



Caché Monitoring Guide

Version 5.2
01 September 2006

Caché Monitoring Guide

Caché Version 5.2 01 September 2006

Copyright © 2006 InterSystems Corporation.

All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.



The Caché product and its logos are registered trademarks of InterSystems Corporation.



The Ensemble product and its logos are registered trademarks of InterSystems Corporation.



The InterSystems name and logo are trademarks of InterSystems Corporation.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

Caché, InterSystems Caché, Caché SQL, Caché ObjectScript, Caché Object, Ensemble, InterSystems Ensemble, Ensemble Object, and Ensemble Production are trademarks of InterSystems Corporation. All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel: +1 617 621-0700

Fax: +1 617 374-9391

Email: support@InterSystems.com

Table of Contents

Introduction	1
1 Monitoring Caché Using the System Management Portal	3
1.1 Monitoring System Dashboard Indicators	3
1.2 Monitoring System Performance	7
1.3 Monitoring Locks	8
1.4 Monitoring Log Files	10
1.5 Viewing CSP Sessions	11
1.6 Viewing Background Tasks	11
2 Using the Caché Monitor	13
2.1 Monitor Architecture	13
2.2 System Monitor	14
2.2.1 System Monitor Manager	14
2.3 Application Monitor	16
2.4 Caché Metrics	16
2.5 Alerts	17
2.6 Managing the Application Monitor	17
2.6.1 Manage Application Monitor	18
2.6.2 Manage Monitor Classes	18
2.6.3 Manage Email Options	19
2.6.4 Manage Alerts	19
2.7 Writing User-defined Monitor Classes	20
2.7.1 Example of a User-defined Monitor Class	21
2.8 Generating Metrics	24
2.9 Viewing Metrics Data	25
2.10 Caché Monitor Errors	25
3 Gathering Global Activity Statistics Using ^GLOSTAT	29
3.1 Running ^GLOSTAT	29
3.2 Overview of ^GLOSTAT Statistics	31
3.3 Examples of ^GLOSTAT Output	32
3.3.1 Example A	33
3.3.2 Example B	33
3.3.3 Example C	34

3.3.4 Example D	34
3.3.5 Example E	37
3.3.6 Example F	37
3.3.7 Example G	38
4 Monitoring System Performance Using ^PERFMON	41
4.1 Using ^PERFMON	42
4.1.1 Start	43
4.1.2 Stop	44
4.1.3 Pause	44
4.1.4 Resume	45
4.1.5 Clear	45
4.1.6 Report	46
4.2 Report Examples	49
5 Examining Routine Performance Using ^%SYS.MONLBL	51
5.1 Start Monitoring	51
5.2 Monitoring Options	54
5.2.1 Report Statistics	54
5.3 Sample Output	55
5.4 Programming Interface	57
Appendix A: Monitoring Caché Using BMC PATROL	59
A.1 Running PATROL with Caché	59
A.1.1 Configure PATROL Settings	60
A.1.2 Caché PATROL Routine	61
A.2 Caché PATROL Knowledge Modules	62
A.2.1 Adding Caché Modules to PATROL	62
A.3 Caché Metrics Used with BMC PATROL	63
Appendix B: Monitoring Caché Using SNMP	69
B.1 Using SNMP with Caché	69
B.2 Managing SNMP in Caché	70
B.3 Caché as a Subagent	71
B.4 Caché MIB Structure	72
B.4.1 Extending the Caché MIB	73
Appendix C: Monitoring Caché Using WMI	77
C.1 Configuring WMI in Caché	77
C.2 Using WMI with Caché	78

List of Tables

System Performance Indicators	4
ECP and Shadowing Indicators	5
System Time Indicators	5
System Usage Indicators	6
Errors and Alerts Indicators	6
Licensing Indicators	7
Task Manager Upcoming Tasks	7
System Usage Statistics	8
Lock Modes	10
Statistics Produced by ^GLOSTAT	31
Metrics for Custom Report	46
Caché PATROL Metrics	64

Introduction

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 System Management Portal](#) — The System Dashboard of the System Management Portal contains many metrics that show you the state of your Caché instance at a glance. You can also click a details link to see more details of these metrics. Many of these metrics are also accessible in other locations in the Operations list of the System Management Portal.
- [Using the Caché Monitor](#) — The Caché Monitor is a system monitor facility that provides services for gathering and monitoring performance and system data, generating alerts, handling notifications, and generating and maintaining a persistent repository of such data.

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

- [Gathering Global Activity Statistics Using ^GLOSTAT](#)
- [Monitoring System Performance Using ^PERFMON](#)
- [Examining Routine Performance Using ^%SYS.MONLBL](#)

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](#)

1

Monitoring Caché Using the System Management Portal

You can monitor many aspects of your Caché instance starting at the System Dashboard of the System Management Portal. From this page, 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 Locks](#)
- [Monitoring Log Files](#)
- [Monitoring System Performance](#)
- [Viewing CSP Sessions](#)
- [Viewing Background Tasks](#)

1.1 Monitoring System Dashboard Indicators

The **[Home] > [System Dashboard]** page of the System Management Portal displays real-time status of key system performance indicators in 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 any of these indicators to display a description of the metric in the bottom detail box at the lower left corner of the page.

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)).

In the description box, click the details link to display the **[Home] > [System Usage]** page in the bottom detail box. See the [Monitoring System Performance](#) section for details.

ECP and Shadowing Indicators

Indicator	Definition
Application Servers	Summary status of ECP (Enterprise Caché Protocol) application servers connected to this system.
Application Server Traffic	Most recently measured ECP application server traffic in bytes/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 Clients	Summary status of shadow clients connected to this system.
Shadow Servers	Summary status of shadow servers connected to this system as a data source.

For more information on the first four indicators, the ECP indicators, see the “[Configuring Distributed Systems](#)” chapter of the *Caché Distributed Data Management Guide*.

In the description box for the last two indicators, the shadow indicators, click the details link to display the **[Home] > [Shadow Servers]** page. For more information on shadowing, see the “[Shadow Journaling](#)” chapter of the *Caché High Availability Guide*.

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 **[Home] > [Backup]** page. For more information on developing a backup plan, see the “[Backup and Restore](#)” chapter of the *Caché High Availability Guide*.

System Usage Indicators

Indicator	Definition
Database Space	Indicates whether there is a reasonable amount of disk space available for database files. Clicking details displays the [Home] > [Databases] > [Freespace] page.
Journal Space	Indicates whether there is a reasonable amount of disk space available for journal files. Clicking details displays the [Home] > [Journals] page.
Journal Entries	Number of entries written to the system journal. Clicking details displays the [Home] > [Journals] page.
Lock Table	Indicates the current status of the system Lock Table. Clicking details displays the [Home] > [Locks] page.
Write Daemon	Indicates the current status of the system Write daemon.
Processes	Most recent number of running processes. Clicking details displays the [Home] > [Processes] page.
CSP Sessions	Most recent number of CSP sessions. Clicking details displays the [Home] > [CSP Sessions] page.
Most Active Processes	Running processes with highest amount of activity (lines of code executed). Clicking details displays the [Home] > [Processes] page.

For more information on each of these topics, click **Help** under the InterSystems logo on the portal page displayed when you click for more details.

