Monitoring Guide

Version 2023.1
2024-07-11

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

Monitoring Guide
InterSystems IRIS Data Platform Version 2023.1 2024-07-11

For Support questions about any InterSystems products, contact:

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

Table of Contents

List of Tables

1

Monitoring InterSystems IRIS Using the Management Portal

You can monitor many aspects of your InterSystems IRIS® data platform instance starting at the System Dashboard of the Management Portal. From the dashboard you can view performance indicators and then, for selected indicators, navigate to more detailed information.

For an overview of general InterSystems IRIS monitoring tools, see System Monitoring Tools.

Note: To access the **System Operation** tools described on this page, a user must be a member of a role with privileges to the **%Admin_Operate** resource. For more information, refer to this section of our guide to using the Management Portal.

1.1 Monitoring System Dashboard Indicators

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

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

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

Table 1–1: System Performance Indicators

Indicator	Definition
Globals/Second	Most recently measured number of global references per second.

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

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

Table 1–2: ECP Indicators

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

For more information on the ECP indicators, see Horizontally Scaling Systems for User Volume with InterSystems Distributed Caching.

Table 1–3: System Time Indicators

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

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

Table 1–4: System Usage Indicators

Indicator	Definition
Database Space	Indicates whether there is a reasonable amount of disk space available for database files. Clicking **... more details** displays the **System Operation** > **Databases** page.
Database Journal	Indicates the current status of the database journal. Clicking **... more details** displays the **System Operation** > **Journals** page.
Journal Space	Indicates whether there is a reasonable amount of disk space available for journal files. Clicking **... more details** displays the **System Operation** > **Journals** page.
Journal Entries	Number of entries written to the system journal. Clicking **... more details** displays the **System Operation** > **Journals** page.

Indicator	Definition
Lock Table	Current status of the system Lock Table. Clicking **... more details** displays the **System Operation** > **Locks** > **Manage Locks** page.
Write Daemon	Current status of the system Write daemon.
Transactions	Current status of open local and remote (ECP) transactions. If there are no open transactions, status is **Normal**; status may also be **Warning** (if the duration of the longest open local or remote transaction is greater than 10 minutes) and **Troubled** (if greater than 20 minutes). Clicking **... more details** displays the **Transactions** page (**System Operation** > **Transactions**).
Processes	Most recent number of running processes. Clicking **... more details** displays the **Processes** page (**System Operation** > **Processes**).
Web Sessions	Most recent number of web sessions. Clicking **... more details** displays the **Web Sessions** page (**System Operation** > **Web Sessions**).
Most Active Processes	Running processes with highest amount of activity (number of commands executed). Clicking **... more details** displays the **Processes** page (**System Operation** > **Processes**).

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

Table 1–5: Errors and Alerts Indicators

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

See Monitoring Log Files for more details.

Table 1–6: Licensing Indicators

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

Click the **... more details** link in the bottom details box to display the **System Operation** > **License Usage** page. For more information on licensing, see Managing InterSystems IRIS Licenses.

Table 1–7: Task Manager Upcoming Tasks

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

Click the **... more details** link in the bottom details box to display the **System Operation** > **Task Manager** > **Upcoming Tasks** page. For details on the Task Manager, see Using the Task Manager.

1.2 Monitoring System Usage and Performance

System performance metrics are described in the following tables:

- System Usage Table
- Shared Memory Heap Usage

1.2.1 System Usage Table

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

Table 1–8: System Usage Statistics

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

Statistic	Definition
Journal block writes	Number of 64-KB journal blocks written to the journal file.
Routine lines	Number of routine lines executed since system startup.
Last update	Date and time stamp of the displayed statistics.

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

1.2.2 Shared Memory Heap Usage

To view the InterSystems IRIS's shared memory heap (`gmheap`) usage, navigate to the **System Usage** page (**System Operation** > **System Usage**), and click the **Shared Memory Heap Usage** link.

Note: To learn how to change the size of the shared memory heap (`gmheap`), see `gmheap`.

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

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

Table 1–9: Shared Memory Heap Usage

Identifier	Definition
Miscellaneous	Shared memory allocated for the static memory table (SMT) and general string table (GST).
Audit System	Shared memory used for system auditing.
Classes Instantiated	Shared memory allocated/available/used for the class hash table and control blocks.
Database Encryption Key Change	Shared memory allocated/available/used for database encryption key changes.
Semaphore Objects	Shared memory allocated/available/used for semaphore objects.
Event System	Shared memory allocated/available/used for the event system.
Global Mapping	Shared memory allocated/available/used for global mapping and subscript-level mapping (SLM).
License Upgrade	Shared memory allocated/available/used for license upgrades.
Lock Table	Shared memory allocated/available/used for the lock system.
National Language Support	Shared memory allocated/available/used for National Language Support (NLS) tables.
Performance Monitor	Shared memory allocated/available/used for the Performance Monitor (**^PERFMON**).

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

1.2.3 Monitoring SQL Activity

To inspect the SQL statements currently running on your IRIS system, navigate to the **SQL Activity** page (**System Operation** > **SQL Activity**). This page provides a table with the following information about each active SQL statement:

- the **Process** ID associated with it
- the ID of the **User** executing it
- the **Namespace** containing the table or tables which the statement is querying
- the statement's **Type** (e.g. *DynamicQuery* for a Dynamic SQL query)
- the **Elapsed time** since the statement began its execution
- an excerpt from the text of the **Statement** itself.

Selecting any of the rows in this table reveals two further tables, which provide the following additional details about the corresponding SQL statement:

Table 1–10: Selected statement details

Row label	Value
Process	The ID of the process associated with the statement. This field links to the **Process Details** page for this process.
Transaction?	Whether or not the statement is active as part of an SQL transaction.
Start time	The time at which the statement began its execution.
Parameters	Where applicable, the first ten parameters that the statement is acting upon. For a Dynamic SQL query, this is a list of the literal values input as parameters into the query, replacing occurrences of the "?" character in the order listed. For commands such as **INSERT** or **UPDATE**, this is a list of values for the fields being inserted or updated for a record.
Statement	The full text of the statement. Where applicable, this field also includes a link to the **SQL Statement Details** page for this statement.
Cached Query	Where applicable, the name of the routine within which the statement is cached.

