utility. See the Monitoring Performance Using ^mgstat section of the *Caché Monitoring Guide*.
Profile *	Information about the **^pButtons** profile that created this log.
ps:	Four samples taken at even intervals over the course of the run using the command **ps -eflv**.
sar -d	Disk (block) device throughput and latency statistics.
sar -g	Page out rates.
sar -n DEV	Network device throughput.
sar -n EDEV	Network device error rates.
sar -p	Page in and page fault rates.
sar -u	CPU usage statistics.
sysctl -a *	Kernel and system parameter settings.
vm_stat *	memory page information.

Table 10–3: Caché Performance Data Report for IBM AIX® Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
AIX info *	Output from the **oslevel**. **uname -a**, **prtconf**, and **lspv** commands
Configuration *	Caché instance name and hostname from the server, the full Caché version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.
cpu type *	Information on processors installed and whether or not SMT is enabled; output from **lsattr -El proc0**.

Section	Description
cstat -c	Four samples taken at even intervals over the course of the run using the command **ccontrol stat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all CACHE.DAT and CACHE.EXT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide.
cstat -D	Eight samples taken at even intervals over the course of the run using the command **ccontrol stat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide. For information about monitoring block collisions using the **^BLKCOL** utility, see the "Monitoring Block Collisions Using ^BLKCOL" chapter of this guide.
df -k *	Information about mounted file systems, including mount points, logical volumes, and free space; output from **df -k** command.
filesystems *	Current /etc/filesystems file.
ioo -a *	Current values of I/O tunable parameters; output from **ioo -a** command. Included *only* if the user initiating the **^pButtons** profile run has root access.
iostat -DIT	Long listing of extended disk/device statistics with sample time for IBM AIX® 5.3 and newer; output from **iostat -DIT** command. Information varies for releases before IBM AIX® 5.3.
ipcs *	Interprocess communication configuration information, including shared memory, semaphores, and message queues; output from **ipcs -a** command.
license *	Caché license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.

Section	Description
mount *	Information on all file systems and their mount options.
mgstat	Caché-specific data taken over the course of the run using the **^mgstat** utility. See the Monitoring Performance Using ^mgstat section of the *Caché Monitoring Guide*.
Profile *	Information about the **^pButtons** profile that created this log.
ps:	Four samples taken at even intervals over the course of the run using the command **ps aux**.
sar -d	Included *only* if the user initiating the**^pButtons** profile run has root access and /usr/sbin/sar exists.
sar -r	Included *only* if the user initiating the**^pButtons** profile run has root access and /usr/sbin/sar exists.
sar -u	CPU statistics that includes micropartitioning information if used. Included *only* if the user initiating the**^pButtons** profile run has root access and /usr/sbin/sar exists.
vmo –a	Current values of virtual memory tunable parameters; output from **vmo -a** command. Included *only* if the user initiating the**^pButtons** profile run has root access.
vmstat -s *	Absolute counts of virtual memory statistics, including total page ins and page outs.
vmstat -t	Virtual memory and CPU (paging, queuing, and CPU) statistics with timestamps.
vmstat -v *	Samples virtual memory statistics, including free pages, **pbuf** usage, and **fsbuf** usage.

Table 10–4: Caché Performance Data Report for Red Hat Linux/SuSE Linux Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
Configuration *	Caché instance name and hostname from the server, the full Caché version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.

Section	Description
cstat -c	Four samples taken at even intervals over the course of the run using the command **ccontrol stat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all CACHE.DAT and CACHE.EXT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide.
cstat -D	Eight samples taken at even intervals over the course of the run using the command **ccontrol stat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **cstat** utility, see the "Monitoring Caché Using the cstat Utility" appendix of this guide. For information about monitoring block collisions using the **^BLKCOL** utility, see the "Monitoring Block Collisions Using ^BLKCOL" chapter of this guide.
df -k *	Information about mounted file systems, including mount points, logical volumes, and free space; output from **df -k** command.
free -m	Memory usage statistics in MB (**-m**).
iostat	CPU and disk throughput.
license *	Caché license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
mgstat	Caché-specific data taken over the course of the run using the **^mgstat** utility. See the Monitoring Performance Using ^mgstat section of the *Caché Monitoring Guide*.
Profile *	Information about the **^pButtons** profile that created this log.
ps:	Four samples taken at even intervals over the course of the run using the command **ps -efly**.

Section	Description
sar -d	Disk (block) device throughput and latency statistics.
sar -u	CPU usage statistics include **iowait** percentage.
vmstat -n	CPU, queuing, paging statistics. Only one header is printed (**-n**) .

11

Monitoring Performance Using ^mgstat

This chapter describes the **^mgstat** utility, a tool for collecting basic performance data. The utility works with all versions of Caché since release 5.0.

Note: This utility may be updated between releases. Contact the InterSystems Worldwide Response Center (WRC) for information about downloading newmgstat.xml from ftp://ftp.intersys.com/pub/performance/.

The calling sequence uses the following arguments:

Argument	Description
sample time	Required. This argument specifies the frequency (in seconds) for sampling counters; defaults to 2 seconds if *sample time* is not specified. **Note:** If you specify a *sample time* greater than 10 seconds, **^mgstat** reduces it to 10 seconds. See the *number of samples* argument in this table.
number of samples	Required. This argument specifies the maximum number of samples to be obtained; defaults to 10 iterations if *number of samples* is not specified. **Note:** If **^mgstat** reduces the *sample time*, it increases the specified *number of samples* to ensure that the duration (*sample time* * *number of samples*) of the run is effectively the same as it would have been if none of the arguments were modified.
filename	Optional. This argument is used only by other WRC performance tools.
page length	Optional. If you run **^mgstat** interactively, this argument specifies the number of lines to display before the header rows are repeated. **Note:** This argument is ignored when you run **^mgstat** as a background job. If you specify a *page length* less than 5 lines, **^mgstat** increases it to 5 lines; if you omit this argument or specify a *page length* of 0, **^mgstat** displays the header once, at the beginning of the page.

For example, running **^mgstat** as a background job to specify that file samples be obtained every 5 seconds until 17280 samplings are obtained (in the Terminal, from the %SYS namespace), enter the following:

```
%SYS>JOB ^mgstat(5,17280)
```

Alternatively, running **^mgstat** interactively to specify the same samplings, and to redisplay the headings after each 10 rows of data, enter the following:

```
%SYS>DO ^mgstat(5,17280,,10)
```

By default **^mgstat** generates a filename based on the *server name*, *configuration name*, and *date and time*, with the "mgst" extension, which is recognized by an analyzer tool written in Microsoft Excel that aids graphing of the data. By default, the file is located in the manager's directory of the Caché instance (*install-dir*\mgr); if, however, the output directory has been changed through the **^pButtons** utility (see Change Output Directory in the "Monitoring Performance Using ^pButtons" chapter in this guide), **^mgstat** uses that output directory.

Note: The mgst file is also generated when you run the **^pButtons** utility (see the "Monitoring Performance Using ^pButtons" chapter in this guide.

To ensure minimal impact on system performance, the **^mgstat** utility extracts various counter information from shared memory. If the utility is running and an apparent performance issue occurs, data is available to help you investigate the problem; for assistance with your analysis, contact the InterSystems Worldwide Response Center (WRC), which can provide tasks that automate both the running of **^mgstat** and the purging of files.

Most of the reported data is averaged in per-second values, except as noted in the table below. The generated output file is in a readable, comma-separated value (CSV) format, which is more easily interpreted with a spreadsheet tool such as Microsoft Excel. The first line of the file is a header line which includes the filename and the utility version. as well information about buffer allocation and the version of the product being monitored. The number of columns of data depends on the version of the product: the first two columns are the date and time; the remaining columns are:

Column	Description	Notes
Glorefs	Global references (database accesses). Indicates the amount of work that is occurring on behalf of the current workload; although global references consume CPU time, they do not always require physical reads because of the buffer pool.	
RemGrefs *	Remote global references (database accesses). Indicates the number of global references that are generated on behalf of ECP application servers.	
GRratio	Ratio of global references to remote global references.	
PhyRds	Physical reads from disk. A high number of physical reads may indicate a performance problem; you can improve the performance by increasing the number of database (global) buffers.	
Rdratio	Ratio of logical block reads to physical block reads, but zero if physical block reads is zero.	
Gloupds	Global updates (sets or kills).	
RemGupds *	Remote global updates.	
Rourefs	Routine references (includes **tag^routine**).	

Column	Description	Notes
RemRrefs *	Remote routine references.	
RouLaS	Routine loads and saves (fetch from or save to disk). A high number of routine loads/saves may indicate a performance problem; you can improve the performance by increasing the number of routine buffers.	
RemRLaS *	Remote routine loads and saves.	
PhyWrs	Physical writes to disk.	
WDQsz	Write Daemon Queue size (in blocks).	Not per second.
WDtmpq	Updated blocks in CACHETEMP.	Not per second. Not available in Caché 5.0.x.
WDphase	Phase of the Write Daemon. The most common phases are: • 0: Idle (WD is not running) • 5: WD is updating the Write Image Journal (WIJ) file. • 7: WD is committing WIJ and Journal. • 8: Databases are being updated.	Not per second.
Wijwri	Number of 256-KB blocks written to the WIJ. This is non-zero when the WD is writing data to the WIJ.	Not available in Caché 5.0.x.
RouCMs	Number of Routine Cache Misses.	Caché 2007.1 and higher.
Jrnwrts	Number of blocks written to journals.	
GblSz	Number of seizes on the global resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
pGblNsz	Percentage of NSeizes on the global resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
pGblAsz	Percentage of **ASeizes** on the global resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
RouSz	Number of seizes on the routine resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
pRouAsz	Percentage of **ASeizes** on the routine resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	

Column	Description	Notes
ObjSz	Number of seizes on the object resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
pObjAsz	Percentage of **ASeizes** on the object resource; see "Considering Seizes, ASeizes, and NSeizes" in this chapter.	
ActECP †	Number of active ECP connections.	Not per second.
Addblk †	Number of blocks added to ECP Client's cache.	
PrgBufL †	Number of blocks purged from ECP Client's cache due to global buffer shortage (on the ECP Client). A high number may indicate a performance problem on the ECP client; you can improve performance by increasing the number of global buffers on the ECP Client.	
PrgSrvR †	Number of blocks purged from ECP Client's cache by ECP server.	
BytSnt †	Number of bytes sent as an ECP Client.	
BytRcd †	Number of bytes received as an ECP Client.	
WDPass	WD cycle since startup.	Not per second. Not available in Caché 5.0.x.
IJUCnt	Number of jobs that WD is waiting for to continue this cycle.	Not per second. Not available in Caché 5.0.x.
IJULock	Indicates whether or not the IJULock flag is set . If IJULock is set, all updates are locked out while the WD finalizes the write cycle.	Not per second. Not available in Caché 5.0.x.

† ECP data displayed only for ECP configurations.

* 0 is displayed unless this is an ECP configuration.

11.1 Considering Seizes, ASeizes, and NSeizes

A **Seize** occurs whenever a job needs exclusive access on a given resource to guarantee that an update occurs without interference from other processes. If the Seize is not immediately satisfied, the update is postponed until it is satisfied.

On a single-CPU system, the process immediately hibernates (because it cannot do anything until the process holding the resource relinquishes it, which does not occur until after its own update completes).

On a multiple-CPU system, the process enters a holding loop in the "hope" that it will gain the resource in a reasonable time, thus avoiding the expense of hibernating. If the process gets access to the resource during the hold loop, the loop immediately exits and the process continues with its update; upon completing the update, the process relinquishes the

resource for other processes that may be waiting for it; this is an **Aseize**. If, at the end of the hold loop, the resource is still held by another process, the process continues to hibernate and wait to be woken up when the resource is released; this is an **Nseize**.