Errors and Alerts Indicators

Indicator	Definition
Serious Alerts	Number of serious alerts that have been raised. Clicking details displays the [Home] > [System Logs] > [View Console Log] page.
Application Errors	Number of application errors that have been logged. Clicking details displays the [Home] > [System Logs] > [View Application Error Log] page.

See the [Monitoring Log Files](#) section in this chapter for more details.

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.

In the description box, click the details link to display the **[Home] > [License Usage] > [License Activity Summary]** page. For more information on licensing, see the “[Managing Caché Licenses](#)” chapter of the *Caché System Administration Guide*

Task Manager Upcoming Tasks

Indicator	Definition
Upcoming Tasks	Lists a maximum of five tasks scheduled to run within the next twenty-four hours.
Task	Name of the upcoming task.
Time	Time the task is scheduled to run.
Status	Task status, one of: scheduled, completed, running.

In the description box, click the details link to display the **[Home] > [Task Manager] > [View 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

Navigate to the **[Home] > [System Usage]** page to view several metrics that help you monitor system performance. See the “[Gathering Global Activity Statistics with ^GLOSTAT](#)” chapter for an alternative method of retrieving these statistics.

System Usage Statistics

Statistic	Definition
Global references (all)	Logical count of accesses to globals, including Sets , Kills , \$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 (2-KB or 8-KB) read from disk for both global and routine references.
Block writes	Number of physical database blocks (2-KB or 8-KB) written to disk for both global and routine references.
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	
Last Update	Date and timestamp of the displayed statistics.

1.3 Monitoring Locks

Caché locks are created when a Caché process issues a **Lock** command on a Caché entity, such as a local 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.

To display or delete locks, navigate to the **[Home] > [Locks]** page from the **Operations** section of the System Management Portal home page. The table lists one row for each lock owner and lock waiter. If one lock is owned by more than one process, each owner has its own row.

The **LockTable** 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.
Mode/Count	Lock mode and count of the lock item. If the lock count is 1 the count is not displayed. For details, see the Lock Modes table.
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 the lock located, if it is the local system the column is blank.
Remote	Indicates whether the lock is located on a remote system. a zero (0) indicates the lock is local.
Remove	If this lock is removable, this option along with the Remove all locks for process option appears in the row. Click the appropriate button to remove the lock or all locks for the process.

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* guide.

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 System Management Portal:

1. Navigate to the **[Home] > [Configuration] > [Advanced Settings]** page and enter **Lock** in the **Filter** box to shorten the list.
2. In the [LockTableSize](#) row, click **Edit** to display the **Configuration Setting** page.
3. In the **Value** 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 **GenericHeapSize**. 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 **Apply Changes** to activate the new value when Caché restarts.

The following table lists the possible values of the mode portion of the **Mode/Count** column.

Lock Modes

Mode	Description
Exclusive	Exclusive lock mode.
Shared	Share lock mode.
LockZA	ZALLOCATE lock mode.
WaitLock	Waiting for exclusive lock mode.
WaitShare	Waiting for share lock mode.
WaitLockZA	Waiting for ZALLOCATE lock mode.
LockPending	Exclusive lock pending, waiting for server to grant the exclusive lock.
SharePending	Share lock pending, waiting for server to grant the share lock.
DelockPending	Delock pending, waiting for server to release the lock.
Lost	Lock lost due to network reset.

For more detailed information, see the SYS.Lock class documentation in the *Caché Class Reference*.

1.4 Monitoring Log Files

Caché reports general messages, system errors, certain operating system errors, and network errors through an operator console facility. For Caché Windows-based systems, there is no operator console as such and all console messages are sent to a log file, `cconsole.log`, in the system manager directory, typically `CacheSys\Mgr`. For Caché systems on UNIX-based or OpenVMS platforms, you can send operator console messages to the console log file or the console terminal.

The console log file is a plain text file and may be viewed with any editor or text viewer and by using the System Management Portal:

1. Navigate to the **[Home] > [System Logs]** page by clicking **System Logs** in the **Operations** column of the home page.
2. Click **View Console Log** under the **System Logs** column to display the text output of the console log file.

If you have any trouble starting Caché, it is recommended that you view the console log file. The `cconsole.log` file grows until it reaches a maximum size, when it is renamed `cconsole.old`. The value is configurable (default is 5 MB) from the **[Home] > [Configuration] > [Advanced Settings]** page of the System Management Portal. Update the number of MBs for the [ConsoleLogSize](#) setting.

You may also view log information about application errors and ODBC errors from the **[Home] > [System Logs] > [View Application Error Log]** and the **[Home] > [System Logs] > [View xDBC Error Log]** pages of the System Management Portal.

1.5 Viewing CSP Sessions

Navigate to the **[Home] > [CSP Sessions]** page to view all currently active CSP sessions.

1.6 Viewing Background Tasks

Navigate to the **[Home] > [Background Tasks]** page to view the logs of any completed or running background tasks.

2

Using the Caché Monitor

The Caché System Monitor is an extendable system monitor facility. It provides services for gathering and monitoring performance and system data, generating alerts, handling notifications, and generating and maintaining a persistent repository of such data.

2.1 Monitor Architecture

The Caché Monitor is a modular system that can run multiple instances of the Monitor, in multiple namespaces.

Each Monitor instance acts the same in each namespace in which it is activated:

- At startup time, the Monitor examines a list of which Monitor Metrics are active. For each active Metric, the Monitor reads metadata that tells the Monitor which Metrics classes to invoke, and the alerts that have been defined for each such class.
- The Monitor then invokes standard methods in each of the active Metrics classes. The Metrics data returned from each Metrics class is examined to determine the following:
 - If the data is marked as persistent, save the persistent sample class.
 - If an alert is defined on that class, pass the sample data to the alert handler. The alert handler determines if a threshold has been reached and, if so, execute a notification per the user instructions.

2.2 System Monitor

The System Monitor is a variant of the standard Caché Monitor, adapted to special system usage.

At Caché startup, a System Monitor is started in the %SYS namespace. This Monitor unconditionally monitors the internal Caché system traps which have been designed to work with the System Monitor. No persistent data is kept.

Alerts are handled in the same way as for the standard Monitor. An alert is generated for any system trap that occurs, and notification provided per the user-defined instructions for email notification. See below for instructions on setting email options.

See the Monitor Errors section on the system traps that are handled by the System Monitor.

2.2.1 System Monitor Manager

The System Monitor Manager (^MONMGR) is a character-based application that allows you to perform System Monitor management functions. It must be executed in the %SYS namespace, and sets parameters for System Monitor only.

The options provided in ^MONMGR are:

```
%SYS>Do ^MONMGR
```

- 1) Manage MONITOR
- 2) Set Email options
- 3) List Email options
- 4) Test Email options
- 5) Set Sample Interval
- 6) Exit

Option?

Start, Stop, Refresh the System Monitor. Refreshing the Monitor causes it to pick up new email options and sample interval, without having to restart the System Monitor.

Option? 1

- 1) Refresh MONITOR
- 2) Halt MONITOR
- 3) Start MONITOR
- 4) Exit

Option?