Table 1–11: Execution statistics

Row label	Value (overall and over the last week)
Times executed	The number of times the statement has been executed.
Average rowcount	The average number of rows the statement has returned upon each execution.
Average runtime	The average runtime for this statement.
Standard deviation	A measure of the degree of variation in runtimes for executions of the statement over the given interval.

1.3 Monitoring Locks

InterSystems IRIS locks are created when an InterSystems IRIS process issues a LOCK command on an ObjectScript local variable or global variable, as long as the entity is not already locked by another process. Entities need not exist in the database to lock them. See Locking and Concurrency Control.

To display locks system-wide, navigate to the **View Locks** page (**System Operation** > **Locks** > **View Locks**). To delete selected locks system-wide, navigate to the **Manage Locks** page (**System Operation** > **Locks** > **Manage Locks**). In both cases, the displayed lock table lists one row for each held lock and for each waiting lock request, identifying the owner. A single row may identify multiple locks held by an owner on the same entity. For example, holding an incremented lock or holding both a Shared lock and an Exclusive lock. If more than one process holds a lock on the same entity, each owner has its own row.

Optionally select either or both of the following check boxes at the top of this page, to include additional information if needed. This information is not included by default, for performance reasons:

- **Owner's routine information**. This option adds the **Routine** column to the display.
- **SQL table name**. This option adds the **SQL table name** column to the display.

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

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

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

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

You may need to enlarge the size of the lock table if your system uses a large number of locks. You can do this using the Management Portal; for instructions, see locksiz.

For more detailed information and alternative ways to manage locks, see Lock Management.

1.4 Monitoring InterSystems IRIS Logs

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

- Several log files are available in the install-dir\mgr directory; two can be viewed using the Management Portal.
- You can view the application error log or driver error log using the Management Portal
- The contents of the InterSystems IRIS system error log, or syslog, can be reviewed using one of several methods.

You can also enable *structured logging*, which will write the same messages seen in messages.log to a machine-readable file that can be ingested by your choice of monitoring tool. See Setting Up Structured Logging.

1.4.1 Log Files in the install-dir\mgr Directory

The following log files are available in the *install-dir*\mgr directory. They are saved as plain text files and can be viewed using any text editor or viewer. The messages log and System Monitor log can be viewed using the Management Portal.

alerts log

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

initialization log

The initialization log, iboot.log, contains information about the initialization of the InterSystems IRIS instance.

journal history log

The journal history log, journal.log, contains a list of all journal files maintained by the InterSystems IRIS instance and is used by all journal-related functions, utilities, and APIs to locate journal files. See Journaling.

messages log

InterSystems IRIS reports a variety of messages to the messages log file (messages.log), including: general messages; startup/shutdown, license, and network errors; certain operating system errors; and the success or failure of jobs started remotely from other systems. System Monitor also writes notifications to the messages log. The directory for messages.log can be configured (see console), but the default location is *install-dir*\mgr.

On Windows-based platforms, all console messages are sent to the messages log file, messages.log. On UNIX®/Linux platforms, you can configure console messages to be sent to the messages log file, the console terminal, or both.

The size of the messages.log file is monitored by System Monitor. The file grows until it reaches the configured maximum size, at which point InterSystems IRIS saves the file and starts a new one. See MaxConsoleLogSize for information about configuring the maximum messages log size.

You can view the messages log from the **View Messages Log** page of the Management Portal (**System Operation** > **System Logs** > **Messages Log**). If the messages log is larger than 1 MB, only the most recent 1 MB portion is displayed by the Management Portal. Click the **Show entire file** link to display the entire file, which may require some time if the file is very large.

Note: If you have trouble starting InterSystems IRIS, use any text editor or text viewer to view the messages log.

System Monitor log

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

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

You can view the messages log from the **System Monitor Log** page of the Management Portal (**System Operation** > **System Logs** > **System Monitor Log**). If the System Monitor log is larger than 1 MB, only the most recent 1 MB portion is displayed by the Management Portal. Click the **Show entire file** link to display the entire file, which may require some time if the file is very large.

1.4.2 Application and Database Driver Error Logs

The **View Application Error Log** page (**System Operation** > **System Logs** > **Application Error Log**) allows you to view application errors.

Likewise, the **xDBC Error Log** page (**System Operation** > **System Logs** > **xDBC Error Log**) allows you to view database driver errors.

1.4.3 InterSystems IRIS System Error Log

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

By default, the system error log contains the 500 most recent log items. For information about configuring the number of items in the system error log, see errlog.

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

- Open the Terminal, enter **set $namespace = "%SYS"** to switch to the %SYS namespace, and enter **do ^SYSLOG**. You can also enter **do FILTER^SYSLOG**, which has options to limit the output based on specific error codes or process ID.
- Run a diagnostic report, as described in Using the Diagnostic Report.
- Run the **irisstat** command with the **–e1** option, as described in Running irisstat with Options.
- Run the **IRISHung** script, as described in IRISHung Script.

You can configure InterSystems IRIS to write the system error log to the messages log during shutdown using the **ShutDownLogErrors** setting (see ShutDownLogErrors).

2

Using the InterSystems Diagnostic Report

InterSystems provides a mechanism to run a Diagnostic Report on your InterSystems IRIS® data platform instance. The Diagnostic Report is a snapshot of information about an instance that should be run and sent to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems. For more details on the type of information collected, see the Diagnostic Report Contents section.

This topic describes how to configure and run the Diagnostic Report as a task from the Management Portal. For more information on this task, see the %SYS.Task.DiagnosticReport entry in the *InterSystems Class Reference*.

2.1 Running the Diagnostic Report Task

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

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

For example, on September 24, 2023 at 8:46 p.m., running the Diagnostic Report task with a license key issued to MyCompany on an instance installed in C:\MyInstallDir generates the following report file:

```
C:\MyInstallDir\mgr\MyCompany201909242046.html
```

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

Viewing the Diagnostic Report Task History

Click **Task History** at the top of the **Diagnostic Report** page to display the history for the Diagnostic Report task. (See Using the Task Manager for information about tasks and task history.)