Nseizes are a natural consequence of running multiple processes on a single-CPU system; Aseizes are a natural consequence of running multiple processes on a multi-CPU system. The difference is that Nseizes incur system, or privileged, CPU time because the operating system must change the context of the running process, whereas an Aseize incurs user time on the CPU because it continues to run until the resource is gained and released, or until it gives up and hibernates. In general, on multi-CPU systems it is more expensive for the operating system to do the context switch than to loop a few times to avoid this operation because there is both CPU overhead and memory latency associated with context switching on multi-CPU systems.

12

Caché History Monitor

The Caché History Monitor maintains a historical database of performance and system usage metrics. Its primary purposes are to:

- Provide a performance baseline and help with analysis of performance issues.
- Help analyze system usage over time for capacity planning.

This database is defined in the SYS.History class package and kept in the %SYS namespace. All of the details of the database structure are published there and the data is accessible through SQL or the normal persistent object access. The class documentation in SYS.History also contains descriptions of all the individual properties, methods and queries that are available.

The data is generally organized into performance (see SYS.History.Performance) and system usage (see SYS.History.SystemUsage) data. The performance metrics are intended to be sampled at short intervals (by default, 30 seconds), and the system usage data at longer intervals (by default, 5 minutes). At the beginning of each day, the individual interval samples are summarized into hourly and daily tables as averages, maximums, minimums, standard deviation, medians and totals. You can select which, if any, of the summary functions are kept for each metric class. The interval and hourly data may be purged automatically after a defined number of days (by default, seven (7) days and 60 days, respectively); the daily summary is intended for long-term analysis and can be purged manually.

This chapter contains the following sections:

12.1 Base Metrics

All of the collected metrics are defined in four %SerialObject classes in SYS.History. These same classes are used as the basis for the Interval, Hourly, and Daily databases, so all of the properties are defined as %Numeric types to allow for decimal values in the summaries.

The performance related metrics are defined in:

- SYS.History.Performance — The properties in this class are general performance metrics like global references and routine calls.

Note: These properties are all "counter" types and the interval data is collected as deltas, which represent the change in the counter over the last interval. When this data is summarized into hourly and daily values, the data is normalized to per-second rates

- SYS.History.WriteDaemon — The properties in this class describe the performance of write daemon cycles. The system automatically keeps track of the last 20 write daemon cycles, and the History Monitor stores the data for the cycles that occurred in each interval. Typically, there are multiple cycles within each interval.

The system usage metrics are defined in:

- SYS.History.SystemUsage — The properties in this class track how busy the system is but do not tend to change as quickly or dramatically as the performance data; things like the number of processes in Caché and license information.
- SYS.History.Database — This class tracks the database growth, file size and free space, for each local database.

12.2 Collecting Data

To begin collecting data, you must do the following:

- Use the Caché System Monitor **^%SYSMONMGR** utility in the %SYS namespace to activate the %Monitor.System.HistoryPerf and/or %Monitor.System.HistorySys classes in Caché Application Monitor (which is part of System Monitor). These classes are registered in the %SYS namespace by default.
- Restart System Monitor in the %SYS namespace.

See Using ^%SYSMONMGR to Manage Application Monitor and Start/Stop System Monitor in the "Using Caché System Monitor" chapter of this guide for information about these procedures.

The detailed interval collection of data is defined in two persistent classes:

- SYS.History.PerfData — Includes the performance and write daemon classes as embedded objects.
- SYS.History.SysData — Includes the system usage and database classes.

The corresponding %Monitor classes must be activated in Application Monitor in order to collect data and build the history data:

- %Monitor.System.HistoryPerf — Collects instances of SYS.History.PerfData samples.
- %Monitor.System.HistorySys — Collects SYS.History.SysData samples.

System Monitor, including Application Monitor, starts by default in the %SYS namespace when the Caché instance starts. You can configured other startup namespaces, however. The %Monitor classes are provided by default only in %SYS, but can be added to other configured startup namespaces using **^%SYSMONMGR**.

12.3 Summaries

The %Monitor.System.HistoryPerf and %Monitor.System.HistorySys classes, as executed by Application Monitor, also create the hourly and daily summaries at the end of each day. The summaries are defined as the persistent classes SYS.History.Hourly and SYS.History.Daily; they include all four of the base classes as embedded objects.

For each metric property, the system may calculate the average, maximum (high-water mark), standard deviation, minimum, median, or total for each hour and for the whole day. The summary functions are selectable (or may be disabled) for each base class (SYS.History.Performance, SYS.History.WriteDaemon, SYS.History.SystemUsage, or SYS.History.Database) and for each summary period class, using the **SetSummary()** method of each of the base classes. By default, the History Monitor calculates average, maximum and standard deviation for each class for both hourly and daily summaries.

Note: The counter properties of the SYS.History.Performance class are normalized to per second rates for these calculations (except Total).

Purging Data

After creating the summaries, Application Monitor automatically purges the interval and hourly databases. The default is seven (7) days for interval data and 60 days for hourly data, but these may be changed using the **SetPurge()** method in SYS.History.PerfData and SYS.History.Hourly classes. The SYS.History.Daily data is not automatically purged, but can be done manually using the **SYS.History.Daily:Purge()** method.

12.4 Accessing the Data

Since the database is defined as persistent classes, the data is available using standard SQL or persistent object access. Using the SQL browser in the Management Portal is a quick and easy way to see the various SQL schemas/tables that are created, including the individual property values.

There are several basic queries implemented in each of the persistent classes in SYS.History (SYS.History.PerfData, SYS.History.SysData, SYS.History.Hourly, and SYS.History.Daily) that can be used to access the individual tables for a date range; for more information about the queries, see the class reference documentation.

The are also several **Export()** methods provided for each persistent class so that the individual tables can be exported to files in CSV format, suitable for use with a spreadsheet such as Microsoft Excel. In particular, the **SYS.History.PerfData:Export()** method creates a file that is very similar in format to that created by the **^mgstat** utility (for more information, see the "Monitoring Performance Using ^mgstat" chapter of this guide).

12.5 Adding User-defined Metrics

You can add user-defined metrics to the History Monitor (SYS.History package):

1. Create a class, or multiple classes, that inherit from SYS.History.Adaptor and add %Numeric properties to define the metrics.

   Note: User-written classes must be in the %SYS namespace, and should begin with "Z" or "z" to prevent naming conflicts with system classes and problems during upgrades.

2. Code the **Sample()** method to instantiate the class and provide periodic values for each property. This method is called when the interval data is collected.

3. When you compile your class, it is added as an embedded object to an interval persistent class in SYS.History. You can choose where and when it is collected using the *INTERVAL* parameter provided in SYS.History.Adaptor class. This selects which interval class it is added to and which %Monitor class does the collection, as shown in the following table:

INTERVAL Selected	Interval Class Used	%Monitor Class Used
"User" (default)	SYS.History.User	%Monitor.System.HistoryUser
"UserPerf"	SYS.History.UserPerf	%Monitor.System.HistoryPerf
"UserSys"	SYS.History.UserSys	%Monitor.System.HistorySys

Selecting "UserPerf" or "UserSys" lets you collect data at the same interval and with the same timestamp as SYS.History.PerfData or SYS.History.SysData, which makes it easier to correlate your data with the system data. "User" gives you a choice of a third (unrelated) time interval.

Note: There are several parameters in the SYS.History.Adaptor class that provide options for how properties are collected and summarized; for more information, see the SYS.History.Adaptor class reference documentation.

4. User-defined classes are also added as embedded objects to the SYS.History.UserHourly and SYS.History.UserDaily summary classes. The user-defined metrics are summarized and automatically purged just like the system metrics.

   Important: User-defined metric classes become embedded objects in persistent data. You should not change definitions after data collection has started: deleting objects can result in orphaned data; re-defining existing classes or properties can cause already stored data to be misinterpreted.

   However, because of the Cache Objects *schema evolution* feature (for information, see Schema Evolution in the "Defining Persistent Object Classes" chapter of *Using Caché Objects*), you can safely add new objects and properties.

A

Monitoring Caché Using BMC PATROL

Important: As of early 2015, InterSystems will stop enhancing the PATROL knowledge module files provided with Caché; in 2016, InterSystems will remove these PATROL knowledge files from supplied kits. However, BMC provides an SNMP interface for its PATROL product, allowing you to continue to use PATROL in conjunction with the Caché SNMP interface documented in the "Monitoring Caché Using SNMP" appendix of this guide. See http://blog.intersystems.com/compatibility/2015/01/ for more information.

This appendix describes the interface between Caché and BMC PATROL.

BMC PATROL is a tool for monitoring and managing various software systems. Caché supplies PATROL extensions so that you can monitor and collect information about Caché.

This interface allows users to monitor metrics of one or multiple Caché systems from the PATROL Console. The interface requires that the PATROL daemon is running on the Caché system to collect and output the metric values and that the Caché knowledge module files (*.km) are loaded into the PATROL Console to read and display these values.

This appendix provides information on the following topics:

- Running PATROL with Caché
- Caché PATROL Knowledge Modules
- Caché Metrics Used with BMC PATROL

A.1 Running PATROL with Caché

Run the **^PATROL** ObjectScript routine on each Caché installation that you want to monitor using the start and stop entry points, or by setting it to automatically run at system startup.

The routine starts a background process that outputs metrics to a file, patrol.dat, located in the Caché manager's directory *install-dir*\Mgr (where install-dir is the Caché installation directory). The file is rewritten for each collection period, so the file size is static. The file also includes an identifying header and a time stamp so that the PATROL Console can determine that it is active and up-to-date.

There are two ways to run and configure PATROL in Caché:

- Configure PATROL Settings
- Caché PATROL Routine

A.1.1 Configure PATROL Settings

Automatic PATROL Startup — You can set an option to automatically start the PATROL daemon at Caché startup using the Management Portal:

1. Navigate to the **System** > **Configuration** > **Monitor Settings** page of the Management Portal.
2. Click `Yes` in the **Start Patrol at System Startup** setting box.

   When `Yes`, the connection to PATROL starts automatically whenever Caché starts up. The default is `No`. When you edit this setting, the Caché end of the PATROL interface immediately stops and starts.
3. Click **Save**.

PATROL Arguments — From the same **System** > **Configuration** > **Monitor Settings** page of the Management Portal, you can also set the following PATROL settings:

- **Patrol Top Processes to Monitor** — Number of processes displayed in the Process Status window on the PATROL console. This window shows the *top* processes as sorted by global or routine activity. The default number of processes is 20. A value of 0 tells the PATROL utility to stop calculating the top processes, potentially saving significant work on systems with many processes. The valid range is 1–10000 processes.
- **Patrol Display Mode** — Controls how the monitoring data is displayed in the PATROL console. The default option is `Total`. Options are as follows:
  - `Total` displays the total counts since the collection was started.
  - `Delta` displays the count for the last collection period.
  - `Rate` displays a calculated count per second.
- **Patrol Collection Interval Seconds** — Number of seconds between each time Caché collects data and makes it available to PATROL. The default is 30 seconds; the valid range is 1–900 seconds.

By default, PATROL uses standard Caché system counters to generate metrics (for more information see Caché Metrics Used with BMC PATROL in this appendix). However, you can activate the extended counters, which use more system resources than the standard counters. To change the counters, use the following function before starting the **^PATROL** routine:

```
$$Light^PATROL(n)
```

where *n* is:

- `0` — PATROL uses extended counters
- `1` — PATROL uses standard counters (“light” version)

To see which version is currently specified, in the Terminal, enter the following command:

```
%SYS>write $$Light^PATROL()
```

A.1.2 Caché PATROL Routine

Caché provides entry points to the **^PATROL** routine to start and stop PATROL.