Email options provide for sending email on notification. The user specifies the following:

1. **Mail server** — The user must specify the name of the email server that handles email in their environment. The user's IT staff should be able to provide this information.
2. **Recipients** — A list of recipients of the email. The list must be a list of valid email addresses.
3. **Sender** — This is text that defines the sender of the email. It need not be a replyable email account.

```
Option? 2
Mail server? doc.server
Recipients? userdoc@company.com
Sender? userdoc@company.com
```

- 1) Manage MONITOR
- 2) Set Email options
- 3) List Email options
- 4) Test Email options
- 5) Set Sample Interval
- 6) Exit

```
Option? 3
```

```
Mail server: doc.server

Recipients: userdoc@company.com

Sender: userdoc@company.com
```

- 1) Manage MONITOR
- 2) Set Email options
- 3) List Email options
- 4) Test Email options
- 5) Set Sample Interval
- 6) Exit

```
Option? 4
```

```
Sending email on Mail Server doc.server
From: userdoc@company.com
To: userdoc@company.com
```

- 1) Manage MONITOR
- 2) Set Email options
- 3) List Email options
- 4) Test Email options
- 5) Set Sample Interval
- 6) Exit

```
Option?
```

The Monitor runs repeatedly once started. The default time between sample gathering and alerts processing is ten seconds. This time may be changed to any interval by setting this option (5).

2.3 Application Monitor

The Application Monitor provides for monitoring Caché system-defined and user-defined metrics. You first registers system classes in the local namespace, and create your own user-defined metric classes. Next, you activate the classes you want to actively monitor. You then start an instance of the Application Monitor (**%MONAPP**) in the local namespace.

When an Application Monitor is started in the local namespace, it searches for all classes that have been activated. For each such class, it creates sample data in local namespace globals. Monitor metadata is kept in globals `^ISCMonitor()`. Sample data is kept as persistent data for the sample class.

You may view this data by invoking class methods, or via a browser. All sample classes are automatically CSP-enabled.

See the sections on Managing the Application Monitor, and writing your own User-Defined Monitor Classes, below.

```
^ISCMonitor("Monitor", "AppInterval")=300
^ISCMonitor("Monitor", "Interval")=10
^ISCMonitor("Monitor", "Metric", "%Monitor.System.Dashboard", "Active")=1
^ISCMonitor("Monitor", "Recipients")=""
^ISCMonitor("Monitor", "SmtServer")=""
```

2.4 Caché Metrics

There are the following types of metrics with corresponding predefined Caché sample system metric classes. View the Caché Class Reference entry for a list of properties corresponding to the sample metrics in each category:

- *System Activity Counters* — %Monitor.System.Sample.SystemMetrics
- *Global Metrics* — %Monitor.System.Sample.Globals
- *Server Metrics* — %Monitor.System.Sample.Servers
- *Client Metrics* — %Monitor.System.Sample.Clients
- *Free Space Metrics* — %Monitor.System.Sample.Freespace
- *Audit Metrics* — %Monitor.System.Sample.AuditCount

- *Console Metrics* — %Monitor.System.Sample.Dashboard Monitors the cconsole.log file. Alerts may be defined on severity level. The System Monitor always monitors the cconsole.log file, and provides alerts for severity level of Severe or higher.

These classes monitor counters of system activity. You can obtain global metrics for all the globals in the current namespace (%Monitor.System.Globals) and system metrics for total system activity (%Monitor.System.SystemMetrics). You can also obtain global and system metrics relative to:

- *Routines* — %Monitor.System.Sample.Routines) – for each routine currently executing
- *Processes* (%Monitor.System.Sample.Processes)– for each Caché process

Similar functions that control the same **MONITOR** facility are available through the classes in the %Monitor.System package, which also allow 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 *Caché Class Reference* for more details.

2.5 Alerts

An alert specifies a list of properties in a class to monitor. An expression is specified, for example, “%1 < 100 && %1 > 20”. During Monitor sampling, the values of the properties in the class are evaluated. If the expression evaluates to true, an alert is triggered. A notification action is taken whenever an alert is triggered.

The expression provides for parameter substitution. %1 indicates the first property in the list of monitored properties, %2 the second property, etc.

An Alert is defined by using the Monitor Manager Alert management functions.

2.6 Managing the Application Monitor

The Monitor Manager (^%MONAPPMGR) is a character-based utility that allows the user to do Management functions. It may be executed in a user’s namespace, and sets parameters for that namespace only. In this way, multiple Managers, each with their own activated classes, and notification options, may all run at the same time.

```
%SYS>Do ^%MONAPPMGR
```

- 1) Manage Application Monitor
- 2) Manage Monitor Classes
- 3) Manage Email options
- 4) Manage Alerts
- 5) Exit

Option?

2.6.1 Manage Application Monitor

Option? 1

- 1) Start Application Monitor
- 2) Halt Application Monitor
- 3) Refresh Application Monitor
- 4) Activate/Deactivate a Monitor Class
- 5) Set Sample Interval
- 6) Exit

Option?

Start, Stop, Refresh the Monitor — Refreshing the Monitor causes it to pick up new lists of activated classes, and notification options, without having to restart the Monitor.

Activate/Deactivate a Monitor Class — Of all the classes defined in the namespaces as Monitor-enabled, only those that are marked as Active will actually be sampled by the Application Monitor

Interval — The Monitor runs repeatedly once started. The default time between sample gathering and alerts processing is 20 seconds. This time may be changed to any interval by setting this option.

2.6.2 Manage Monitor Classes

Option? 2

- 1) Activate/Deactivate a Monitor Class
- 2) List Monitor Classes
- 3) Register Monitor System Classes
- 4) Exit

Option?

1. **Activate/Deactivate a Monitor Class** Of all the classes defined in the namespaces as Monitor-enabled, only those that are marked as Active will actually be sampled by the Application Monitor
2. **List Monitor Classes** — Lists all the Monitor classes defined in the namespace, and whether or not they are activated.

3. Register Monitor System Classes — System classes must be registered in the local namespace in which they will be monitored. Registration sets Monitor globals in the local namespace that tells the Monitor of the activation status of defined classes.

```

Option? 2
Class                                     Active
-----
%Monitor.System.AuditCount               N
%Monitor.System.Clients                  N
%Monitor.System.Dashboard                Y
%Monitor.System.Database                 N
%Monitor.System.Freespace                N
%Monitor.System.Globals                  N
%Monitor.System.Processes                N
%Monitor.System.Routines                  N
%Monitor.System.Servers                  N
%Monitor.System.SystemMetrics            N

```

- 1) Activate/Deactivate a Monitor Class
- 2) List Monitor Classes
- 3) Register Monitor System Classes
- 4) Exit

Option?

2.6.3 Manage Email Options

Option? 3

- 1) Set Email options
- 2) List Email options
- 3) Test Email options
- 4) Exit

Option?

Email notification may be configured in the same way as for the System Monitor. See the section on the System Monitor for email definitions.

When alerts are defined, they may be configured to use email as a means of notification. See Alerts below.

2.6.4 Manage Alerts

Option? 4

- 1) Create Alert
- 2) Delete Alert
- 3) List Alerts
- 4) Exit

Option?