2.2 Configuring Diagnostic Report Settings

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

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

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

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

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

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

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

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

  Important: The report includes private application information. InterSystems keeps all data strictly confidential.

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

Additional information for the WRC:

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

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

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

2.3 Diagnostic Report Contents

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

- Basic Information
- Advanced Information

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

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

2.3.1 Basic Information

The basic information includes the following categories:

General

Displays the following information:

- Full host name (with domain)
- IP address
- User name
- Date and time report was created
- InterSystems IRIS version string (*$ZVersion*)
- InterSystems IRIS objects version string
- InterSystems ODBC/JDBC server version information
- National Language Support (NLS) information
- Free block count information

- Operating system version (**uname -a** on UNIX® systems)
- Extended system information (only on Windows systems if the PsInfo.Exe utility is in the InterSystems IRIS Bin directory).

Key File

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

License Counts

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

%SS

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

Operating System Processes List

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

Spin Counts

Displays spin count information.

CPF File

Displays the contents of the active InterSystems IRIS configuration file (iris.cpf).

SysLog

Displays the contents of the InterSystems IRIS system error log; see InterSystems IRIS System Error Log for more information.

Security

Displays a listing of the following security information:

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

Audit

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

messages

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

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

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

2.3.2 Advanced Information

The advanced information includes the following categories:

irisstat Snapshot #1

Displays output of the InterSystems statistics utility (**irisstat**) run with the following options:

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

For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility.

irisstat Snapshot #2

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

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

```
File /iris/iristestsys/mgr/irisstat201103151102.html is too big to be appended to
the Log File. A copy has been left in the Directory.
```

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

Network Status

Displays network information — output of the following utilities:

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

Dump License

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

Dump Files in Manager's Directory

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

GloStat

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

3

Using Log Monitor

Log Monitor monitors the InterSystems IRIS® data platform instance's messages log for errors and traps reported by InterSystems IRIS daemons and user processes; and generates corresponding notifications, including email if configured. You can manage Log Monitor using the **^MONMGR** utility.

3.1 System Monitoring Tools

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

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

3.2 Log Monitor Overview

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

The messages log contains useful information about the InterSystems IRIS instance, ranging from general messages to errors and traps to System Monitor alerts and warnings. By generating notifications based on messages log contents, Log Monitor raises the visibility of alerts for system operators.

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

Log Monitor operates with the following settings by default:

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

  Note: Log Monitor creates the alerts log the first time it generates a notification. The alerts log, or alerts.log, is located in the <*install-dir*>/mgr directory.

You can configure Log Monitor using the interactive ^**MONMGR** utility, described in the following section.

3.3 Configuring the Log Monitor

The Log Monitor Manager utility, ^**MONMGR**, allows you to configure and manage Log Manager. You can stop and start Log Monitor, change the default settings, and configure email notifications.

To start the Log Monitor Manager:

1. Enter the following command in the Terminal. ^**MONMGR** must be executed in the %SYS namespace.

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

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

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

   Option?
   ```

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

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

3.3.1 Start/Stop/Update Monitor

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

```
Option? 1

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

Option?
```

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

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

3.3.2 Manage Monitor Options

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

```
Option? 2

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

Option?
```

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

Option	Description
`1) Set Monitor Interval`	Lets you change the interval at which the messages log is scanned. InterSystems recommends an interval no longer than the default of 10 seconds.
`2) Set Alert Level`	Lets you set the severity level of messages log entries generating notifications, as follows: • 1 – warning, severe and fatal • 2 – severe and fatal • 3 – fatal only
`3) Manage Email Options`	Lets you configure Log Monitor email notifications using the Manage Email Options submenu.

Option	Description
`4) Exit`	Returns to the main menu.

Note: Because Log Monitor generates a notification only for the first in a series of messages log entries from a given process within about an hour, setting the alert level to 1 could mean that when a warning has generated an alerts log entry or email message, a subsequent severity 2 alert from the same process does not generate a notification.

3.3.3 Manage Email Options

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