To start PATROL:

```
Do start^PATROL(display,process,timer)
```

The arguments are described in the following table:

Argument	Description	Default	Management Portal Monitor Setting
display	Display mode. The literals `total`, `delta`, or `rate` to indicate the type of numbers to output.	`total`	**Patrol Display Mode**
process	How many processes for which to pass %SS statistics.	20	**Patrol Top Processes to Monitor**
timer	Collection period in seconds.	30	**Patrol Collection Interval Seconds**

For example:

```
Do start^PATROL("total",20,30)
```

Sets the PATROL console to display the total statistic counts, since the collection started, for the top 20 processes; Caché sends the information every 30 seconds. The collection period argument is passed to the PATROL console so that the collection and display update are synchronized.

To stop PATROL:

```
Do stop^PATROL
```

A.2 Caché PATROL Knowledge Modules

The architecture of PATROL is based on the concept of knowledge modules. A *knowledge module* contains a set of commands, parameters to monitor, and actions used by PATROL. The Caché plug-in for PATROL consists of several knowledge modules to be loaded into the PATROL Console. These .km files are provided in the Caché Patrol directory, *install-dir*\Patrol.

Once these knowledge modules are loaded, the Console automatically attempts to discover Caché installations on all connected systems. The discovery process either searches the Registry on Windows platforms or parses the output from the **ccontrol list** command on UNIX®/Linux platforms. For each Caché installation it finds it checks to see if the patrol.dat file exists in the Caché manager's directory and if the time stamp within that file is current. Caché installations which are currently reporting Caché metrics for PATROL appear in the PATROL Console.

A.2.1 Adding Caché Modules to PATROL

Included in the Patrol directory is the ISC_CACHE.kml file listing all of the Caché knowledge modules. Use the following procedure to load and activate the knowledge modules:

1. From the PATROL Console **File** menu, click **Load KM**.
2. Select the ISC_CACHE.kml file in the Caché Patrol directory, *install-dir*\Patrol.
3. The ISC_CACHE module should appear in the **Desktop** tab of the **Console** in a few seconds.
4. Right-click ISC_CACHE and choose **Add Configuration** from the **KM Commands** menu.
5. In the **Add Configuration** dialog box, enter an instance name and the Caché installation directory.
6. You may need to wait for, at most, 30 seconds (PATROL default sync period), before PATROL recognizes Caché statistics.

For more information you can consult the BMC Web site.

If any Caché installations are discovered on a system, then the main entry for Caché (the Caché class) appears under that system entry. Each Caché instance (each Caché configuration installed on that system) appears under the Caché class. Under each Caché instance are the general metric categories of Overview, Global, Routines, Disk Activity, Network, and Other.

For example:

```
- PATROLMainMap
  - TEST1
    - ISC_CACHE
      - ISC_Config_CACHE
        + ISC_DiskActivity
        + ISC_Global
        + ISC_Network
        + ISC_Other
        + ISC_Overview
        + ISC_Routine
```

The categories expand to show all the individual metrics. The metrics under **Overview** are gauges showing current levels. The others are graphs showing values over time.

Right clicking the Caché configuration allows the user to select Caché-specific commands, to either **Remove Configuration** or show a **Process Status** window.

Manually adding a configuration should not normally be necessary, since all Caché installations should be automatically discovered, but might be useful if there is a question or a problem with a specific installation.

Error messages from the Caché KMs may be output to the **System Output** window. Check these messages if you have questions about Caché installations that are not automatically discovered.

A.3 Caché Metrics Used with BMC PATROL

The list of metrics for Caché:

Table I–1: Caché PATROL Metrics

Category	Metric
Overview	Global Refs (gauge)
	Global Sets, Reads, Kills (graph)
	Net Global Refs (gauge)
	Net Global Sets, Reads, Kills (graph)
	Routine Lines (gauge)
	Routine Loads (gauge)
	Locks (gauge)
	Process Count (graph)
	Cache Efficiency (graph) (= 100*(LogicalReads/(LogicalReads + Physical Reads)))
	Licenses Used (gauge)

Category	Metric
Global	Global Refs
	Global Sets [1]
	Global Kills [2]
	Global Reads
	Blocks Allocated [2]
	Locks
	Successful Locks
	Failed Locks
	Job InGlobal
	WD QueSize
	Global AvailBufs
	Que Gaccess
	Que GaccUpd
	Que GBFAny
	Que GBFSpec
	Journal Entries
	Jrn FileSize
	Jrn EndOffset
	Tot Global Bufs
	GThrottle Cur
	GThrottle Max
	GThrottle Cnt
Routine	Routine Lines [3]
	Routine Loads
	Routine Fetches

Category	Metric
Disk Activity	Physical Directory Reads
	Physical U-Ptr Reads
	Physical B-Ptr Reads
	Physical Data Reads
	Physical Routine Reads
	Physical Map Reads
	Physical Other Reads
	Physical Directory Writes
	Physical U-Ptr Writes
	Physical B-Ptr Writes
	Physical Data Writes
	Physical Routine Writes
	Physical Map Writes
	Physical Other Writes
	Logical Directory Reads
	Logical U-Ptr Reads
	Logical B-Ptr Reads
	Logical Data Reads
	Logical Routine Reads
	Logical Map Reads
	Logical Other Reads
Network	Net Global Refs
	Net Global Sets
	Net Global Kills [2]
	Net Global Reads
	Net Requests Sent [2]
	Net Cache Hits
	Net Cache Misses
	Net Locks [2]
	Net Retransmits [2]
	Net Buffer
	Net GblJobs

Category	Metric
Other	Terminal Reads [2]
	Terminal Writes [2]
	Terminal Read Char [2]
	Terminal Write Char [2]
	Sequential Read [2]
	Sequential Write [2]

When using the default Caché standard counters (“light” version), the following notes are applicable:

[1] Global Sets is a combination of Global Sets and Global Kills.

[2] Counter is not available; therefore, this value is 0.

[3] Routine Lines reflects the total commands executed.

B

Monitoring Caché Using SNMP

This document describes the interface between Caché and SNMP (Simple Network Management Protocol). SNMP is a communication protocol that has gained widespread acceptance as a method of managing TCP/IP networks, including individual network devices, and computer devices in general. Its popularity has expanded its use as the underlying structure and protocol for many enterprise management tools. This is its main importance to Caché: a standard way to provide management and monitoring information to a wide variety of management tools.

SNMP is both a standard message format and a standard set of definitions for managed objects. It also provides a standard structure for adding custom-managed objects, a feature that Caché uses to define its management information for use by other applications.

The interface description includes the following topics:

- Using SNMP with Caché
- Caché as a Subagent
- Managing SNMP in Caché
- SNMP Troubleshooting
- Caché MIB Structure
- Sample User-defined SNMP Monitor Class

Note: For a detailed practical look at configuring SNMP to work with Caché by an InterSystems senior technology architect, see InterSystems Data Platforms and performance - Part 5 Monitoring with SNMP on InterSystems Developer Community.

B.1 Using SNMP with Caché

SNMP defines a client/server relationship where the client (a network management application) connects to a server program (called the SNMP agent) which executes on a remote network device or a computer system. The client requests and receives information from that agent. There are four basic types of SNMP messages:

- **GET** – fetch the data for a specific managed object
- **GETNEXT** – get data for the *next* managed object in a hierarchical tree, allowing system managers to walk through all the data for a device
- **SET** – set the value for a specific managed object

- **TRAP** – an asynchronous alert sent by the managed device or system

The SNMP MIB (Management Information Base) contains definitions of the managed objects. Each device publishes a file, also referred to as its MIB, which defines which portion of the standard MIB it supports, along with any custom definitions of managed objects. For Caché, this is the ISC-CACHE.mib file, located in the *install-dir*\SNMP directory. In addition, if you are using Ensemble, refer to the ISC-ENSEMBLE.mib file in the same directory.

B.2 Caché as a Subagent

The SNMP client connects to the SNMP agent which is listening on a well-known address, port `161`. Since the client expects to connect on this particular port, there can only be one SNMP agent on a computer system. To allow access to multiple applications on the system, developers can implement *master agents*, which may be extended or connected to multiple subagents. InterSystems has implemented the Caché SNMP interface as a subagent, designed to communicate through an SNMP master agent.

Most operating systems that Caché supports provide an SNMP master agent which is extensible in some way to support multiple subagents. Many of these agents, however, implement their extensibility in a proprietary and incompatible manner. Caché implements its subagent using the Agent Extensibility (AgentX) protocol, an IETF-proposed standard as described in RFC 2741.

Some of the standard SNMP master agents support AgentX. If the SNMP master agent supplied by an operating system is not AgentX-compatible, you can replace it with the public domain Net-SNMP agent.

Note: The exception is the Windows standard agent which does not support AgentX and for which the Net-SNMP version may not be adequate. For this exception, InterSystems supplies a Windows extension agent DLL, iscsnmp.dll, which handles the connection between the standard Windows SNMP service extension API and the Caché AgentX server.

B.3 Managing SNMP in Caché

Since SNMP is a standard protocol, the management of the Caché subagent is minimal. The most important task is to verify that the SNMP master agent on the system is compatible with the Agent Extensibility (AgentX) protocol (see Caché as a Subagent) and it is active and listening for connections on the standard AgentX TCP port `705`. On Windows systems, the system automatically installs a DLL to connect with the standard Windows SNMP service. Verify that the Windows SNMP service is installed and started either automatically or manually.

Important: Some SNMP master agents, notably Net-SNMP on Linux, do not enable AgentX by default and do not use TCP port `705` by default once they are enabled. For Net-SNMP you must modify the snmpd.conf file to enable communications with the Caché subagent. Recent versions of Net-SNMP also implement VACM (View-based Access Control Model) security and, by default, only allow access to the mib-2.system subtree; the Caché subagent starts and runs without error, but no SNMP requests are forwarded to Caché. You must expand the "views" defined in snmpd.conf to include the Caché MIB subtree.

Next, enable the Caché monitoring service using the following steps:

1. Navigate to **System** > **Security Management** > **Services** in the Management Portal.
2. Click the **`%Service_Monitor`** service.
3. Select the **Service enabled** check box and click **Save**.

4. Return to the list of services page and ensure that the **%Service_Monitor** service is enabled.

Finally, configure the Caché SNMP subagent to start automatically at Caché startup using the following steps:

1. Navigate to **System** > **Configuration** > **Monitor Settings** in the Management Portal.
2. Select Yes for the **Start SNMP Agent at System Startup** setting and click **Save**.
3. When you edit this setting, the Caché end of the SNMP interface immediately stops and starts.

You can also start and stop the Caché SNMP subagent manually or programmatically using the **^SNMP** routine:

```
Do start^SNMP(<port>,<timeout>)
Do stop^SNMP
```

where <*port*> is the TCP port for the connection (default is 705) and <*timeout*> is the TCP port read timeout value (default is 20 seconds). Until the <*timeout*> value is reached, Caché logs any problems encountered while establishing a connection or answering requests in the SNMP.LOG file in the *install-dir*\Mgr directory.

Note: When the SNMP master agent is restarted, it may be necessary to manually restart the Caché SNMP subagent using the **^SNMP** routine, as described in the foregoing.

B.4 SNMP Troubleshooting

The Caché subagent (running the **^SNMP** routine) depends on the correct installation and configuration of the SNMP master agent supplied by the operating system. As noted in Caché as a Subagent, there are two main ways in which the **^SNMP** routine communicates with this master agent:

- Primarily, **^SNMP** uses the AgentX protocol on TCP port 705
- On Windows, **^SNMP** uses a Windows extension agent DLL installed as iscsnmp.dll.