1) Create Alert — To create an alert, specify the following:

Input Field	Purpose
Alert Name	Name of the alert
Application Name	Qualifier to group alerts. Give a set of alerts a group name to gather those alerts into a user-defined “Application.”
Action	One of: None, Notify via email, or Notify via user-class method.
Notify Method	(Full class name and method, for example, A.B.MyNotify) If an action is selected as Notify via class method, then this Notify Method is called if the alert is triggered and the Notify Method displays a list of its values.
Class	Name of the Metric class to monitor for alerts.
Properties	Property of the Monitor Class to be evaluated.
Expression	Expression to use to evaluate the Property. For example, “%1 = “User” && %2 < 100”. %1 refers to the first property in the list of properties, %2 the second property, etc.

2.7 Writing User-defined Monitor Classes

The Cache Monitor provides predefined system classes for monitoring system metrics. Users may also write their own Monitor Classes to monitor user application data and counters.

A Monitor Class is any class that inherits from the abstract Monitor class, %Monitor.Abstract. The %Monitor.System classes are examples of Monitor Classes. Users may also write their own Monitor Classes by following these steps:

1. 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 the user’s class definition. See the %Monitor.Adaptor class documentation in the *Caché Class Reference* for full details on this class, and the code that the user must write.
2. Compile your class. Compiling classes that inherit from %Monitor.Adaptor generates new classes in a subpackage of the users class called Sample. So, for instance, if a user is compiling A.B.MyMetric, a new class is generated in A.B.Sample.MyMetric. The user need do nothing with this class.

All Sample classes are automatically CSP-enabled, so that sample data for the users metric may be viewed by pointing to A.B.Sample.MyMetric.cls.

The Monitor automatically invokes this class and generates data and alerts if the class has been Activated. See Monitor Manager for class Activation.

2.7.1 Example of a User-defined Monitor Class

Our example gets the free space for each dataset in a Caché instance. At each sampling, we would want n instances of sample data objects, each instance corresponding to a dataset. In our 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 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:

```
Class MyMetric.Freespace Extends %Monitor.Adaptor [ ProcedureBlock ]
{
  /// Name of dataset
  Property DBName As %Monitor.String;

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

4. Add control properties as needed:

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

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

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

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

```
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 Q 0

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

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

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

```

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

/// Initialize routine metrics.
Method Initialize() As %Status
{
  s ..Rspec = ##class(%Library.ResultSet).%New("SYS.Database:FreeSpace")
  d ..Rspec.Execute("*",0)
  Quit $$$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.
{
  // Get freespace data
  Set stat = ..Rspec.Next(.sc)

  // Quit if we have done all the datasets
  If 'stat Q $$$Error($$$GeneralError,"End of Sample")

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

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

```

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

```

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

/// Open a tcp/ip device to send warnings
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 has created a new class, **Freespace**, in the package **MyMetric.Sample**:

```
/// Persistent sample class for MyMetric.Freespace
Class MyMetric.Sample.Freespace Extends %Monitor.Sample [ClassType=persistent]
{
    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 this class to write custom queries against your sample data.

2.8 Generating Metrics

The `%Monitor.SampleAgent` class is the class that does the actual sampling. It is this class that invokes 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 class **Initialize()** method for the class, then repeatedly invokes the **GetSample()** method for the class. On each call to **GetSample()**, the SampleAgent creates a sample class for the metrics class. The pseudo-code for these operations follows:

```
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 d sampler.Collect() h 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. */
```

2.9 Viewing Metrics Data

The simplest way to view metrics is using a Web browser; all metrics classes are CSP-enabled. This means that every sample is available for viewing through a browser: In our example, the CSP uri is:

```
http://localhost:8972/csp/user/MyMetrics.Sample.Freespace.cls
```

The output looks like:

```
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
```

The CSP code is generated automatically when the sample class is generated.

Alternatively, you use the **Display(<metrics class>)** method of the %Monitor.View class:

```
%SYS>s mclass="Monitor.Test.Freespace"  
  
%SYS>s col=##class(%Monitor.SampleAgent).%New(mclass)  
  
%SYS>w col.Collect()  
1  
%SYS>w ##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
```

2.10 Caché Monitor Errors

The following system errors are notified by the System Monitor unconditionally:

1. Process halt due to segment violation (access violation).
2. <FILEFULL>in database %
3. AUDIT: ERROR: FAILED to change audit database to '%.. Still auditing to '%.
4. AUDIT: ERROR: FAILED to set audit database to '%.
5. Sync failed during expansion of sfn #, new map not added

6. Sync failed during expansion of sfn #, not all blocks added
7. WD failed to allocate wdqlist...freezing system
8. WD: CP has exited - freezing system
9. Write Daemon encountered serious error - System Frozen
10. Insufficient global buffers - WD in panic mode
11. WD Panic: SFN x Block y written directly to database
12. Unexpected Write Error: dkvolblk returned %d for block #%%d in %
13. Unexpected Write Error: dkswrite returned %d for block #%%d in %
14. Unexpected Write Error: %d for block #%%d in %.
15. Cluster crash - All Cache systems are suspended
16. System is shutting down poorly, because there are open transactions, or ECP failed to preserve its state
17. SERIOUS JOURNALING ERROR: JRNSTOP cannot open %.* Stopping journaling as cleanly as possible, but you should assume that some journaling data has been lost.
18. Unable to allocate memory for journal translation table
19. Journal file has reached its maximum size of %u bytes and automatic rollover has failed
20. Write to journal file has failed
21. Failed to open the latest journal file
22. Sync of journal file failed
23. 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 (see ^SYSLOG) or switch journaling to a new device.
24. Error logging in journal
25. Journaling Error x reading attributes after expansion
26. ECP client daemon/connection is hung
27. Cluster Failsoft failed, couldn't determine locksysid for failed system - all cluster systems are suspended
28. enqpijstop failed, declaring a cluster crash
29. enqpijchange failed, declaring a cluster crash

30. Failure during WIJ processing - Declaring a crash
31. Failure during PIJ processing - Declaring a crash
32. Error reading block – recovery read error
33. Error writing block – recovery write error
34. 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
35. CP: Failed to create monitor for daemon termination
36. CP: WD has been on pass %d for %d seconds - freezing system. System will resume if WD completes a pass
37. WD: CP has died before we opened its handle - Freezing system
38. WD: Error code %d getting handle for CP monitor - CP not being monitored
39. WD: Control Process died with exit code %d - Freezing system
40. CP: Daemon died with exit code %d - Freezing system
41. Performing emergency Cache shutdown due to Operating System shutdown
42. CP: All processes have died - freezing system
43. cforce failed to terminate all processes
44. Failed to start mlock process (OpenVMS)
45. Failed to start slave write daemon
46. ENQDMN exiting due to reason #
47. Daemon %c found alive lock already owned by another process (OpenVMS)
48. Daemon %c failed to acquire alive lock. (OpenVMS)

3

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 System Management Portal to view the statistics reported by **^GLOSTAT**. Logon to the portal application for the system you are monitoring and navigate to the **[Home] > [System Usage]** page.

3.1 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:

```
Do ^GLOSTAT
```

2. The following prompt appears:

Should detailed statistics be displayed for each block type? No =>

As shown, the default is No. Press **Enter** to show summary block totals (Example A), or type Yes and press **Enter** to display detailed statistics by block type (Example B).

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), zero statistics (z), timed stats (# sec > 0), quit?