```
Option? 3

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

Option?
```

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

Option	Description
`1) Enable / Disable Email`	Enabling email causes Log Monitor to: • send an email notification for each item currently in the alerts log, if any • delete the alerts.log file (if it exists) • send email notifications for messages log entry of the configured severity from that point forward Disabling email causes Log Monitor to write entries to the alerts log. **Note:** Enabling/disabling email does not affect other email settings; that is, it is not necessary to reconfigure email options when you enable/disable email.
`2) Set Sender`	Select this option to enter text that indicating the sender of the email, for example `Log Monitor`. The text you enter does not have to represent a valid email account. You can set this field to NULL by entering `-` (dash).
`3) Set Server`	Select this menu item to enter the name and port number (default 25) of the email server that handles email for your site. Consult your IT staff to obtain this information. You can set this field to NULL by entering `-` (dash).
`4) Manage Recipients`	This option displays a submenu that lets you list, add, or remove the email addresses to which each notification is sent: **Note:** Each valid email address must be added individually; when you select `2) Add Recipient`, do not enter more than one address when responding to the `Email Address?` prompt.

Option	Description
`5) Set Authentication`	Lets you specify the authentication username and password if required by your email server. Consult your IT staff to obtain this information. If you do not provide entries, the authentication username and password are set to NULL. You can set the **User** field to NULL by entering - (dash).
`6) Test Email`	Sends a test message to the specified recipients using the specified email server.
`7) Exit`	Returns to the Manage Monitor Options submenu.

3.4 Log Monitor Errors and Traps

The following messages log errors always generate Log Monitor notifications:

- Process halt due to segment violation (access violation).
- <FILEFULL>in database %
- AUDIT: ERROR: FAILED to change audit database to '%. Still auditing to '%.
- AUDIT: ERROR: FAILED to set audit database to '%.
- Sync failed during expansion of sfn #, new map not added
- Sync failed during expansion of sfn #, not all blocks added
- WRTDMN failed to allocate wdqlist...freezing system
- WRTDMN: CP has exited - freezing system
- write daemon encountered serious error - System Frozen
- Insufficient global buffers - WRTDMN in panic mode
- WRTDMN Panic: SFN x Block y written directly to database
- Unexpected Write Error: dkvolblk returned %d for block #%d in %
- Unexpected Write Error: dkswrite returned %d for block #%d in %
- Unexpected Write Error: %d for block #%d in %.
- Cluster crash - All Cache systems are suspended
- System is shutting down poorly, because there are open transactions, or ECP failed to preserve its state
- SERIOUS JOURNALING ERROR: JRNSTOP cannot open %.* Stopping journaling as cleanly as possible, but you should assume that some journaling data has been lost.
- Unable to allocate memory for journal translation table
- Journal file has reached its maximum size of %u bytes and automatic rollover has failed
- Write to journal file has failed
- Failed to open the latest journal file
- Sync of journal file failed
- Journaling will be disabled in %d seconds OR when journal buffers are completely filled, whichever comes first. To avoid potential loss of journal data, resolve the cause of the error (consult the InterSystems IRIS system error log, as described in InterSystems IRIS System Error Log) or switch journaling to a new device.

- Error logging in journal
- Journaling Error x reading attributes after expansion
- ECP client daemon/connection is hung
- Cluster Failsoft failed, couldn't determine locksysid for failed system - all cluster systems are suspended
- enqpijstop failed, declaring a cluster crash
- enqpijchange failed, declaring a cluster crash
- Failure during WIJ processing - Declaring a crash
- Failure during PIJ processing - Declaring a crash
- Error reading block – recovery read error
- Error writing block – recovery write error
- WIJ expansion failure: System Frozen - The system has been frozen because WIJ expansion has failed for too long. If space is created for the WIJ, the system will resume otherwise you need to shut it down with irisforce
- CP: Failed to create monitor for daemon termination
- CP: WRTDMN has been on pass %d for %d seconds - freezing system. System will resume if WRTDMN completes a pass
- WRTDMN: CP has died before we opened its handle - Freezing system
- WRTDMN: Error code %d getting handle for CP monitor - CP not being monitored
- WRTDMN: Control Process died with exit code %d - Freezing system
- CP: Daemon died with exit code %d - Freezing system
- Performing emergency Cache shutdown due to Operating System shutdown
- CP: All processes have died - freezing system
- irisforce failed to terminate all processes
- Failed to start auxiliary write daemon
- ENQDMN exiting due to reason #
- Becoming primary mirror server

4

Introduction to System Monitor Tools

System Monitor is a flexible, user-extensible utility used to monitor an InterSystems IRIS® data platform instance and generate notifications when the values of one or more of a wide range of metrics indicate a potential problem.

4.1 About System Monitor

System Monitor tools include the following monitoring tools:

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

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

4.2 See Also

- Introduction to System Monitor
- Using Core System Monitor
- Using Health Monitor
- Application Monitor
- System Monitoring Tools
- Manage Email Options (information about generating email messages from notifications in the messages log, including those generated by System Monitor)
- Monitoring Log Files (includes information on the log files generated by this tool)

5

Using System Monitor

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

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

System Monitor is part of the System Monitor tools.

5.1 The System Monitor Process

In each namespace in which it is configured to run, System Monitor gathers and delivers system metric information in three stages using three types of classes (or System Monitor *components*). *Sensor classes* collect information, *subscriber classes* evaluate the information to form notifications, and *notifier classes* post the notifications to the proper alerting systems. The following describes the sequence in greater depth:

1. Obtain metric information

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

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

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

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

2. Evaluate metric information

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

method, the subscriber class evaluates the value and if appropriate returns a notification containing text and a severity code.

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

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

3. Generate notifications

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

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

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

5.2 Tracking System Monitor Notifications

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

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

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

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

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

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

5.3 System Monitor Status and Resource Metrics

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

Table 5–1: System Monitor Status and Resource Notifications

Metric	Description	Notification Rules

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

Metric	Description	Notification Rules
ISCAgent (mirror members only)	ISCAgent status	• Unresponsive for <1 minute — warning • Unresponsive for >1 minute — alert

5.4 System Monitor Health State

Based on notifications posted to the messages log (see Monitoring Log Files), including both system alerts generated directly by the InterSystems IRIS instance and alerts and warnings generated by System Monitor and its Health Monitor component, System Monitor maintains a single value summarizing overall system health in a register in shared memory.

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

Table 5–2: System Monitor Health State

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

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

- The journal daemon is paused for more than 5 seconds or frozen (see Journal I/O Errors).
- Any of switches 10, 11, 13, or 14 are set (see Using Switches).
- The write daemon is stopped for any reason or sets the **updates locked** flag for more than 3 seconds.
- The number of available global buffers (in the database cache) falls into the critical region and remains there for more than 5 seconds.

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

You can view the System Monitor health state using:

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

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

5.5 System Monitor Defaults

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

5.5.1 Default System Monitor Components

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

Sensor classes:

- SYS.Monitor.SystemSensors

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

- **%SYS.Monitor.AppMonSensor**

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

Subscriber classes:

- SYS.Monitor.Health.Control

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

- SYS.Monitor.SystemSubscriber

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

Notifier class:

- SYS.Monitor.SystemNotify

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

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

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

5.5.2 Default System Monitor Namespace

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

Health Monitor runs only in the %SYS namespace.

5.5.3 Default System Monitor Settings

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

By default, the System Monitor:

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

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

5.6 Using the ^%SYSMONMGR Utility

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

Important: All manual operations using the **^%SYSMONMGR** utility described in this section can be executed programatically using the methods in the %Monitor.Manager API.

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

```
do ^%SYSMONMGR
```

The main menu appears.

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

Option?
```

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

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

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

5.6.1 Start/Stop System Monitor

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

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

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

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

Note: System Monitor monitors the size of the messages log and rolls it over when required. When System Monitor is stopped, the messages log may exceed the limit set by the MaxConsoleLogSize configuration setting until the instance is restarted or the `PurgeErrorsAndLogs` task is run. See Monitoring Log Filesfor information about the messages log.

5.6.2 Set System Monitor Options

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

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

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

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

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

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

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

5.6.3 Configure System Monitor Components

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

5.6.3.1 Configure System Monitor Classes

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

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

Enter 1 to display the options for configuring classes:

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

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

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

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

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

5.6.3.2 Configure System Monitor Namespaces

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

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

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

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

Enter 2 to display the options for configuring namespaces:

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

Enter 1 to list the currently configured startup namespaces.