Detailed instructions for configuring the SNMP agent should be supplied with the operating system, and system managers should take some time to understand how to do this. The following are some basic guidelines and tips for troubleshooting if problems are encountered in getting Caché to communicate with the SNMP agent.

General

On all systems:

- Make sure the SNMP agent is working independently of Caché and you can at least query the mib-2.system tree for general system information. If this fails, on Windows check the Windows SNMP Service; on UNIX®/Linux see if the SNMP daemon (snmpd) is running.
- If you can successfully query the SNMP system information but not the Caché MIB, then check for a background process in Caché running the **^SNMP** routine. Try starting it using the **$$start^SNMP()** function. If the routine starts but does not continue running, check for errors in the cconsole.log and SNMP.log log files in the Caché *install-dir*/mgr directory. On Windows, iscsnmp.dll logs any errors it encounters in %System%\System32\snmpdbg.log (on a 64–bit Windows system, the file is in the SysWOW64 subdirectory).
- Make sure the Caché %Service_Monitor service is enabled.
- More information can be logged to the SNMP.log file if you set **^SYS("MONITOR","SNMP","DEBUG")=1** in the %SYS namespace and restart the **^SNMP** Caché subagent process. This logs details about each message received and sent.

Windows

On Windows systems:

- Not all Windows versions install the Windows SNMP service by default. You may need to do this as an additional step. Make sure the **Security** tab of the **Properties** dialog for the service has at least a **public** community with **READ** rights. To send SNMP traps, you must define a **Community Name** and **Destination** on the **Trap** tab of the properties dialog.
- Caché expects the SNMP service to be installed *before* you install Caché, so it can add iscsnmp.dll to the proper Registry keys. Once Caché is installed, the SNMP service must be restarted so that it properly loads iscsnmp.dll and will find and communicate with the new Caché instance.

  Note: If Caché is installed before the SNMP service, iscsnmp.dll cannot be properly registered, and you must use the **set myStatus=$$Register^SNMP()** function to do this after the Windows SNMP service is installed. Once this is done the SNMP service must be restarted.

- On Windows, the **$$start^SNMP()** function only signals the SNMP service, and the Caché **^SNMP** process is actually started by a callback from the SNMP service into Cache. It may take a few seconds for the process to start, and a few more seconds before it can respond to queries.

UNIX®

Many UNIX operating systems (such as IBM AIX®) do *not* support the AgentX protocol at this time. If your system does not support AgentX, you must install a separate SNMP agent which supports AgentX, such as Net-SNMP.

Linux/Mac OS X and Net-SNMP

On Linux and Mac OS X systems:

- AgentX support is *not* enabled by default, and the default port is not 705. You must modify the snmpd.conf file and add **master agentx** and **agentXSocket TCP:localhost:705**, or use **snmpd -x TCP:localhost:705** on the command line.
- Basic system information like `syslocation`, `syscontact` and `sysservices` must be defined in snmpd.conf to enable successful startup of the snmpd daemon.
- Recent versions of Net-SNMP also implement VACM (View-based Access Control Model) security and, by default, allow access to the `mib-2.system` subtree only; as a result, the Caché subagent starts and runs without error, but no SNMP requests are forwarded to Caché. You must expand the "views" defined in snmpd.conf to include the Caché MIB subtree.
- To send SNMP traps, you must define a destination using the **trapsink** parameter in snmpd.conf, for example **trapsink 192.16.61.36 public**.

B.5 Caché MIB Structure

All of the managed object data available through the Caché SNMP interface is defined in the Caché MIB file, ISC-CACHE.mib, which is located in the *install-dir*\SNMP directory. Typically an SNMP management application must load the MIB file for the managed application to understand and appropriately display the information. Since this procedure varies among applications, consult your management application documentation for the appropriate way to load the Caché MIB.

The specific data defined in the Caché MIB is documented in the file itself and, therefore, is not repeated here. However, it may be valuable to understand the overall structure of the Caché MIB tree, especially as it relates to multiple instances on the same system.

Note: The best way to view the MIB tree is to load the MIB into a management application or MIB browser. These tools display the MIB as a tree with object IDs (OIDs), matching text representations of the objects, and descriptions of the objects.

SNMP defines a specific, hierarchical tree structure for all managed objects called the Structure of Management Information (SMI), which is detailed in RFC 1155. Each managed object is named by a unique object identifier (OID), which is written as a sequence of integers separated by periods, for example: `1.3.6.1.2.1.1.1`. The MIB translates this dotted integer identifier into a text name.

The standard SNMP MIB defines many standard managed objects. To define application-specific extensions to the standard MIB, as Caché does, an application uses the *enterprise* branch which is defined as:

```
iso.org.dod.internet.private.enterprises (1.3.6.1.4.1)
```

The Internet Assigned Numbers Authority (IANA) assigns each organization a private enterprise number as the next level in the hierarchy. For Caché this is `16563` which represents *intersystems*.

Below this, Caché implements its enterprise private subtree as follows:

- The level below *intersystems* is the "product" or application ID level. For Caché this is `.1` (`iscCache`). This serves as the MIB module identity.
- The next level is the "object" level, which separates data objects from notifications. For Caché, these are `.1` (`cacheObjects`) and `.2` (`cacheTraps`). By convention, the *intersystems* tree uses a brief lowercase prefix added to all data objects and notification names. For Caché this is *cache*.
- The next level is the "table" or group level. All data objects are organized into tables, even if there is only one instance or "row" to the table. This serves to organize the management data objects into groups. This is also necessary to support multiple Caché instances on one machine. All tables use the Caché instance name as the first index of the table. The tables may also have one or more additional indices.
- The next level is the "conceptual row" for the table (as required by the SNMP SMI). This is always `.1`.
- Finally, the individual data objects contained in that table, including any that are designated as indices.
- The notifications (traps) are defined as individual entries at the same hierarchical level as the "table". For more information, see Caché SNMP Traps in this appendix.
- Caché-specific auxiliary objects sent via notifications (traps) are defined as individual entries at the same hierarchical level as the "table". For more information, see Caché SNMP Traps in this appendix.

For example, encode the size of a database as:

```
1.3.6.1.4.1.16563.1.1.3.1.6.4.84.69.83.84.1
```

This translates to:

```
iso.org.dod.internet.private.enterprises.intersystems.iscCache.cacheObjects
.cacheDBTab.cacheDBRow.cacheDBSize.TEST(instname).1(DBindex)
```

B.5.1 Ensemble MIB Structure

In addition to the Caché managed object data defined in ISC-CACHE.mib, Ensemble also has the information defined in ISC-ENSEMBLE.mib available. Ensemble uses the same IANA private enterprise number as Caché, `16563` which represents *intersystems*.

Below this, Ensemble implements its enterprise private subtree as follows:

- The level below *intersystems* is the "product" or application ID level. For Ensemble this is `.2` (`iscEnsemble`). This serves as the MIB module identity.

- The next level is the "object" level, which separates data objects from notifications. For Ensemble, these are: `.1` (`ensObjects`) and `.2` (`ensTraps`). By convention, the *intersystems* tree uses a brief lowercase prefix added to all data objects and notification names. For Ensemble this is *ens*.
- Subsequent levels follow the same implementation as Caché described in the Caché MIB Structure section.

B.5.2 Extending the Caché MIB

Application programmers can add managed object definitions and extend the MIB for which the Caché subagent provides data. This is not intended to be a complete MIB editor or SNMP toolkit; rather, it is a way to add simple application metrics that you can browse or query through SNMP.

Note: The objects must follow the basic Caché SNMP structure, there is limited support for SNMP table structures (only integer-valued indexes are supported), and SNMP traps are not created (see the %Monitor.Alert class). A basic understanding of SNMP structure of management information is helpful.

To create these objects do the following:

1. Create Caché object definitions in classes that inherit from the %Monitor.Adaptor class. See the *InterSystems Class Reference* for details about adding managed objects to the %Monitor package.
2. Execute an SNMP class method to enable these managed objects in SNMP and create a MIB definition file for management applications to use. The method to accomplish this is **MonitorTools.SNMP.CreateMIB()**.

See the MonitorTools.SNMP class documentation in the *InterSystems Class Reference* for details of the **CreateMIB()** method parameters.

The method creates a branch of the private enterprise MIB tree for a specific application defined in the %Monitor database. In addition to creating the actual MIB file for the application, the method also creates an internal outline of the MIB tree. The Caché subagent uses this to register the MIB subtree, walk the tree for **GETNEXT** requests, and reference specific objects methods for gathering the instance data in **GET** requests.

All the managed object definitions use the same general organization as the Caché enterprise MIB tree, that is: `application.objects.table.row.item.indices`. The first index for all tables is the Caché application ID. All applications must register with the IANA to obtain their own private enterprise number, which is one of the parameters in the **CreateMIB()** method.

To disable the application in SNMP, use the **MonitorTools.SNMP.DeleteMIB()** method. This deletes the internal outline of the application MIB, so the Caché subagent no longer registers or answers requests for that private enterprise MIB subtree.

For an example of defining a user Monitor class, see Sample User-defined SNMP Monitor Class in this chapter.

B.5.3 Caché SNMP Traps

In addition to the object data and metrics available through SNMP queries, Caché can send asynchronous alerts or SNMP traps. The following table describes the Caché-specific SNMP traps.

Table II–1: Caché SNMP Notification Objects (Traps)

Trap Name (Number)	Description
cacheStart (1)	The Caché instance has been started.
cacheStop (2)	The Caché instance is in the process of shutting down.
cacheDBExpand (3)	A Caché database has expanded successfully.

Trap Name (Number)	Description
cacheDBOutOfSpace (4)	Future expansion of a Caché database may be limited; there is not enough free space on the file system for 10 more expansions or there is less than 50 MB of free space.
cacheDBStatusChange (5)	The read/write status of a Caché database has been changed.
cacheDBWriteFail (6)	A write to a Caché database has failed. It includes the Caché error code for the failed write.
cacheWDStop (7)	The Write Daemon for a Caché instance has stalled.
cacheWDPanic (8)	The Write Daemon for a Caché instance has entered "panic" mode; that is, the Write Daemon is out of buffers and must write database blocks directly to disk without first committing them to the Write Image Journal (WIJ) file.
cacheLockTableFull (9)	The lock table for a Caché instance is full, which causes subsequent Locks to fail.
cacheProcessFail (10)	A process has exited Caché abnormally (due to an access violation). For detailed information, see the cconsole.log file.
cacheECPTroubleDSrv (11)	A connection to this ECP Data Server for a Caché database has encountered a serious communication problem. For detailed information, see the cconsole.log file.
cacheECPTroubleASrv (12)	A connection from this ECP Application Server to a remote Caché database has encountered a serious communication problem. For detailed information, see the cconsole.log file.
cacheAuditLost (13)	Caché has failed to record an Audit event. The most likely cause is a problem with space for the Audit database, which requires operator assistance.
cacheDaemonFail (14)	A major Caché system process (daemon) has died because it encountered an unhandled error.
cacheLoggedError (15)	A "severe " error has been logged in the cconsole.log file. This trap includes the error message defined in cacheSysErrorMsg.
cacheLicenseExceed (16)	A request for a license has exceeded the number of licenses currently available or allowed.
cacheAppAlert (100)	This is a generic trap that can be used by Caché applications to generate alerts via SNMP. For detailed information about how this trap can be used, see the **%Monitor.Alert.External** class method.

The following table describes the Caché-specific auxiliary objects that can be sent in the traps described in the preceding table.