You may enter one of the following:

Response	Action
c	Displays the report again with updated cumulative statistics since the last initialization.
z	Initializes the ^GLOSTAT statistical counters to zero.
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 C and Example G).

4. To report statistics for a determinate interval (Example D):
 - a. Enter z to initialize the statistics to zeroes.
 - b. Either enter q to exit the utility, or remain at the prompt without entering anything.
 - c. Wait for desired period of time.
 - d. If you quit ^GLOSTAT, run it again. Otherwise, enter c to continue.
 - e. The ^GLOSTAT report displays statistics that reflect operations since you initialized the counters.

Note: The ^GLOSTAT utility uses 32-bit counters for most statistics; extremely large time intervals may cause some counters to overflow.

3.2 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 E).

The following table defines the ^GLOSTAT statistics.

Statistics Produced by ^GLOSTAT

Statistic	Definition
Global references (all)	Logical count of accesses to globals, including Sets , Kills , \$Data , \$Order , \$Increment , \$Query , and global references in expressions.
Global update references	Logical count of global references that are Sets , Kills , or \$Increments .
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).)
Cache Efficiency	Number of all global references divided by the number of physical block reads and writes. <i>Not a percentage.</i>
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.)

Statistic	Definition
Physical Block Reads	Number of physical database blocks (2-KB or 8-KB) read from disk for both global and routine references.
Physical Block Writes	Number of physical database blocks (2-KB or 8-KB) written to disk for both global and routine references.
Blocks Queued to be Written	Number of 2-KB or 8-KB database blocks that have been queued to be written to disk.

If the option is selected, ^GLOSTAT also reports the following detailed block statistics for the “Logical Block Requests,” “Physical Block Reads,” and “Blocks Queued to be Written” statistics. A total for all block types within each category appears in the right-most column (Example F).

Block types include:

Label	Description
Data	Data blocks.
Dir	Directory blocks.
Bdata	Big data blocks—blocks that contain big strings.
Map	Map blocks.
Upper ptr	Upper pointer blocks.
Bottom ptr	Bottom pointer blocks.
Other	Other blocks that are not any of the above. (For example, incremental backup or storage allocation information.)

3.3 Examples of ^GLOSTAT Output

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

- [Example A](#) — Initial running with no detailed statistics on a stand-alone or server configuration.
- [Example B](#) — Initial running with detailed block statistics on a stand-alone or server configuration.

- [Example C](#) — Subsequent running with no detailed statistics at a timed interval.
- [Example D](#) — Subsequent runnings with no detailed statistics showing the use of the `c`, `z`, and `q` responses.
- [Example E](#) — Initial running with no detailed statistics on a client configuration.
- [Example F](#) — Initial running with detailed block statistics on a client configuration.
- [Example G](#) — Initial and subsequent runnings with detailed block statistics at a timed interval.

3.3.1 Example A

The following is sample output of the initial running of the ^GLOSTAT routine. No detailed statistics are chosen and the Caché instance is either a stand-alone configuration or a server:

```
%SYS>Do ^GLOSTAT

Should detailed statistics be displayed for each block type? No =>n

Statistics                               Total
-----
Global references (all):                 6,445,330
Global update references:                1,322,207
Routine calls:                          896,625
Routine buffer loads and saves:         9,008
Logical block requests:                 2,997,805
Block reads:                            34,674
Block writes:                           1,918
Cache Efficiency:                       176
Journal Entries:                        211,164
Journal Block Writes:                   363

Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.2 Example B

The following is sample output of the initial running of the ^GLOSTAT routine. Detailed block statistics have been chosen and the Caché instance is either a stand-alone configuration or a server:

Gathering Global Activity Statistics Using ^GLOSTAT

```
%SYS>do ^GLOSTAT
```

```
Should detailed statistics be displayed for each block type? No =>y
```

```
Statistics
-----
Global references (all):                6,444,341
Global update references:              1,322,017
Routine calls:                          896,130
Routine buffer loads and saves:         9,008
Cache Efficiency:                       176
Journal Entries:                       211,164
Journal Block Writes:                   363
Logical Block Requests      Data:      1,643,739
  Dir:      13,147  Upper ptr:  138,837
  BData:    164,102 Bottom ptr: 1,034,472
  Map:      3,043   Other:      1
Physical Block Reads      Data:      33,846
  Dir:      42     Upper ptr:   31
  BData:    452   Bottom ptr:  290
  Map:      12     Other:      1
Physical Block Writes      Data:      1,952
Blocks Queued to be Written  Dir:      11     Upper ptr:   0
  BData:    317   Bottom ptr:  65
  Map:      22     Other:      6
-----
Total
-----
2,997,341
34,674
1,918
2,373
```

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.3 Example C

The following example shows ^GLOSTAT statistics per second for a 30-second timed interval. The user has not requested detailed statistics for each block type. The Caché instance is either a stand-alone configuration or a server:

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit? 30
```

```
Counts per Second for 30 Seconds...
```

```
Statistics (per second)
-----
Global references (all):                1,369.8
Global update references:              299.6
Routine calls:                          1,057.6
Routine buffer loads and saves:         12.0
Logical block requests:                 696.9
Block reads:                             0.7
Block writes:                             3.2
Cache Efficiency:                       354.3
Journal Entries:                       277.5
Journal Block Writes:                   0.5
```

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.4 Example D

The following example shows ^GLOSTAT results from running the routine multiple times.

After the initial output, entering `z` initializes the counters and then `q` quits the routine. After waiting the desired amount of time, the routine is run again and displays statistics reflecting operations since `z` was entered.

Entering `z` again initializes the statistics a second time, and entering `c` to continue immediately displays the initialized statistics. Entering `c` once more shows the statistics once again accumulating.

Gathering Global Activity Statistics Using ^GLOSTAT

%SYS>Do ^GLOSTAT

Should detailed statistics be displayed for each block type? No =>

Statistics	Total
Global references (all):	85,414
Global update references:	23,073
Routine calls:	67,092
Routine buffer loads and saves:	477
Logical block requests:	64,587
Block reads:	140
Block writes:	49
Cache Efficiency:	452
Journal Entries:	25,341
Journal Block Writes:	44

Continue (c), zero statistics (z), timed stats (# sec > 0), quit? z

Continue (c), zero statistics (z), timed stats (# sec > 0), quit? q

%SYS>Do ^GLOSTAT

Should detailed statistics be displayed for each block type? No =>

Statistics	Total
Global references (all):	4,332
Global update references:	568
Routine calls:	3,487
Routine buffer loads and saves:	95
Logical block requests:	1,423
Block reads:	0
Block writes:	130
Cache Efficiency:	33
Journal Entries:	218
Journal Block Writes:	1

Continue (c), zero statistics (z), timed stats (# sec > 0), quit? z

Continue (c), zero statistics (z), timed stats (# sec > 0), quit? c

Statistics	Total
Global references (all):	0
Global update references:	0
Routine calls:	0
Routine buffer loads and saves:	0
Logical block requests:	0
Block reads:	0
Block writes:	0
Cache Efficiency:	no i/o
Journal Entries:	0
Journal Block Writes:	0