Enter 2 to add a startup namespace.

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

5.6.4 View System Monitor State

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

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

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

5.6.5 Manage Application Monitor

See Using ^%SYSMONMGR to Manage Application Monitor.

5.6.6 Manage Health Monitor

See Using ^%SYSMONMGR to Manage Health Monitor.

5.6.7 View System Data

Enter 7 at the main menu (or 6 in namespaces other than **%SYS**) to display options for viewing System Monitor information about the system.

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

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

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

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

5.7 Defining System Monitor Components

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

5.7.1 Sensor Classes

Sensor classes extend %SYS.Monitor.AbstractSensor. The System Monitor controller initially calls each sensor class's **Start()** method; thereafter, on each cycle, it calls the **GetSensors()** method. The **SetSensor()** method is used within the

sensor class to set sensor name/value pairs in the SensorReading property, which is returned by **GetSensors()** and passed to all subscriber classes.

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

5.7.2 Subscriber Classes

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

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

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

5.7.3 Notifier Classes

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

5.8 See Also

- Introduction to System Monitor
- Using Core System Monitor
- Using Health Monitor
- Application Monitor
- System Monitoring Tools
- Manage Email Options (information about generating email messages from notifications in the messages log, including those generated by System Monitor)
- Monitoring Log Files (includes information on the log files generated by this tool)

6

Using Health Monitor

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

Health Monitor is part of the System Monitor tools.

6.1 Health Monitor Overview

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

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

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

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

The following subsections describe how Health Monitor works and contain information about configuring and extending it in various ways:

- Health Monitor Process Description
- Sensors and Sensor Objects
- Periods
- Charts
- Notification Rules

6.1.1 Health Monitor Process Description

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

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

- Health Monitor monitors a number of system sensors, which are represented as *sensor objects*. Every sensor object has a base (minimum) value for sensor samples, and optionally includes two notification threshold values (one for alerts, and the other for warnings) which can be set as absolute values or multipliers. These values determine when Health Monitor sends notifications.

  Sensors and Sensor Objects lists all the sensor objects.

- For the duration of a predefined *period*, each sensor is sampled every 30 seconds; samples below the base value are discarded. By default there are 63 weekly periods (nine per day), but you can configure your own weekly, monthly, quarterly, or yearly periods. Periods lists the default periods.

- For a given sensor, unless the notification thresholds are set as absolute values, Health Monitor evaluates the sensor readings based on a *chart*. If the necessary chart for the current period does not exist, Health Monitor places the sensor in *analysis mode* to generate the chart.

  You can edit or create a chart to calibrate how Health Monitor evaluates sensor readings. For more information, see Charts.

- If a sensor is not in analysis mode, it is in *monitoring mode*. In monitoring mode, sensor readings are evaluated by the appropriate subscriber class. To ensure that notifications are not triggered by transient abnormal samples, every six sample values are averaged together to generate one reading every three minutes, and it is these readings that are evaluated.

- When a sequence of readings meets the criteria for a notification (as described in Notification Rules), the subscriber class generates an alert or a warning by passing a notification containing text and a severity code to the system notifier, SYS.Monitor.SystemNotify.

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

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

6.1.2 Sensors and Sensor Objects

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

Some sensors represent an overall metric for the InterSystems IRIS instance. These are the sensors which, in the following table, have no value listed in the **Sensor Item** column. For example, the **LicensePercentUsed** sensor samples the percentage of the instance's authorized license units that are currently in use, while the **JournalGrowthRate** sensor samples the amount of data (in KB per minute) written to the instance's journal files.

Other sensors collect information about a specific sensor item (either a CSP server, a database, or a mirror). For example, **DBReads** sensors sample the number of reads per minute from each mounted database. These sensors are specified as *<sensor_object> <sensor_item>*; for example, the **DBLatency** *install-dir***\IRIS\mgr\user** sensor samples the time (in milliseconds) required to complete a random read on the **USER** database.

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

Table 6–1: Health Monitor Sensor Objects

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

Sensor Object	Sensor Item	Description	Base	Max Val.	Max Mult.	Warn Val.	Warn Mult.
DBWrites	*database_directory*	Writes per minute to the listed mounted database.	1024	—	2	—	1.6
DiskPercentFull	*database_directory*	Disk percentage used for the listed mounted database.	50	99	—	95	—
ECPAppServerKBPerMinute		KB per minute sent to the ECP data server.	1024	—	2	—	1.6
ECPConnections		Number of active ECP connections.	100	—	2	—	1.6
ECPDataServerKBPerMinute		KB per minute received as ECP data server.	1024	—	2	—	1.6
ECPLatency		Network latency (milliseconds) between the ECP data server and this ECP application server.	1000	3000	—	3000	—
ECPTransOpenCount		Number of open ECP transactions	100	—	2	—	1.6
ECPTransOpenSecsMax		Duration (seconds) of longest currently open ECP transaction	60	—	2	—	1.6
GlobalRefsPerMin		Global references per minute.	1024	—	2	—	1.6
GlobalSetKillPerMin		Global sets/kills per minute.	1024	—	2	—	1.6
JournalEntriesPerMin		Number of journal entries written per minute.	1024	—	2	—	1.6
JournalGrowthRate		Number of KB per minute written to journal files.	1024	—	2	—	1.6
LicensePercentUsed		Percentage of authorized license units currently in use.	50	—	1.5	—	—
LicenseUsedRate		License acquisitions per minute.	20	—	1.5	—	—
LockTablePercentFull		Percentage of the lock table in use.	50	99	—	85	—
LogicalBlockRequestsPerMin		Number of logical block requests per minute.	1024	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max Val.	Max Mult.	Warn Val.	Warn Mult.
MirrorDatabaseLatencyBytes	*mirror_name*	On the backup failover member of a mirror, number of bytes of journal data received from the primary but not yet applied to mirrored databases on the backup (measure of how far behind the backup's databases are).	$2*10^7$	—	2	—	1.6
MirrorDatabaseLatencyFiles	*mirror_name*	On the backup failover member of a mirror, number of journal files received from the primary but not yet fully applied to mirrored databases on the backup (measure of how far behind the backup's databases are).	3	—	2	—	1.6
MirrorDatabaseLatencyTime	*mirror_name*	On the backup failover member of a mirror, time (in milliseconds) between when the last journal file was received from the primary and when it was fully applied to the mirrored databases on the backup (measure of how far behind the backup's databases are).	1000	4000	—	3000	—
MirrorJournalLatencyBytes	*mirror_name*	On the backup failover member of a mirror, number of bytes of journal data received from the primary but not yet written to the journal directory on the backup (measure of how far behind the backup is).	$2*10^7$	—	2	—	1.6
MirrorJournalLatencyFiles	*mirror_name*	On the backup failover member of a mirror, number of journal files received from the primary but not yet written to the journal directory on the backup (measure of how far behind the backup is).	3	—	2	—	1.6