Table II–2: Caché-specific Auxiliary Objects Sent in Traps

Auxiliary Object Name (Number)	Description
cacheDBWriteError (1)	The Caché-specific error code for a failed database write. Possible values are: <DATABASE>, <DISKHARD>, <BLOCKNUMBER>, <FILEFULL> or <DATABASE MAP LABEL>.
cacheApp (2)	A short text string (maximum of 20 characters) that identifies the application that generated (or was the source of) a cacheAppAlert trap.

Auxiliary Object Name (Number)	Description
cacheAppSeverity (3)	A code that indicates the severity of the problem for a cacheAppAlert trap. The code can be **0** (info), **1** (warning), **2** (severe), or **3** (fatal)
cacheApptext (4)	A text string description (maximum of 1024 characters) of the problem, error, or event that caused the cacheAppAlert trap.

The following table describes the Ensemble-specific SNMP traps.

Table II–3: Ensemble SNMP Notification Objects (Traps)

Trap Name (Number)	Description
ensEvent (1)	The Caché implementation of SNMP signals an Ensemble_LogEvent each time an Ensemble business host posts an alert to the Ensemble Event Log.

B.6 Sample User-defined SNMP Monitor Class

This section describes an example of how to define a user Application Monitor class (see Caché Application Monitor in the "Using Caché System Monitor" chapter of this guide) that you can query via SNMP. The Application Monitor includes only properties with %Monitor data types in the SNMP data.

Example Sample Class

The following is the sample class for this example:

Class Definition

```
Class SNMP.Example Extends %Monitor.Adaptor
{

  /// Give the application a name. This allows you to group different
  /// classes together under the same application level in the SNMP MIB.
  /// The default is the same as the Package name.
  Parameter APPLICATION = "MyApp";

  /// This groups a set of properties together at the "table" level of the
  /// SNMP MIB hierarchy. The default is the Class name.
  Parameter GROUPNAME = "MyTable";

  /// An integer metric counter
  Property Counter1 As %Monitor.Integer;

  /// Another integer metric counter
  Property Counter2 As %Monitor.Integer;

  /// A status indicator as a string data type
  Property Status As %Monitor.String;

  /// The method is REQUIRED. It is where the Application Monitor
  /// calls to collect data samples, which then get picked up by the
  /// ^SNMP server process when requested.
  Method GetSample() As %Status
  {
    set ..Counter1=$r(200)
    set ..Counter2=200+$r(100)
    set n=$r(4)
    set ..Status=$s(n=1:"Crashed",n=2:"Warning",n=3:"Error",1:"Normal")
    Quit $$$OK
  }

}
```

Before compiling this class in a user namespace, Caché must load supporting classes into the namespace; these classes are required to store the data samples for SNMP.

To load the classes, run **^%SYSMONMGR**, as described in Using ^%SYSMONMGR to Manage Application Monitor in the "Using Caché System Monitor" chapter of this guide, and do the following:

- Select option 2, `Manage Monitor Classes`
- Select option 3, `Register Monitor System Classes`.

When you compile the sample class, it creates the SNMP.Sample.Example class to store the sample data.

Important: Do not delete generated sample classes explicitly; if you select both the Application Monitor and generated sample classes for deletion, the sample class routines remain although the monitor class routines are deleted, which causes an error. To ensure that all sample class routines are properly removed, delete *only* the Application Monitor class that generated it; when you delete the monitor class both the monitor class and generated sample class, as well as related routines for both classes, are deleted. For example, to delete a sample class (for example, SNMP.Sample.Example), use the Management Portal to delete the monitor class from which it is generated (that is, SNMP.Example).

Run **^%SYSMONMGR** to activate the sample class and start the Application Monitor to collect samples:

1. Select option 2, `Manage Monitor Classes`.
2. Select option 1, `Activate/Deactivate a Monitor Class`.
3. To see an numbered list of registered Monitor Classes, enter `?`.
4. Enter the number of Monitor Class you want to activate; for example, to activate a user-defined class named SNMP.Example, enter the number next to the class name.
5. Select option 6, `Exit` (to return to the Application Monitor main menu).
6. Select option 1, `Manage Application Monitor`.
7. Select option 1, `Start Application Monitor`.
8. Select option 5, `Exit` (to return to the Application Monitor main menu).
9. Select option 6, `Exit` (to exit from Application Monitor main menu).

Note: For information about configuring and using Caché Application Monitor, see Caché Application Monitor in the "Using the Caché System Monitor" chapter of this guide.

Example of Creating a User MIB

To create the SNMP MIB, run the **MonitorTools.SNMP:CreateMIB** method from the %SYS namespace. See the MonitorTools.SNMP class documentation for details.

The input parameters for the method are similar to the following:

```
CreateMIB("MyApp","USER",99990,1,"mycorp","myapp","mc","MC-MYAPP","Unknown",1)
```

Important: Do not use `99990` as the Enterprise ID for production; each organization should register with the IANA for their own ID.

```
USER>set $namespace = %SYS"

%SYS>Do ##class(MonitorTools.SNMP).CreateMIB("MyApp","USER",99990,1,"mycorp",
"myapp","mc","MC-MYAPP","Unknown",1)


Create SNMP structure for Application - MyApp

     Group - MyTable
          Counter1 = Integer
          Counter2 = Integer
```

```
          Status = String


Create MIB file for MyApp

     Generate table MyTable
          Add object Counter1
          Add object Counter2
          Add object Status


%SYS>
```

This creates the MC-MYAPP.MIB file in your default directory (*install-dir*\Mgr\User), which you can load into your SNMP management application.

Note: You may need to restart the SNMP master agent and the Caché **^SNMP** service on your system before each recognizes this MIB.

```
--
-- MIB file generated for mcMyApp product.
--
-- Sep 16, 2008
--

MC-MYAPP DEFINITIONS ::= BEGIN

IMPORTS

  MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
  Counter32, Gauge32, Integer32
  FROM SNMPv2-SMI
  DisplayString
  FROM SNMPv2-TC
  enterprises
  FROM RFC1155-SMI
  cacheSysIndex
  FROM ISC-CACHE;


mcMyApp MODULE-IDENTITY
    LAST-UPDATED "200809161700Z"
    ORGANIZATION "mycorp"
    CONTACT-INFO "
                  Unknown"
    DESCRIPTION ""
    ::= { mycorp 1 }

mycorp OBJECT IDENTIFIER ::= { enterprises 16563 }

myappObjects OBJECT IDENTIFIER ::= { mcMyApp 1 }

--
-- Begin tables
--

-- Table myappMyTable

myappMyTable         OBJECT-TYPE
       SYNTAX        SEQUENCE OF myappMyTableR
       MAX-ACCESS    not-accessible
       STATUS        current
       DESCRIPTION
       ""
   ::= { myappObjects 1 }

myappMyTableR        OBJECT-TYPE
       SYNTAX        myappMyTableR
       MAX-ACCESS    not-accessible
       STATUS        current
       DESCRIPTION
       "Conceptual row for MyTable table."
       INDEX  { cacheSysIndex }
   ::= { myappMyTable 1 }

myappMyTableR ::=
       SEQUENCE {
        myappCounter1   Integer32
        myappCounter2   Integer32
        myappStatus   DisplayString
       }
```

```
myappCounter1        OBJECT-TYPE
       SYNTAX        Integer32
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       ""
       ::= { myappMyTableR 1 }

myappCounter2        OBJECT-TYPE
       SYNTAX        Integer32
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       ""
       ::= { myappMyTableR 2 }

myappStatus          OBJECT-TYPE
       SYNTAX        DisplayString
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       "Status"
       ::= { myappMyTableR 3 }

--  End of MyTable table

myappTraps  OBJECT IDENTIFIER ::= { mcMyApp 2 }

-------------------------------------------------------------
 END
```

C

Monitoring Caché Using WMI

Important: WMI support is deprecated in release 2015.1 of Caché. InterSystems recommends the use of SNMP and the robust Caché SNMP interface (see the "Monitoring Caché Using SNMP" appendix of this guide) for monitoring Caché. See http://blog.intersystems.com/compatibility/2014/5 for more information.

Windows Management Instrumentation (WMI) is a feature of the Windows operating system that provides a standardized way of collecting management information. It allows users and programmers to access management information from the operating system and other applications in a variety of ways, including scripts, programming languages, and management tools and applications. WMI is the Microsoft implementation of the Web-based Enterprise Management (WBEM) standard from the Distributed Management Task Force (DMTF).

Caché implements several WMI classes of management information. It includes both an Instance provider to allow queries of performance and management data, and an Event provider which signals significant events or errors that may occur in Caché. See the Windows Management Instrumentation (WMI) section of the MSDN Library for a description of how to query WMI classes and how to receive WMI events.

The interface description includes the following topics:

- Configuring WMI in Caché
- Using WMI with Caché
- Generating a WMI Documentation Text File

C.1 Configuring WMI in Caché

To prepare the Caché environment for collecting WMI information, complete the following two tasks:

- Enable Monitoring and WMI
- Compile WMI Classes

C.1.1 Enable Monitoring and WMI

To use WMI to collect Caché information, you must enable WMI collection as well as enabling the Caché monitoring service. From the Management Portal perform the following steps:

1. Navigate to **System** > **Configuration** > **Monitor Settings** in the Management Portal.
2. If the first line in the setting box indicates that the Monitor service is disabled, perform the following steps:

a. Click **Edit** to display the **Edit Service** page for the **%Service_Monitor** service.

b. Select the **Service enabled** check box and click **Save**, which returns you to the **Monitor Settings** page.

3. Select Yes from the **WMI Enabled** list and click **Save**.

C.1.2 Compile WMI Classes

The IscProc.mof file defines the InterSystems WMI classes. The file is installed in the WMI subdirectory of the manager's directory *install-dir*\Mgr (where install-dir is the installation directory your instance). The following example uses C:\MyCache as the installation directory.

This text file describes the InterSystems management classes in Managed Object Format (MOF). You must compile these classes into the WMI repository on your system before you can use them. From a Windows command prompt, use the mofcomp.exe Microsoft tool to compile the classes as shown in the following example:

```
C:\>cd c:\MyCache\mgr\WMI

C:\MyCache\Mgr\WMI>mofcomp IscProv.mof
Microsoft (R) 32-bit MOF Compiler Version 5.1.2600.2180
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: IscProv.mof
MOF file has been successfully parsed
Storing data in the repository...
Done!

C:\MyCache\Mgr\WMI>
```

C.2 Using WMI with Caché

Caché stores its WMI classes in the root\cache namespace in the WMI repository. Once you compile these WMI classes and they are in the WMI repository, you can access the management information using scripts, programming languages, or management tools.

Using a basic tool such as WMI CIM Studio (available as part of the WMI Administrative Tools from Microsoft) you can browse through all the classes in the root\cache namespace, list all properties for each class, and view the help text for each class which includes descriptions of each property. You can also query the Caché system for all instances of each class, which displays the live data from a running instance.

The Cache_DatabaseSystem class enumerates all instances of Caché installed on the system (from the Windows Registry), and shows which ones are running and connected (EnabledState=2). Other classes only show data for WMI-enabled instances of Caché.

Caché also signals WMI events, which you can receive using scripts, programs, or management tools. The WMI Administrative Tools include a basic WMI Event Viewer. The Cache_Event class is defined as a subclass of the __ExtrinsicEvent class; you can view its properties by navigating the __SystemClass hierarchy using WMI CIM Studio. The WMI Events are the same events defined as SNMP Notifications. You can find the most current list by inspecting the ISC-CACHE.MIB file in the SNMP directory of your Caché installation. See the "Monitoring Caché Using SNMP" chapter of this guide for details.