Continue (c), zero statistics (z), timed stats (# sec > 0), quit? c

Statistics	Total
Global references (all):	9,362
Global update references:	1,742
Routine calls:	7,382
Routine buffer loads and saves:	96
Logical block requests:	4,404
Block reads:	4
Block writes:	0
Cache Efficiency:	2,341
Journal Entries:	1,548

```
Journal Block Writes:                3
Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.5 Example E

The following is sample output of the initial running of the ^GLOSTAT routine. No detailed statistics are chosen and the Caché instance is a client:

```
%SYS>DO ^GLOSTAT
Should detailed statistics be displayed for each block type? No =>n

Statistics                Local                Remote                Total
-----
Global references (all):    1,558,696                0                1,558,696
Global update references:    531,676                0                531,676
Routine calls:              95,987                0                95,987
Routine buffer loads and saves: 747                0                747
Logical block requests:    1,121,187                n/a                1,121,187
Block reads:                3,155                0                3,155
Block writes:                1,450                n/a                1,450
Cache Efficiency:            338                no gets
Journal Entries:            525,177                n/a                525,177
Journal Block Writes:        12,546                n/a                12,546

Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.6 Example F

The following is sample output of the initial running of the ^GLOSTAT routine. Detailed block statistics are chosen and the Caché instance is a client:

Gathering Global Activity Statistics Using ^GLOSTAT

```
%SYS>DO ^GLOSTAT
```

```
Should detailed statistics be displayed for each block type? No =>y
```

Statistics	Local	Remote	Total
Global references (all):	1,558,611	0	1,558,611
Global update references:	531,676	0	531,676
Routine calls:	95,979	0	95,979
Routine buffer loads and saves:	747	0	747
Cache Efficiency:	338	no gets	
Journal Entries:	525,177	n/a	525,177
Journal Block Writes:	12,546	n/a	12,546
Logical Block Requests	Data:	608,909	
Dir: 792	Upper ptr:	2,475	
BData: 81,750	Bottom ptr:	426,330	
Map: 879	Other:	1	1,121,136
Physical Block Reads	Data:	2,823	
Dir: 11	Upper ptr:	9	
BData: 240	Bottom ptr:	66	
Map: 5	Other:	1	3,155
Physical Block Writes			1,450
Blocks Queued to be Written	Data:	959	
Dir: 10	Upper ptr:	0	
BData: 430	Bottom ptr:	38	
Map: 17	Other:	7	1,461

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```

3.3.7 Example G

The following example shows ^GLOSTAT statistics per second for a 30-second timed interval after the initial running when the user requests detailed statistics for each block type. The Caché instance is either a stand-alone configuration or a server:

```
%SYS>Do ^GLOSTAT
```

```
Should detailed statistics be displayed for each block type? No =>y
```

Statistics				Total
-----				-----
Global references (all):				50,818
Global update references:				8,920
Routine calls:				25,279
Routine buffer loads and saves:				1,027
Cache Efficiency:				100
Journal Entries:				236
Journal Block Writes:				4
Logical Block Requests		Data:	16,375	
Dir:	1,760	Upper ptr:	263	
BData:	165	Bottom ptr:	2,538	
Map:	83	Other:	1	21,185
Physical Block Reads		Data:	153	
Dir:	6	Upper ptr:	5	
BData:	165	Bottom ptr:	38	
Map:	4	Other:	1	372
Physical Block Writes				134
Blocks Queued to be Written		Data:	104	
Dir:	6	Upper ptr:	0	
BData:	0	Bottom ptr:	29	
Map:	9	Other:	5	153

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit? 30
```

```
Counts per Second for 30 Seconds...
```

Statistics (per second)				Total
-----				-----
Global references (all):				4,232.7
Global update references:				1,453.4
Routine calls:				446.8
Routine buffer loads and saves:				14.1
Cache Efficiency:				507.9
Journal Entries:				33.3
Journal Block Writes:				0.1
Logical Block Requests		Data:	909.8	
Dir:	15.6	Upper ptr:	21.3	
BData:	0.2	Bottom ptr:	589.3	
Map:	1.3	Other:	0	1,537.5
Physical Block Reads		Data:	6.8	
Dir:	0	Upper ptr:	0	
BData:	0.2	Bottom ptr:	0.1	
Map:	0	Other:	0	7.1
Physical Block Writes				1.2
Blocks Queued to be Written		Data:	2.6	
Dir:	0.0	Upper ptr:	0	
BData:	0	Bottom ptr:	0.2	
Map:	0.1	Other:	0	2.9

```
Continue (c), zero statistics (z), timed stats (# sec > 0), quit?
```


4

Monitoring System Performance Using ^PERFMON

Caché provides a utility called ^**PERFMON**, which provides functions and a rudimentary menu to call these functions it uses to control the **MONITOR** facility.

Note: The **MONITOR** facility is an internal system feature that was accessible through the ^**MONITOR** routine in previous versions of Caché. The ^**PERFMON** functions were available in previous versions in the ^**VPMON** routine.

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, stops, 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 the following chapters in this guide:

- [“Using the Caché Monitor”](#)
- [“Examining Routine Performance Using ^%SYS.MONLBL”](#)

4.1 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. Clear Counters
6. Report Statistics
```

```
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](#)
- [Clear](#)
- [Report](#)

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.
```

4.1.1 Start

Turns on collection of the statistics.

Format:

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

Parameters:

- *process* — number of process slots to reserve (default = 10)
- *routine* — number of routine slots to reserve (default = 60)
- *global* — number of global slots to reserve (default = 25)
- *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

4.1.2 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

4.1.3 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

4.1.4 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

4.1.5 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

4.1.6 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
 - 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 (no pagination)
 - D – for comma delimited data which can be read into a spreadsheet
- *output* — enter a file name, 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.

Metrics for Custom Report

Metric Number	Column Title	Description
1	GloRef	Global references

Metric Number	Column Title	Description
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	RouBlkRd	Routine 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	RouBlkWt	routine 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	RouBlkBuf	routine 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

Metric Number	Column Title	Description
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
32	NCacheHit	network cache hits
33	NCacheMiss	network cache misses
34	NetLock	network locks
35	RtnLine	M commands
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.

4.2 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.

Monitoring System Performance Using ^PERFMON

```
%SYS>Do ^PERFMON
```

1. Start Monitor
2. Stop Monitor
3. Pause Monitor
4. Resume Monitor
5. Clear Counters
6. Report Statistics

```
Enter the number of your choice: 6
```

```
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, I=Incoming or O=Outgoing node
Sort ('P', 'R', 'G', 'I' or 'O'): g
Format may be: P=Print, D=Delimited data
Format ('P' or 'D'): 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. Clear Counters
6. Report Statistics

```
Enter the number of your choice: 6
```

```
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, I=Incoming or O=Outgoing node
Sort ('P', 'R', 'G', 'I' or 'O'): p
Format may be: P=Print, D=Delimited data
Format ('P' or 'D'): p
File name: perfmonC.txt
```

5

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 Caché 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 document contains the following sections:

- [Start Monitoring](#)
- [Monitoring Options](#)
- [Sample Output](#)
- [Programming Interface](#)

5.1 Start Monitoring

If the monitor is not running when you call `^%SYS.MONLBL`, the routine displays a warning message and gives you the one option to start the monitor. For example:

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

1.) Start Monitor

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 major 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 NOT taken from the memory already allocated by Cache, but a new, separate section of shared memory is created. This should be considered if you are using '*' wildcards and trying to analyze a large number of routines.

Enter '1' to select routines and enable the monitor:

Enter 1 to begin the dialog to provide the appropriate information to start monitoring.

You can select which routines and processes to monitor and which metrics to collect. You provide monitoring 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. You may use asterisk (*) wild cards to select multiple routines. Press **Enter** twice after entering the last routine name to end the list.