Sensor Object	Sensor Item	Description	Base	Max Val.	Max Mult.	Warn Val.	Warn Mult.
MirrorJournalLatencyTime	*mirror_name*	On the backup failover member of a mirror, time (in milliseconds) between when the last journal file was received from the primary and when it was fully written to the journal directory on the backup (measure of how far behind the backup is).	1000	4000	—	3000	—
PhysicalBlockReadsPerMin		Number of physical block reads per minute.	1024	—	2	—	1.6
PhysicalBlockWritesPerMin		Number of physical block writes per minute.	1024	—	2	—	1.6
ProcessCount		Number of active processes for the InterSystems IRIS instance.	100	—	2	—	1.6
RoutineCommandsPerMin		Number of routine commands per minute.	1024	—	2	—	1.6
RoutineLoadsPerMin		Number of routine loads per minute.	1024	—	2	—	1.6
RoutineRefsPerMin		Number of routine references per minute.	1024	—	2	—	1.6
SMHPercentFull		Percentage of the shared memory heap (generic memory heap) in use.	50	98	—	85	—
TransOpenCount		Number of open local transactions (local and remote).	100	—	2	—	1.6
TransOpenSecondsMax		Duration (seconds) of longest currently open local transaction.	60	—	2	—	1.6
WDBuffers		Average number of database buffers updated per write daemon cycle.	1024	—	2	—	1.6
WDCycleTime		Average number of seconds required to complete a write daemon cycle.	60	—	2	—	1.6
WDWIJTime		Average number of seconds spent updating the write image journal (WIJ) per cycle.	60	—	2	—	1.6

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

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

When you are monitoring a mirror member (see Mirroring), the following special conditions apply to Health Monitor:

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

6.1.3 Periods

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

Table 6–2: Default Health Monitor Periods

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

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

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

Descriptions are optional for user-defined periods.

6.1.4 Charts

If the notification threshold values for a sensor object are not given as multipliers (or not specified), Health Monitor requires a chart to evaluate those sensor readings. Health Monitor generates the necessary charts by calculating the mean, standard deviation, and maximum value from sample sensor readings. This section describes how Health Monitor generates charts in analysis mode, and how to edit or create custom charts.

6.1.4.1 Analysis Mode

Before Health Monitor can evaluate sensor samples, it checks whether that sensor requires a chart. If a chart is required but does not yet exist, Health Monitor automatically puts the sensor in analysis mode.

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

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

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

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

6.1.4.2 Editing a Chart

The **^%SYSMONMGR** utility lets you display a list of all current charts, including the mean and sigma of each. You can also display the details of a particular chart, including the individual readings and highest reading. To access these options from the utility, select `Configure Charts` from the Configure Health Monitor Classes submenu .

The `Configure Charts` option also provides two ways to customize alerting by customizing charts:

- You can change the mean and/or sigma to whatever values you wish by editing an existing chart. The standard notification rules apply, but using the values you have entered.
- You can create a chart, specifying an alert value and a warning value. Creating a chart is similar to setting an absolute value for the notification threshold; alerts and warnings are generated based solely on the values you supply for the chart.

Note: When listing, examining, editing, or creating charts, the **Item** heading or prompt refers to a database (specified by a directory path), a Web Gateway server (specified by an IP address), or a mirror (specified by the mirror name). See Sensors and Sensor Objects for more information.

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

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

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

Note: A chart generated by Health Monitor, including one you have edited, can be automatically recalibrated as described in Analysis Mode. In addition, all charts generated by Health Monitor, including those that have been edited, are deleted when an InterSystems IRIS instance is upgraded.

A chart *created* using the **Configure Charts** submenu or the **CreateChart()** class method, however, is never automatically recalibrated or deleted on upgrade. A user-created chart is therefore permanently associated with the selected sensor/period combination until you select the **Reset Charts** option within the **Reset Defaults** option of the Configure Health Monitor Classes submenu or select **Recalibrate Charts** within the **Configure Charts** option.

6.1.5 Notification Rules

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

Note also that:

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

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

6.1.6 Examples

In this example, the chart for the **DBReads** *install-dir***IRIS\mgr\user** sensor during the Monday 09:00 a.m. - 11:30 a.m. period indicates that the mean reads per minute from the **USER** database is 2145, with a sigma of 141 and maximum value of 2327. The default notification threshold multipler for **DBReads** is 2. An alert is generated for this sensor when three consecutive readings exceed the greater of the following two values:

- maximum multiplier * (chart mean + (3 * chart sigma))

  2 * (**2145** + (3 * **141**)) = 5136

- maximum multiplier * (chart maximum value + chart sigma))

  2 * (**2327** + **141**) = 4936

So, or this sensor during this period, an alert is generated if three consecutive readings are greater than 5136.