The current events signaled by Caché include:

Event	Description
cacheStart	Caché instance startup
cacheStop	Caché instance shutdown

Event	Description
cacheDBExpand	Database expansion successful
cacheDBOutOfSpace	Database expansion close to limit (not enough free space for 10 more expansions or less than 50 MB free space)
cacheDBStatusChange	Database read/write status change
cacheDBWriteFail	Database write failure
cacheWDStop	Write daemon stalling
cacheWDPanic	Write daemon entering *panic* mode
cacheLockTableFull	Lock table full (out of memory)
cacheProcessFail	Caché process access violation
cacheECPTroubleDSrv	ECP data server connection in *trouble* mode
cacheECPTroubleASrv	ECP application server connection in *trouble* mode
cacheAuditLost	System unable to record audit events
cacheLoggedError	Writing *severe* error to the console log
cacheLicenseExceed	License request exceeds available or allowed licenses
cacheAppAlert	Generating an application alert by calling the **%Monitor.Alert.External** method

There are additional events if you are using Ensemble. See Using WMI with Ensemble for details.

The iscprov.dll handles all communications between WMI and Caché and implements both the Caché WMI instance provider and event provider. The Caché installation registers the file and puts it in the C:\Program Files\Common Files\InterSystems\Cache directory.

The WMI service loads the provider DLL when requests are made and may unload it after it has been idle for a period of time. When the DLL is loaded it initiates communication with Caché and starts a **^WMI** server process in Caché. That server process is then terminated when the DLL is unloaded. Depending on how often your script or management application intends to collect information, you may want to change the default "lifetime" period for instance providers, which is only thirty seconds. Using the WMI CIM Studio from the \root namespace, expand the class hierarchy for __SystemClass, __CacheControl, and __ObjectProviderCacheControl. Change the value of the ClearAfter property to allow the Caché instance provider to remain loaded for a longer period.

Caché writes important messages from the **^WMI** server process in the cconsole.log file. It also logs any errors that occur in the iscprov.dll (which implements the WMI Instance Provider and Event Provider for Caché) in the iscwmi.log file in the \WINDOWS\system32\WBEM\Logs\ directory.

C.2.1 Using WMI with Ensemble

Ensemble also stores its WMI classes in the root\cache namespace in the WMI repository. Once you compile the Ensemble WMI classes and they are in the WMI repository, you can manage them in the same manner as described in Using WMI with Caché.

The Ensemble_LogEvent class is defined as a subclass of the __ExtrinsicEvent class. Ensemble sends a WMI Ensemble_LogEvent instance whenever it posts an alert to the Ensemble Event Log.

C.3 Generating a WMI Documentation Text File

You can generate a text file of the classes and properties that InterSystems provides for WMI by using the **doc** entry point of the Caché **^WMIMOF** routine. For example:

```
%SYS>Do doc^WMIMOF

File: <IscProv.txt>
```

Press **Enter** to create the IscProv.txt file in the Mgr directory:

```
Creating doc file for WMI Classes ...
 ;;DatabaseSystem;;;;Cache;
 ;;CommonDatabase;;;;Cache;
 ;;SystemStatistics;Abstract;;;Cache;
 ;;DatabaseStatistics;;SystemStatistics;;Cache;
 ;;ECPClientStatistics;;SystemStatistics;;Cache;
 ;;ECPServerStatistics;;SystemStatistics;;Cache;
 ;;WriteDemonStatistics;;SystemStatistics;;Cache;
 ;;GlobalBufferStatistics;;SystemStatistics;;Cache;
 ;;ResourceSeizeStatistics;;SystemStatistics;;Cache;
 ;;ECPConnection;Abstract;;;Cache;
 ;;ECPServerConnection;;ECPConnection;;Cache;
 ;;ECPClientConnection;;ECPConnection;;Cache;
 ;;ECPServerConnectionStats;;;;Cache;
 ;;ShadowJournal;Abstract;;;Cache;
 ;;ShadowSourceConn;;ShadowJournal;;Cache;
 ;;ShadowDestServer;;ShadowJournal;;Cache;
 ;;Event;;__ExtrinsicEvent;;Cache
 ;;LogEvent;;__ExtrinsicEvent;WMIMOF3;Ensemble
 ;;Production;;;WMIMOF3;Ensemble;
 ;;EventLog;;;WMIMOF3;Ensemble;
%SYS>
```

D

Monitoring Caché Using Web Services

This appendix introduces and briefly describes how to use Caché support for the WS-Management specification, which enables you to remotely monitor a Caché instance via SOAP.

D.1 Overview of Caché Support for WS-Monitoring

Following the WS-Management specification, the SYS.WSMon package provides a web service that you can use to remotely monitor a Caché instance. It is functionally similar to the SNMP interface (see the "Monitoring Caché Using SNMP" appendix of this guide) and WMI interface (see the "Monitoring Caché Using WMI" appendix of this guide), but uses the built-in Caché web services support.

The support for WS-Management includes the following elements:

- The *Caché Monitoring Web Service* (SYS.WSMon.Service) that provides methods that return information about a Caché instance.
- A Caché web client (SYS.WSMon.Client) that can invoke methods in this Monitoring Web Service or in the Monitoring Web Service of another Caché instance.

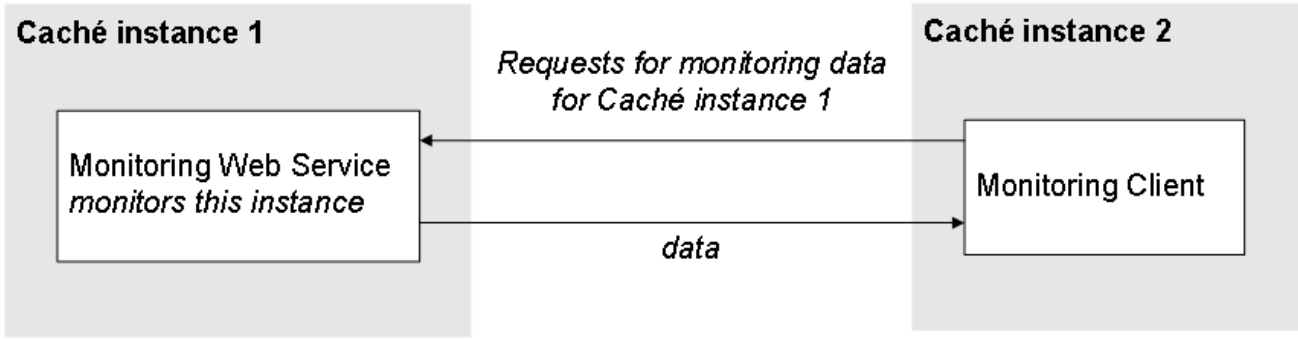
  Instead of using this web client, you can create your own web client, possibly using third-party technology.
- Several XML-enabled classes that this web service and client use to represent monitoring information.

  These classes include SYS.WSMon.wsEvent, which can represent events.
- A *sample event sink web service* (SYS.WSMon.EventSink) that can receive and process events. Via a SOAP call, you can subscribe to this sample event sink service so that it will receive events from any Monitoring Web Service.

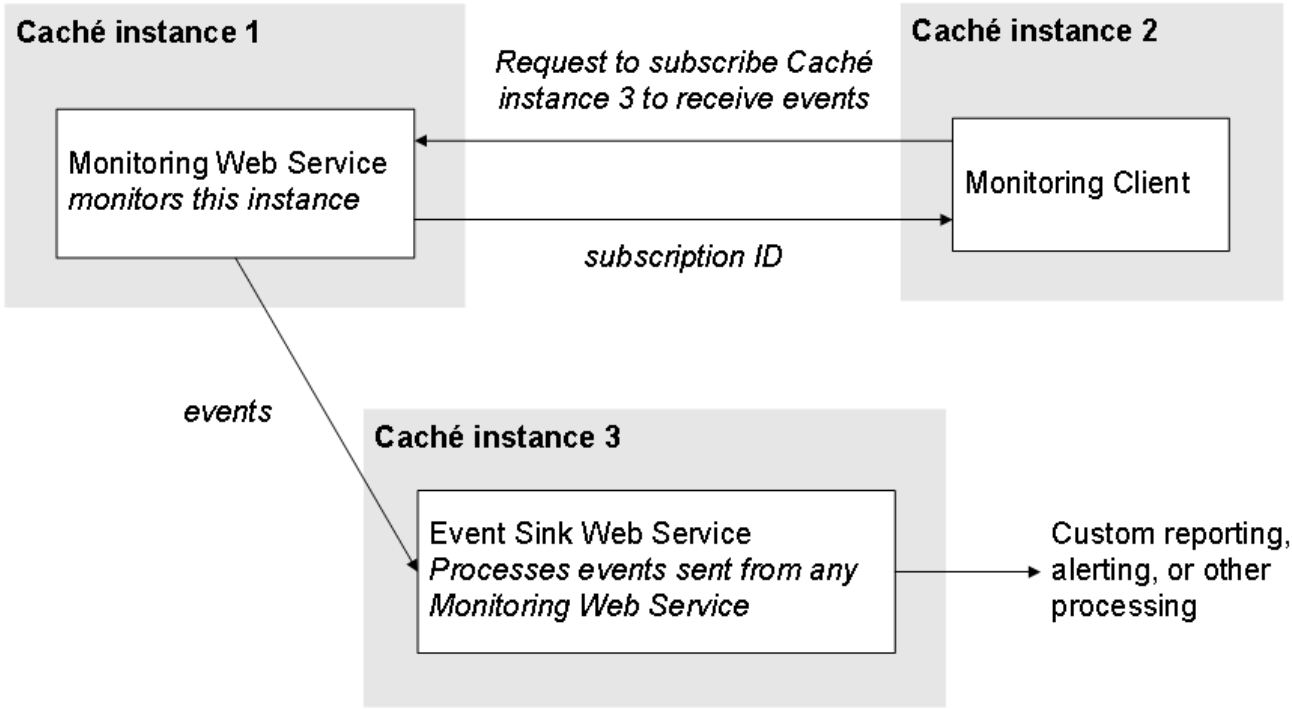
  Instead of using this sample, you can create your own, possibly using third-party technology.

These classes are available only in the %SYS namespace.

For basic monitoring, you can use the Caché Monitoring Web Service with a web client in another instance as follows:



In more advanced cases, the web client subscribes an event sink service, possibly running on another Caché instance. For example:



Your event sink web service can perform any processing needed by the business.

Note that Atelier provides the SOAP Wizard, which can generate web services and web clients from a WSDL. For details on using this wizard, see *Creating Web Services and Web Clients in Caché*. (Many third-party technologies also provide similar wizards.)

D.2 Support Details

InterSystems supports the following parts of the WS-Management specification:

- wxf:Get
- wsen:Enumerate
- wsen:Pull
- wsen:Release

- wse:Subscribe
- wse:Renew
- wse:Unsubscribe

For more information, see the WS-Management specification (https://www.dmtf.org/standards/published_documents/DSP0226_1.1.pdf).

D.3 URL for the Monitoring Web Service

For a given Caché instance, the Caché Monitoring Web Service is available at the following URL:

```
http://server:port/csp/sys/SYS.WSMon.Service.cls
```

Where *server* is the server on which Caché is running and *port* is the port that the Caché web server uses. For example:

```
http://localhost:57772/csp/sys/SYS.WSMon.Service.cls
```

Similarly, the WSDL for this web service is available at the following URL:

```
http://server:port/csp/sys/SYS.WSMon.Service.cls?WSDL=1
```

D.4 Web Methods of the Monitoring Web Service

The SYS.WSMon.Service class provides the following web methods:

EnumBuffer()

```
method EnumBuffer() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates the statistics for all buffer sizes. For this instance, the dataset uses the Sample() class query of the SYS.Stats.Buffer class.

For information on working with %XML.DataSet, see the chapter "Using Datasets in SOAP Messages" in *Creating Web Services and Web Clients in Caché* or see the class reference for %XML.DataSet.

Also see the class reference for SYS.Stats.Buffer.

EnumDatabase()

```
method EnumDatabase() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates all databases for this instance. For this instance, the dataset uses the List() class query of the SYS.WSMon.wsDatabase class.