```
Enter routine names to monitor 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:
```

2. Select Metrics to monitor – Enter the number of your choice: 1 (the default) for default metrics, 2 for all available metrics, and 3 to choose a custom list of metrics.

```
Select Metrics to monitor  
1.) Monitor Default Metrics  
2.) Monitor All Metrics  
3.) Customize Monitor Metrics
```

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

The default metrics collected are:

- RtnLine — the number of times a line is executed

- `Time` — the clock time spent in executing that line
- `TotalTime` — the total clock time for that line including time spent in subroutines called by that line

If you choose to customize the list, 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> 3
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 most up to date list when you run the routine. See [Programming Interface](#) for a method of retrieving the list.

3. `Select Processes to monitor` – Enter the number of your choice: 1 (the default) for all processes, 2 for the current process, and 3 to choose a list of processes by process ID number (PID).

```
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 [\[Home\] > \[Processes\]](#) page of the System 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 routine name 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.

```
Monitor started.
```

```
Press RETURN to continue ...
```

After starting the line-by-line monitor you now have other [Monitoring Options](#) described in the following section.

5.2 Monitoring Options

You now have additional menu options:

- 2.) `Stop Monitor` — Stops all ^%SYS.MONLBL monitoring. Deallocates the counter memory and deletes collected data.
- 3.) `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 ^%SYS.MONLBL is running.
- 4.) `Resume Monitor` — Resumes collection after a pause. This option only appears if ^%SYS.MONLBL has been paused.
- 5.) `Clear Counters` — Clears any collected data, but continues monitoring and collecting new data.
- 6.) `Report Statistics` — Displays a report of collected data.

5.2.1 Report Statistics

When you choose to report the statistics of the metrics that have been collecting, you have other options to choose:

1. Presents a list of all monitored routines, and asks you to select one or more routines.
2. Press **Enter** to display all monitored routines.
3. After entering the last routine, press **Enter** again to end the list.

4. You can enter a file name for the output, or enter nothing and press **Enter** to display the report on your terminal.
5. 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).

5.3 Sample Output

The following is a partial sample of output from monitoring the `^%SYS.MONLBL` routine itself:

Routine ^%SYS.MONLBL ...

Line	RtnLine	Time	TotalTime	
.				
.				
79	0	0	0	read !,"FileName: ", filename s:...
80	0	0	0	open filename:"NW":1 if '\$t set ...
81	0	0	0	use filename
82	1	0.000010	0.000010	; Write Rtn Data
83	1	0.000005	0.000005	if (rtnnum > 0) {
84	0	0	0	do writertndata(rtnnum)
85	0	0	0	write !!
86	0	0	0	}
87	0	0	0	else {
88	1	0.000011	0.000011	for rtnnum=1:1:(\$zu(84,16)) {
89	163	0.001803	0.852597	do writertndata(rtnnum)
90	162	0.004931	0.004931	write !!
91	162	0.000428	0.000428	}
92	0	0	0	}
93	0	0	0	goto writertndataend
94	163	0.000337	0.000337	writertndata(rtnnum)
95	163	0.000333	0.000333	; Write Data For a Single Rtn
96	163	0.000512	0.000512	set rtnname = \$zu(84,16,2,rtnnum)
97	163	0.000334	0.000334	s sp=11
98	163	0.030024	0.030024	w !!,"Routine ^",rtnname," ..."
99	163	0.000601	0.000601	set lns = \$zu(84, 16, 1, rtnnum)
100	163	0.003862	0.003862	if lns=0 w " no data yet." quit
101	7	0.000015	0.000015	; Write Column Headers
102	7	0.000309	0.000309	write !!,"Line " s col=5
103	7	0.000039	0.000039	for metric=0:1:(\$zu(84,13)-1) {
104	21	0.000059	0.000059	s column = \$zu(84,13,11,metric) + 1
105	21	0.000146	0.000146	s out=\$piece(\$text(@("Flist+"_column)...
106	21	0.000571	0.000571	w ?col,\$j(out,sp-1)
107	21	0.000060	0.000060	s col=col+sp
108	21	0.000049	0.000049	}
109	7	0.000019	0.000019	for line=0:1:(lns-1) {
110	2958	0.103596	0.103596	write !,(line+1)
111	2959	0.007729	0.007729	s col=5
112	2960	0.009972	0.009972	for metric=0:1:(\$zu(84,13)-1) {
113	8880	0.029456	0.029460	s out=\$zu(84,16,3,line,metric)
114	8883	0.019621	0.019623	; Convert clock/CPU time to seconds
115	8886	0.021176	0.021178	s n=\$zu(84,13,11,metric)
116	8889	0.027933	0.027938	i (n=50)!(n=51) s out=\$select(out=0...
117	8892	0.306696	0.306726	write ?col,\$j(out,sp-1)
118	8895	0.024750	0.024753	s col=col+sp
119	8898	0.020944	0.020947	}
120	2967	0.169251	0.169251	write ?col,\$TEXT@("+" _ (line+1) _ ...
121	2968	0.015208	0.015208	}
122	6	0.000034	0.000034	quit

Total Time for Recursive Code

When a routine contains recursive code, the TotalTime counter for the line which calls back into the same subroutine may seem too large in relation to the total time of the entire routine. This effect is caused by the accumulation on one line, of TotalTime for multiple iterations of the same code. That is, the TotalTime includes the total amount of time spent on all *n* iterations of the subroutine, plus the total time spent on *n-1* iterations, plus the total time for *n-2*, etc. While this is not technically wrong, it may be confusing. Future enhancements to the routine may display a more useful number, the TotalTime for the outer most loop of the recursion.

5.4 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 Caché online Class Reference for `%Monitor.System.LineByLine` for more details.

A

Monitoring Caché Using BMC PATROL

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** Caché 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, `c:\CacheSys\Mgr` by default. 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 System Management Portal:

1. Navigate to the **[Home] > [Configuration] > [Monitor Settings]** page of the System 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 **[Home] > [Configuration] > [Monitor Settings]** page of the System 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.

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	System Management Portal Monitor Setting
display	Display mode. The literals <code>total</code> , <code>delta</code> , or <code>rate</code> 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, that you load into the PATROL Console.