A sensor with no multipliers or maximum values is evaluated with a multiplier of 1. As an example, if the **DBReads** sensor object were edited to remove the multipliers, leaving it with only a base, an alert is generated for **DBReads** *install-dir***IRIS\mgr\user** when three consecutive readings are greater than 2568, calculated as the greater of:

- maximum multiplier * (the chart mean + three times the sigma)

  1 * (**2145** + (3 * **141**)) = 2568

- maximum multiplier * (the highest value in the chart + one sigma)

  1 * (**2327** + **141**) = 2468

6.2 Using ^%SYSMONMGR to Manage Health Monitor

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

```
%SYS>do ^%SYSMONMGR

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

Option?
```

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

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

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

Option?
```

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

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

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

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

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

6.2.1 View Alerts Records

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

6.2.2 Configure Health Monitor Classes

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

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

Option	Description
`1) Activate/ Deactivate Rules`	(not in use in this release)
`2) Configure Periods`	List the currently configured periods and add and delete periods.

Option	Description
`3) Configure Charts`	Lets you • List the mean and sigma of all existing charts, organized by period. • Examine individual charts in detail, including the readings on which the mean and sigma are based, with the highest reading called out. • Change the mean and sigma of an existing chart using the Edit Charts option. • Create a chart, specifying alert and warning thresholds. • Manually recalibrate all charts (including user-created charts) or an individual chart from the most recent data.
`4) Edit Sensor Objects`	List the sensor objects representing the sensors in the SYS.Monitor.SystemSensors class and modify their base, maximum, warning, maximum multiplier, and warning multiplier values.
`5) Reset Defaults`	Lets you • Reset to the default period configuration and remove all existing charts, returning every period to analysis mode (see Health Monitor Process Description). • Remove all existing charts (including user-created charts), returning every period to analysis mode, without removing any user-defined period configuration. • Reset all sensor objects to their default values. • Reset the health monitor options (startup wait time and alert purge time) to their defaults

6.2.3 Set Health Monitor Options

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

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

6.3 See Also

- Introduction to System Monitor
- Using Core System Monitor

- Application Monitor
- System Monitoring Tools
- Manage Email Options (information about generating email messages from notifications in the messages log, including those generated by System Monitor)
- Monitoring Log Files (includes information on the log files generated by this tool)

7

Using Application Monitor

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

Application Monitor is part of the System Monitor tools.

7.1 Overview

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

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

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

7.2 Using ^%SYSMONMGR to Manage Application Monitor

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

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

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

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

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

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

Option?
```

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

7.2.1 Manage Application Monitor

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

Option	Description
`1) Set Sample Interval`	Sets the interval at which metrics are sampled; the default is 30 seconds. This setting can be overridden for an individual class by setting a class-specific interval (using the `Set Class Sample Interval` option on the Manage Monitor Classes submenu). Note: if the System Monitor sampling interval (see `Set Sample Interval` in the Set System Monitor Options submenu) is longer than the sampling interval for an Application Monitoring class, the longer of the two intervals is used. For example, if the System Monitor interval is 30 and the Application Monitor interval is 120, all active Application Monitor classes are sampled every 120 seconds; if the System Monitor interval is 60 and the %Monitor.System.LockTable class interval is 20, the class is sampled every 60 seconds.
`2) Manage Monitor Classes`	Displays the Manage Monitor Classes submenu which lets you manage system- and user-defined monitor classes in the namespace in which you are running the Application Monitor Manager.
`3) Change Default Notification Method`	Lets you specify the default action for alerts when triggered. Any alerts you create use this action unless you specify otherwise.
`4) Manage Email Options`	Displays the Monitor Email Options submenu which lets you enable and configure email notifications so you can specify this action in alerts.
`5) Manage Alerts`	Displays Manage Alerts submenu which lets you create alerts for system and user-defined monitor classes.

7.2.2 Manage Monitor Classes

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

```
Option? 2

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

Option?
```

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

Option	Description
`1) Activate / Deactivate Monitor Class`	Application Monitor samples active classes only. This option lets you activate an inactive class, or deactivate an active one. You can display a numbered list of the system and user-defined classes registered in the local namespace, including the activation state of each, by entering **?** at the `Class?` prompt, then enter either the number or class name.
`2) List Monitor Classes`	Displays a list of the system- and user-defined classes registered in the local namespace, including the activation state of each.

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

7.2.2.1 Debug Monitor Classes

This submenu lets you manage system debugging.

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

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

```
Option? 6

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

Option?
```

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

Input Field	Description
`1) Enable Debug`	Lets you enable debugging. If the class is not creating sample values, then you can check to see if errors are preventing the sample values from being saved.
`2) Disable Debug`	Lets you disable debugging.

Input Field	Description
`3) List Errors`	Displays the definitions of all errors in the local namespace; for example: **%Save()**, **%New()**, **Initialize()** and **GetSample()**. Enable debugging for the class using **^%SYSMONMGR**, and the System Monitor will save the last error caught for specific methods within the class.

7.2.3 Change Default Notification Method

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

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

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

Input Field	Description
`0`	Do not take action when an alert is triggered.
`1`	Send an email message to the configured recipients when an alert is triggered. For information about configuring email, see Manage Email Options.
`2`	Call a notification method when an alert is triggered. If you select this action, the method is called with two arguments – the application name specified in the alert and a %List object containing the properties returned to the monitor class by the sample class (as described in Application Monitor Overview). When prompted, enter the full class name and method, that is *packagename.classname.method*. This method must exist in the local namespace.

7.2.4 Manage Email Options

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

```
Option? 4

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

Option?
```

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

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

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

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

7.2.5 Manage Alerts

An alert specifies:

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

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

- Condition: The lock table is over 90% full, defined by the UsedSpace property (returned when the %Monitor.System.LockTable class calls %Monitor.System.Sample.LockTable) being more than 90% of the TotalSpace property.
- Action: Send an email notification.

The definition of a condition based on properties is called an *evaluation expression*; after specifying the properties of the sample class you want to use, you specify the evaluation expression. Properties are indicated in the expression by placeholders corresponding to the order in which you provide them; for example, if when creating the lock table alert you specify

the UsedSpace property first and then the TotalSpace property, you would enter the evaluation expression as **%1 / %2 > .9**, but if you enter the properties in the reverse order, the expression would be **%2 / %1 > .9**.

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

```
Option? 2

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

Option?
```

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

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

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

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

Input Field	Description
`6) Clear NotifyOnce Alert`	Lets you set an internal `Notified` flag for a specified Alert name when the Alert is triggered. When set, it will not post another Alert.