See the comments for **EnumBuffer()** and see the class reference for SYS.WSMon.wsDatabase.

EnumResource()

```
method EnumResource() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates statistics for all system resource seizes. For this instance, the dataset uses the Sample() class query of the SYS.Stats.Resource class.

See the comments for **EnumBuffer()** and see the class reference for SYS.Stats.Resource.

EnumWriteDaemon()

```
method EnumWriteDaemon() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates statistics for all write daemons. For this instance, the dataset uses the **Sample()** class query of the SYS.Stats.WriteDaemon class.

See the comments for **EnumBuffer()** and see the class reference for SYS.Stats.WriteDaemon.

EventCancel()

```
method EventCancel(id) as %Integer
```

Cancels the subscription for a given web service; see **EventSubscribe()**.

EventSubscribe()

```
method EventSubscribe(location) as %String
```

Subscribes the given web service to receive information about events in this Caché instance. This can be your own web service or can be the SYS.WSMon.EventSink web service, which is provided as an example. If you create your own web service, it must follow the WSDL of the SYS.WSMon.EventSink web service.

For *location*, specify the URL needed to invoke the **EventSink()** method of the web service. For SYS.WSMon.EventSink, you might specify *location* as the following:

```
http://server:port/csp/sys/SYS.WSMon.EventSink.cls
```

Where *server* is the server on which Caché is running, and *port* is the port that Caché uses.

For each event, Caché will attempt to call **EventSink()** method of the given web service, sending an instance of SYS.WSMon.wsEvent.

This method returns an ID that you can use to cancel the subscription; see **EventCancel()**.

GetDisk()

```
method GetDisk() as SYS.Stats.Disk
```

Returns an instance of SYS.Stats.Disk that contains metrics of disk usage for globals for this instance.

See the class reference for SYS.Stats.Disk.

GetECPAppSvr()

```
method GetECPAppSvr() as SYS.Stats.ECPAppSvr
```

Returns an instance of SYS.Stats.ECPAppSvr that contains ECP application server metrics for this instance.

See the class reference for SYS.Stats.ECPAppSvr.

GetECPDataSvr()

```
method GetECPDataSvr() as SYS.Stats.ECPDataSvr
```

Returns an instance of SYS.Stats.ECPDataSvr that contains ECP database server metrics for this instance.

See the class reference for SYS.Stats.ECPDataSvr.

GetGlobal()

```
method GetGlobal() as SYS.Stats.Global
```

Returns an instance of SYS.Stats.Global that contains global metrics for this instance.

See the class reference for SYS.Stats.Global.

GetRoutine()

```
method GetRoutine() as SYS.Stats.Routine
```

Returns an instance of SYS.Stats.Routine that contains routine metrics for this instance.

See the class reference for SYS.Stats.Routine.

GetSystem()

```
method GetSystem() as SYS.WSMon.wsSystem
```

Returns an instance of SYS.WSMon.wsSystem that contains system information about the Caché instance.

See the class reference for SYS.WSMon.wsSystem.

D.5 Monitoring Web Client

The SYS.WSMon.Client class and related classes are a Caché web client that can invoke methods of SYS.WSMon.Server web service in the same Caché instance or another Caché instance.

This web client class uses the following LOCATION parameter:

```
Parameter LOCATION = "http://server:port/csp/sys/SYS.WSMon.Service.cls"
```

Where *server* is the server on which Caché is running and *port* is the port that the Caché web server uses.

Use this web client in the same way that you use other Caché web clients:

1. Create an instance of the web client class.
2. Set its Location property if needed.

   This is necessary if the SYS.WSMon.Server web service that you want to use is on a different machine than the client, or if it uses a port other than 57772.
3. Set other properties if needed.

   See *Creating Web Services and Web Clients in Caché*.
4. Invoke a web method.
5. Examine the value returned by the web method.

   The details depend on the web method you invoke; see the section "Web Methods of the Monitoring Web Service" and see the class reference for the return types.

The following shows an example Terminal session:

```
SAMPLES>set $namespace = %sys"

%SYS>Set client=##class(SYS.WSMon.Client).%New()

%SYS>Set client.Location="http://localhost:57799/csp/sys/SYS.WSMon.Service.cls"

%SYS>Set myroutinestats=client.GetRoutine()

%SYS>Write myroutinestats.RtnCallsLocal
19411581
%SYS>Write myroutinestats.RtnCallsRemote
0
%SYS>Write myroutinestats.RtnCommands
432764817
%SYS>
```

More typically, you create and use the client programmatically, perhaps to retrieve data for display in a user interface.

Note: Remember that the SYS.WSMon package is available only in %SYS namespace, which means that you must be in that namespace to perform the steps described here.

D.6 Processing Events

Caché provides a sample web service (SYS.WSMon.EventSink) that can receive and process events sent by any Caché Monitoring Web Service. You can use this web service or create and use your own.

D.6.1 Using the Sample Event Sink Web Service

SYS.WSMon.EventSink is a sample Caché web service that can receive and process events.

For a given Caché instance, the Caché Monitoring Web Service is available at the following URL:

```
http://server:port/csp/sys/SYS.WSMon.EventSink.cls
```

Where *server* is the server on which Caché is running and *port* is the port that the Caché web server uses.

This web service has one method:

CacheEventSink()

```
Method CacheEventSink(event As SYS.WSMon.wsEvent) As %Integer
```

On Windows platforms, this sample method displays a popup window when an event occurs; for other platforms, it adds an entry to `^SYS("MONITOR","WSMON","EVENT_RECEIVED",$h)`.

This method always returns 1.

To subscribe this sample service so that it will receive events from the Monitoring Web Service, do the following in the Terminal:

```
SER>set $namespace = %sys"

%SYS>Set client=##class(SYS.WSMon.Client).%New()

%SYS>Set eventsinklocation="http://localhost:57772/csp/sys/SYS.WSMon.EventSink.cls"

%SYS>Set subscriptionid=client.EventSubscribe(eventsinklocation)

%SYS>Write subscriptionid
CacheEventSubscription_2
```

Here *eventsinklocation* is the URL for the event sink web service that will process events.

D.6.2 Creating Your Own Event Sink Web Service

To create your own event sink web service, use the SOAP Wizard in Atelier to generate a web service from the following WSDL:

```
http://server:port/csp/sys/SYS.WSMon.EventSink.cls?WSDL=1
```

Where *server* is the server on which Caché is running and *port* is the port that the Caché web server uses.

For details on using this wizard, see *Creating Web Services and Web Clients in Caché*.

Then modify the **CacheEventSink()** method in the generated web service to include your custom logic.

E

Monitoring Caché Using the cstat Utility

This appendix provides an overview of how to use the **cstat** utility. It is intended as an introduction for new users and a reference for experienced users.

Important: When using this utility, you should consult with the InterSystems Worldwide Response Center (WRC) for guidance about specifying appropriate **cstat** options and assistance in interpreting the data produced by the utility.

cstat is a C executable that is distributed with Caché. It is a diagnostic tool for system level problems, including Caché hangs, network problems, and performance issues. When run, **cstat** attaches to the shared memory segment allocated by Caché at start time, and displays InterSystems' internal structures and tables in a readable format. The shared memory segment contains the global buffers, lock table, journal buffers, and a wide variety of other memory structures which need to be accessible to all Caché processes. Processes also maintain their own process private memory for their own variables and stack information. The basic display-only options of **cstat** are fast and non-invasive to Caché.

CAUTION: More advanced (undocumented) options may alter shared memory and should be used with care. These advanced options should be used only at the direction of InterSystems Support personnel; for information, contact the InterSystems Worldwide Response Center (WRC).

This appendix contains the following sections covering **cstat**:

E.1 Basics of Running cstat

In the event of a system problem, the **cstat** report is often the most important tool that InterSystems has to determine the cause of the problem. Use the following guidelines to ensure that the **cstat** report contains all of the necessary information:

- Run **cstat** at the time of the event.
- Use the Diagnostic Report task or CacheHung script unless directed otherwise by InterSystems support personnel.
- Check the contents of the **cstat** report to ensure it is valid.

Since **cstat** is a separate executable file included with Caché, it is run outside of Caché, at an operating system prompt. Therefore, the details of running it depend on the operating system:

Running **cstat** with no options is not a common way to run it, but doing so produces a basic report which is the equivalent of running it with the following default options:

- `-f` (global module flags)
- `-p` (PID table)
- `-q` (semaphores)

For information about **cstat** options, see Running cstat with Options.

E.1.1 Running cstat on Windows

The **cstat`** executable is located in Caché instance's \bin directory. Starting with a Windows command prompt running as Administrator, you can run it as follows:

```
C:\>cd install-dir\bin
C:\install-dir\bin>cstat
```

If you run **cstat** from a directory other than the instance's \bin or \mgr, you must include the `-s` argument to specify the location of the \mgr directory. For example:

```
C:\Users>\install-dir\bin\cstat -s\install-dir\mgr
```

E.1.2 Running cstat on UNIX®

The **cstat`** executable is located in the Caché instance's Bin directory. You can run it from another directory, but unless you are in the instance's mgr or *Bin* directory, you must include the `-s` argument to specify the location of the mgr directory. Starting with a Unix® command prompt, running as root, change to the Bin directory or the mgr directory and run the **cstat** command as follows:

```
bash-3.00$ ./cstat
```

From the Caché installation directory, the command would be as follows:

```
bash-3.00$ ./bin/cstat -smgr
```

You can also invoke **cstat** via the **ccontrol** command, which can be run from any directory as shown in the following example:

```
bash-3.00$ ccontrol stat Cache_Instance_Name
```

where *Cache_Instance_Name* is the name of the Caché instance on which you are running **cstat**.

E.2 Running cstat with Options

Running **cstat** without options produces a basic report. Generally, you run **cstat** to obtain specific information. To specify the information you want, add or subtract options, as follows:

- To include (turn on) an option, specify a flag followed by a `1` (or other level).
- To exclude (turn off) an option, specify a flag followed by a `0`.

For example, to include the Global File Table (GFILETAB) section in the **cstat** report, use the `-m1` option:

```
C:\MyCache\Bin\cstat -m1
```

or, to turn off the default basic options, use the `-a0` option:

```
C:\CACHE\Bin\cstat -a0
```

Many options have more detailed levels than `0` and `1`. These additional levels are described as having "bits," which are displayed in decimal as powers of two and control specific types of information about the option. For example, the basic `-p` option, which displays the PID table, is turned on with a `1`; however, using a `2` adds a **swcheck** column, a `4` adds a **pstate** column. and so on. These bits can be combined; for example, if you want to see the information displayed by both the `2` and `4` bits, specify `-p6`. To ask for all bits, use `-1`, as follows:

```
bash-3.00$ ./cstat -p-1
```

In addition, multiple flags can be combined in a single **cstat** command. For example, the following command turns off the basic options, then turns on all bits for the global module flags and PID table, as well as a detailed level for the GFILETAB:

```
bash-3.00$ ./cstat -a0 -f-1 -p-1 -m3
```

It is common for **cstat** commands to have many flags when you start diagnosing a complex problem; however, the options that make modifications are typically used alone. For example, the `-d` option requests a process dump; before using this option, you might run **cstat** with multiple options to identify the process to dump, but when using `-d`, typically no other options are selected.