- ISC_CACHE.km
- ISC_CACHE_CONFIG.km
- ISC_CACHE_DISK.km
- ISC_CACHE_GLOBAL.km
- ISC_CACHE_NETWORK.km
- ISC_CACHE_OTHER.km
- ISC_CACHE_OVERVIEW.km
- ISC_CACHE_ROUTINE.km

Once these KMs are loaded, the Console automatically attempts to discover Caché installations on all connected systems. The discovery process either searches the Registry on NT or parses the output from the **ccontrol list** command on UNIX and OpenVMS. 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

To add a Caché knowledge module into the Console and activate it:

1. From the PATROL Console **File** menu, click **Load KM**.
2. Select all the *.km files, located in the Caché Patrol directory, `c:\CacheSys\Patrol`, by default.
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 a configuration name and the Caché directory, `C:/CacheSys`, as the install 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:

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) (= $100 * (\text{LogicalReads} / (\text{LogicalReads} + \text{Physical Reads}))$))
	Licenses Used (gauge)

Category	Metric
Global	Global Refs
	Global Sets
	Global Kills
	Global Reads
	Blocks Allocated
	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
	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	

Category	Metric
Network	Net Global Refs
	Net Global Sets
	Net Global Kills
	Net Global Reads
	Net Requests Sent
	Net Cache Hits
	Net Cache Misses
	Net Locks
	Net Retransmits
	Net Buffer
	Net GblJobs
Other	Terminal Reads
	Terminal Writes
	Terminal Read Char
	Terminal Write Char
	Sequential Read
	Sequential Write

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](#)
- [Managing SNMP in Caché](#)
- [Caché as a Subagent](#)
- [Caché MIB Structure](#)

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 SNMP subdirectory of the Caché install directory.

B.2 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 AgentX protocol, and is active and listening for connections on the standard AgentX TCP port 705. On Windows systems, Caché 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.

Next, enable the Caché monitoring service using the following steps:

1. Navigate to the **[Home] > [Security Management] > [Services]** page of the System Management Portal.
2. Click the `%System_Monitor` service.
3. Select the **Service enabled** check box and click **Save**.
4. You return to the list of services page and see that the `%System_Monitor` service is enabled.

Finally, configure the Caché SNMP subagent to start automatically at Caché startup using the following steps:

1. Navigate to the **[Home] > [Configuration] > [Monitor Settings]** page of the System 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
w $$start^SNMP(port,timeout)"
  w !,"          port = TCP port for connection (default is 705)"
  w !,"          timeout = TCP port read timeout (default is "_20_" seconds)"
  w !!,"        w $$stop^SNMP()"
```

Caché logs any problems in establishing a connection or answering requests in the `SNMP.LOG` file in the Caché Manager's directory.

B.3 Caché as a Subagent

The SNMP client connects to the SNMP agent which is listening on a well-known address, UDP 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 (most notably OpenVMS and Tru64 UNIX). If the SNMP master agent supplied by an operating system is not AgentX-compatible, it can be replaced by the public domain NET-SNMP agent. 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 we supply 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.4 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 can be found in the `SNMP` subdirectory under the Caché manager's directory. Typically an SNMP management application must load the MIB file for the managed application to understand and appropriately display the information. This procedure varies between applications; consult the documentation for your management application for the appropriate way to load the Caché MIB.

The specific data defined in the Caché MIB is documented in the file itself, and is not repeated here. But 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.

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 IANA (Internet Assigned Numbers Authority) 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 sub-tree as follows:

- The level below *intersystems* is the product ID. For Caché this is `.1` (`iscCache`). This serves as the MIB module identity and is also known as the application ID.
- The next level separates data objects from notifications. 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 indexes.

- 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 indexes.
- The notifications (traps) are defined as individual entries at the same hierarchical level as the table above.

For example, the size of a database would be encoded 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.4.1 Extending the Caché MIB

Application programmers can add managed object definitions and extend the MIB for which the Caché subagent provides data:

1. Create Caché object definitions in classes that inherit from the %Monitor.Adaptor class. See the *Caché 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 an 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 *Caché 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 sub-tree, 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.indexes*. 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 sub-tree.

The following is an example of a user-defined %Monitor class which you can query via SNMP. The Application Monitor only includes properties with %Monitor data types in the SNMP data.

Before compiling this class in a user namespace, load the %Monitor.SampleAgent class which is required to store the data samples for SNMP. Run ^%MONAPPMGR, select option 2, "Manage Monitor Classes", and then option 3, "Register Monitor System Classes".

```
/// for SNMP. Run ^%MONAPPMGR, select option 2, "Manage Monitor Classes", and
/// then option 3, "Register Monitor System Classes".
///

/// When this class is compiled, it will create the SNMP.Sample.Example
/// class to store the sample data. You then use %MONAPPMGR to "activate"
/// the sample class, and start the Application Monitor to collect samples.
/// Set Sample Interval to an appropriate number of seconds for sampling data.
///
```

To create the SNMP MIB, run the **MonitorTools.SNMP:CreateMIB** method from the %SYS namespace. See the MonitorTools.SNMP class documentation for details.

input parameters for the method might be something like:

```
///
/// 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.

This creates the MC-MYAPP.MIB file in your default directory, which you can load into your SNMP management application. You may need to restart the SNMP master agent and the Caché ^SNMP service on your system before each recognizes this MIB.

```

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;
}

```

This 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
}
}

```


C

Monitoring Caché Using WMI

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).

This release of 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 [Microsoft WMI documentation](#) 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](#)

C.1 Configuring WMI in Caché

To prepare the Caché environment for collecting WMI information, complete the following two tasks:

- [Enable WMI and Monitoring](#)
- [Compile WMI Classes](#)

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 System Management Portal perform the following steps:

1. Navigate to the **[Home] > [Configuration] > [Monitor Settings]** page of the System Management Portal.
2. If the Monitor service is disabled:
 - a. Click **Edit** to navigate to the **[Home] > [Security Management] > [Services]** page.
 - b. Click the `%System_Monitor` service.
 - c. Select the **Service enabled** check box and click **Save** to return to the **Monitor Settings** page.
3. Select **Yes** for the **WMI Enabled** setting and click **Save**.

Compile WMI Classes

The `IscProc.mof` file defines the Caché WMI classes. The file is installed in the WMI subdirectory of the Caché manager's directory (`Cachesys\mgr`, by default).

This text file describes the Caché 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:\cachesys\mgr\WMI
C:\Cachesys\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:\Cachesys\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 the Caché 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 Caché 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 (Enabled-State=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, and the properties can be viewed by navigating the __SystemClass hierarchy using WMI CIM Studio. The WMI Events are the same events defined as SNMP Notifications, and the most current list can be found by inspecting the ISC-CACHE.MIB file in the SNMP directory of your Caché installation. The current events signaled by Caché include:

Event	Description
<i>cacheStart</i>	Caché instance startup
<i>cacheStop</i>	Caché instance shutdown
<i>cacheDBExpand</i>	Database expansion successful
<i>cacheDBOutOfSpace</i>	Database expansion close to limit
<i>cacheDBStatusChange</i>	Database read/write status change
<i>cacheDBWriteFail</i>	Database write failure
<i>cacheWDStop</i>	Write daemon stalling
<i>cacheWDPanic</i>	Write daemon entering <i>panic</i> mode
<i>cacheLockTableFull</i>	Lock table full (out of memory)
<i>cacheProcessFail</i>	Caché process access violation
<i>cacheECPTroubleDSrv</i>	ECP data server connection in <i>trouble</i> mode
<i>cacheECPTroubleASrv</i>	ECP application server connection in <i>trouble</i> mode
<i>cacheAuditLost</i>	System unable to record audit events
<i>cacheLoggedError</i>	Severe error being written to the console log
<i>cacheLicenseExceed</i>	License request exceeds available or allowed licenses

Event	Description
<i>cacheDCPNetError</i>	DCP client communications outage

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.