The cstat Options table describes the options that you can use with the **cstat** command.

Note: For assistance in interpreting the data produced by the **cstat** options described in this table, contact the InterSystems Worldwide Response Center (WRC).

Table V–1: cstat Options

Option	Description
–a[0/1]	Displays "all" information as described in the Running cstat with Options section of this chapter.
–b[bits]	Displays information about global buffer descriptors blocks (BDBs). You can specify a combination of the following bits: • `1` (all) • `2` (cluster) • `4` (ECP server) • `8` (ECP client) • `16` (block contents) • `64` (check block integrity) • 128 (block and LRU summary) **Note:** See also `-l`. Running **cstat -b64** may require extra time.

Option	Description
`-c[bits]`	Displays counters, which are statistics on system performance. You can specify a combination of the following bits: • `1` (global) • `2` (network) • `4` (lock) • `8` (optim) • `16` (terminal) • `32` (symtab) • `64` (journal) • `128` (disk i/o) • `256` (cluster) • `262144` (bshash) • `2097152` (job cmd) • `4194304` (sem) • `8388608` (async disk i/o) • `16777216` (fsync) • `33554432` (obj class) • `67108864` (wd) • `134217728` (bigstr) • `268435456` (swd) • `536870912` (sort) • `1073741824` (symsave) • `2147483648` (freeblkpool)
`-d[pid,opt]`	Creates dump of Caché processes. You can specify the following options: • `0` (full); default • `1` (partial)
`-e[0/1/2]`	Displays the Caché system error log (see Caché System Error Log in the "Monitoring Caché Using the Management Portal" chapter); `-e2` displays additional process information (in hex).

Option	Description
-f[bits]	Displays global module flags. You can specify a combination of the following bits: • 1 (basic) • 64 (resources) • 128 (with detail) • 256 (account detail) • 512 (incstrtab) • 1024 (audit)
-g[0/1]	Displays **^GLOSTAT** information; for information see the "Gathering Global Activity Statistics Using ^GLOSTAT" chapter in this document.
-h	Displays **cstat** usage information.
-j[0/1/2/3/4/5/6]	Displays the journal system master structure, which lists information about journaling status. -j32 displays mirror server information.
-k	Displays information about prefetch daemons used by the **$PREFETCHON** function; see $PREFETCHON in the *Caché ObjectScript Reference.*
-l[bits]	Displays information about least recently used (LRU) global buffer descriptor block (BDB) queue, but not the contents of the BDBs. You can specify a combination of the following bits: • 1 (all) • 2 (cluster) • 4 (ECP server) • 8 (ECP client) • 16 (block contents) • 32, but not 1 (most recently used (MRU) order) **Note:** See also -b.
-m[0/1/3/4/8/16]	Displays Global File Table (GFILETAB), which contains information about all databases, listed by SFN, that have been mounted since the instance of Caché started up. You can specify a combination of the following bits: • 3 (additional details) • 4 (volume queues) • 8 (disk device id table) • 16 (systems remotely mounting this database)
-n[0/1]	Displays information about network structures and local/remote SFN translations; cstat -n-1 also displays namespace structures.

Option	Description
-o1	Clears the resource statistics displayed by **cstat -c** to reestablish a base situation without rebooting Caché. No output is produced.
-p[bits]	Displays information about processes that are running in Caché. The information is obtained from the process ID table (PIDTAB). You can specify a combination of the following flags: • 2 (swcheck) • 4 (pstate and %SS) • 5 (NT mailbox locks); Windows only • 8 (js sum) • 16 (js list) • 32 (grefcnt info) • 64 (gstatebits) • 128 (gstate summary) • 256 (jrnhib) • 512 (transaction summary) • 1024 (pidflags) • 2048 (pgbdbsav); additionally dumps pgshared table • 4096 (freeblk table)
-q[0/1]	Displays information about hibernation semaphores.
-s[dir]	Specifies the directory containing the **cstat** executable when running the command from other than the mgr or bin directories.
-t[seconds]	Runs **cstat** repeatedly in a loop every *seconds* seconds until halted. Only the global module flags section is displayed, as when -f1 is specified.
-u[bits]	Displays information about Caché locks stored in the lock table (see Monitoring Locks in the "Monitoring Caché Using the Management Portal" chapter of this guide). You can specify a combination of the following bits: • 1 (summary) • 2 (waiters) • 4 (intermediate) • 8 (detail) • 16 (watermark) • 32 (buddy memory) • 64 (resource info)

Option	Description
`-v1`	Ensures that the Caché executable associated with the shared memory segment **cstat** is being run on and the **cstat** executable are from the same version; if not, **cstat** will not run.
`-w[bits]`	Displays information about BDBs in write daemon queues.
`-B[0/1]`	Displays, in hex, the contents of blocks held in GBFSPECQ.
`-C[0/1]`	Displays configuration information for inter-job communication (IJC) devices.
`-D[secs],[msecs][,0]`	Displays resource statistics over an interval of 'secs' seconds. Sample block collisions ever 'msec' milliseconds. **Note:** Resource information same as `-c`. The **^BLKCOL** utility, described in the "Monitoring Block Collisions Using ^BLKCOL" chapter of this guide, provides more detailed information about block collisions.
`-E[bits]`	Displays status of cluster on platforms that support clustering. You can specify a combination of the following bits: • `1` (vars) • `2` (write daemon locks) • `4` (enqinuse) • `8/16` (allenq)
`-G[bdb]`	Displays, in hex, the contents of the global buffer descriptors and the global buffer for a specific buffer descriptor block (BDB). **Note:** Same as `-H` except that the information is displayed by BDB.
`-H[sfn],[blk]`	Displays, in hex, the contents of the global buffer descriptors and the global buffer for a specific system file number (sfn) and block number (blk) pair. **Note:** Same as `-G` except that the information is displayed by system file number and block number pair. The block must be in the buffer pool.
`-I[0/1]`	Displays the incremental backup data structures.
`-L[0/1]`	Displays the license. **Note:** Same as **^CKEY** and **%SYSTEM.License.CKEY** method.
`-M[0/1]`	Displays the mailbox log. **Note:** Disabled by default. A special build is required to capture and log the mailbox messages; additional logging may be required.

Option	Description
`–N[value]`	Displays ECP network information. You can specify a combination of the following values: • `1` (client) • `2` (server) • `4` (client buffers) • `8` (server buffers) • `16` (client buffers, in detail) • `32` (user jobs awaiting answer) • `64` (server answer buffers details) • `128` (request global) • `256` (server send answer buffer details; not -1) • `1024` (dump server received request buffers) • `2048` (client trans bitmap) • `4096` (client GLO Q) • `8192` (request global reference dump, in hex) • `65536` (ECP blocks downloaded to clients) • `131072` (client released request buffer details; not -1)

Option	Description
`-R[value]`	Displays information about routine buffers in use (or changing), class control blocks (CCB), and least recently used (LRU) queues. You can specify a combination of the following values: • `1` (routine buffers in use) • `4` (RCT – changed routine table) • `8` (RCT detail) • `16` (`0x10`=all routine buffers) • `32` (`0x20`=LRU Q) • `64` (`0x40`=all CCB's) • `128` (`0x80`=invalidated CCB's) • `0x100` (invalidated subclasses) • `0x200` (buffer address) • `0x400` (buffer descriptors) • `0x800` (procedure table and cached routines buffer number) • `0x1000` (process cached routine names) • `0x2040` (CCB's and CCB details • `0x4000` (cls NS cache) • `0x6000` (cls NS cache details) • `0x8000` (validate shm cls cache) • `0x10000` (dump all class hierarchy) • `0x20000` (dump all class hierarchy details) • `0x40000` (dump process class and routine statistics) • `0x80000` (process cached class names)
`-S[bits]`	Displays information about the cause of a hang based on a self diagnosis of whether or not the system is hung. You can specify a combination of the following bits: • `1` (display diagnosis) • `2` (partial process dump for suspect jobs) • `4` (full process dump for first suspect job and partial dumps for other suspect jobs) **Note:** In a cluster, this option should be run all cluster members.
`-T[0/1]`	Displays hex values of many in-memory tables, including National Language Settings (NLS) tables.

Option	Description
-V[pid]	Displays variables that are part of the process memory structures; of limited value unless you have access to the source code. **Note:** Windows only. Run from the directory that contains the pid.dmp file.
-W	Performs the same function as the **Backup.General.ExternalThaw()** classmethod, and may be used to resume the write daemon after **Backup.General.ExternalFreeze()** has been called in cases in which a new Caché session cannot be started. (See External Backup in the "Backup and Restore" chapter of the *Caché Data Integrity Guide* for information on the use of these methods.) This option will not unfreeze the write daemon from any hang or suspension caused by anything other than a backup. Use of this option is recorded in the console log.
-X[0/1]	Displays the contents of the device translation table. It is organized by device number and shows both the numeric and plaintext class identifiers.

E.3 Viewing cstat Output

cstat data can be viewed immediately (via a terminal) or redirected to an output file (see cstat Text File in this appendix) for later analysis. The most common methods for viewing the data are:

- cstat Text File
- Diagnostic Report Task
- CacheHung Script
- ^pButtons Utility

Note: When Caché is forcibly shut down, **cstat** is run in order to capture the current state of the system. The output is added to the console log as part of the emergency shutdown procedure.

E.3.1 cstat Text File

cstat reports can be redirected to a file instead of the terminal, which might be useful if you want to collect a set of **cstat** options that are not provided by one of the Caché tools (Diagnostic Report Task, CacheHung Script, ^pButtons Utility) or if you are having trouble running those tools.

E.3.2 Diagnostic Report Task

The Diagnostic Report task creates an HTML log file containing both basic and advanced information, which can be used by the InterSystems Worldwide Response Center (WRC) to resolve system problems. For information about the Diagnostic Report task, including the **cstat** options that it uses, see the "Using the Caché Diagnostic Report" chapter in this guide.

Note: The Diagnostic Report task cannot be run on a hung system; if your system is hung, see CacheHung Script in this appendix.

E.3.3 CacheHung Script

The **CacheHung** script is an OS tool used to collect data on the system when a Caché instance is hung. The name of the script, which is located in the *install-dir*/Bin directory, is platform-specific, as specified in the following table:

Platform	Script name
Microsoft Windows	**CacheHung.cmd**
UNIX®/Linux	**CacheHung.sh**

The **CacheHung** script should be run with Administrator privileges. Like the Diagnostic Report Task, the **CacheHung** script runs **cstat** twice, 30 seconds apart, in case the status is changing, and bundles the reports into an html file together with the other collected data. The **cstat** reports taken from **CacheHung** use the following options:

```
cstat -e2 -f-1 -m-1 -n3 -j5 -g1 -L1 -u-1 -v1 -p-1 -c-1 -q1 -w2 -E-1 -N65535
```

CacheHung also runs a third **cstat** using only the -S2 option, which it writes to a separate section of output called "Self-Diagnosis." The -S2 option causes suspect processes to leave mini-dumps; therefore, running **CacheHung** is likely to collect information about the specific processes responsible for the hang, whereas simply forcing the instance down does not collect this information.

In addition, **CacheHung** generates **cstat** output files that are often very large, in which case they are saved to separate txt files. Remember to check for these files when collecting the output.

E.3.4 ^pButtons Utility

The **^pButtons** utility collects detailed performance data about a Caché instance and the platform on which it is running. It runs inside Caché for a configurable amount of time, collects samples over the that interval, and generates a report when it finishes. For information about the **^pButtons** utility, including the **cstat** options that it uses, see the "Monitoring Performance Using ^pButtons" chapter in this guide.