The following table describes the valid responses to Alert prompts:

Table 7–1: Responses to Alert Prompts

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

Input Field	Description
`Property?`	Enter the name of a property defined in the class specified in the preceding prompt that you are using in the evaluation expression, in the descriptive text, or in both.. To display a numbered list of properties defined in the named class, enter **?** at the `Property?` prompt, then enter a number or name. Each property must be entered individually. When you are done, press **Enter** at an empty prompt to display the list of properties in the order in which you specified them.
`Evaluation expression?`	Expression used to evaluate the properties specified at the `Property?` prompt. For example, in **`(%1 = "User") && (%2 < 100)`**, %1 refers to the first property in the list of properties, %2 the second property, and so on.
`Notify once only?`	Enter one of the following: • **`Yes`** – Notify users only the first time an alert is triggered. • **`No`** – Notify users every time an alert is triggered.

7.3 Application Monitor Metrics

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

Sample Classes	Application Monitor System Classes
Audit metrics	%Monitor.System.Sample.AuditCount and %Monitor.System.Sample.AuditEvents
Client metrics	%Monitor.System.Sample.Clients
Web Gateway metrics	%Monitor.System.Sample.CSPGateway
Disk space metrics	%Monitor.System.Sample.Diskspace
Free space metrics	%Monitor.System.Sample.Freespace
Global metrics	%Monitor.System.Sample.Globals
History database metrics (see History Monitor)	%Monitor.System.Sample.HistoryPerf, %Monitor.System.Sample.HistorySys, %Monitor.System.Sample.HistoryUser
Journal metrics	%Monitor.System.Sample.Journals
License metrics	%Monitor.System.Sample.License
Lock table metrics	%Monitor.System.Sample.LockTable
Process metrics	%Monitor.System.Sample.Processes
Routine metrics	%Monitor.System.Sample.Routines
Server metrics	%Monitor.System.Sample.Servers
System activity metrics	%Monitor.System.Sample.SystemMetrics

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

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

7.3.1 Generating Metrics

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

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

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

7.3.2 Viewing Metrics Data

All metrics classes are CSP-enabled; the CSP code is generated automatically when the sample class is generated. Therefore, the simplest way to view metrics is using a web browser. Based on the example in Generating Metrics, the CSP URL has the following form, which uses the <baseURL> for your instance:
http://<baseURL>/csp/user/MyMetrics.Sample.Freespace.cls. The output displayed is similar to:

```
Monitor - Freespace c:\InterSystems\IRIS51\
            Name of dataset:  c:\InterSystems\IRIS51\
Current amount of Freespace:  8.2MB

Monitor - Freespace c:\InterSystems\IRIS51\mgr\
            Name of dataset:  c:\InterSystems\IRIS51\mgr\
Current amount of Freespace:  6.4MB
```

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

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

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

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

Monitor - Freespace    c:\InterSystems\IRIS51\
                Name of dataset:  c:\InterSystems\IRIS51\
    Current amount of Freespace:  8.2MB

Monitor - Freespace    c:\InterSystems\IRIS51\mgr\
                Name of dataset:  c:\InterSystems\IRIS51\mgr\
    Current amount of Freespace:  6.4MB
```

Note: The URL for a class with % (percent sign) in the name must use %25 in its place. For example, the URL for the %Monitor.System.Freespace class has the following form, which uses the <baseURL> for your instance:

http://<baseURL>/csp/sys/%25Monitor.System.Freespace.cls

7.4 Writing User-Defined Application Monitor Classes

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

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

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

```
SAMPLES>d ^%MONAPPMGR


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

Option? 2

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

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

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


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

Option?
```

2. Write a class that inherits from %Monitor.Adaptor. The inheritance provides persistence, parameters, properties, code generation, and a projection that generates the monitor metadata from your class definition. See the %Monitor.Adaptor class documentation for full details on this class, as well as the code you must write.

3. Compile your class. Compiling classes that inherit from %Monitor.Adaptor generates new sample classes in a subpackage of the users class called Sample. For example, if you compile A.B.MyMetric, a new class is generated in A.B.Sample.MyMetric. You do not need to do anything with the generated class.

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

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

Important: The **SECURITYRESOURCE** parameter is empty in %Monitor.Adaptor, and therefore in user classes inheriting from %Monitor.Adaptor unless explicitly modified. Code generation copies the **SECURITYRESOURCE** value from the user-defined class to the generated sample class.

The following simple example retrieves the free space for each dataset in an InterSystems IRIS instance.

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

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

   Class Definition

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

2. Add properties that you want to be part of the sample data. Their types must be classes within the %Monitor package:

   - Gauge
   - Integer
   - Numeric
   - String

   For example:

   Class Definition

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

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

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

   Class Member

   ```
   Parameter INDEX = "DBName";
   ```

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

   Class Member

   ```
   /// Result Set for use by the class
   Property Rspec As %SQL.StatementResult [Internal];
   ```

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

Class Member

```
/// Initialize the list of datasets and freespace.
Method Initialize() As %Status
{
    set stmt=##class(%SQL.Statement).%New()
    set status= stmt.%PrepareClassQuery("SYS.Database","FreeSpace")
    set ..Rspec = stmt.%Execute()
    return $$$OK
}
```

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

Class Member

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

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

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

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

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

Class Definition

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

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

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

/// Result Set
Property Rspec As %SQL.StatementResult [Internal];

/// Initialize the list of datasets and freespace.
Method Initialize() As %Status
{
    set stmt=##class(%SQL.Statement).%New()
    set status= stmt.%PrepareClassQuery("SYS.Database","FreeSpace")
    set ..Rspec = stmt.%Execute()
    return $$$OK
}

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

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

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

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

}
```

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

```
/// Open a tcp/ip device to send warnings
Property io As %Status;

Method Startup() As %Status
{
    set ..io="|TCP|2"
    set host="127.0.0.1"
    open ..io:(host:^serverport:"M"):200
}

Method Shutdown() As %Status
{
    close ..io
}
```

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

Class Definition

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

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

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

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

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

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

}
```

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

Important: If you do modify and recompile an active user-defined Application monitor class, the class is deactivated and the class-specific sample interval override, if any, is removed; to restore it, you must activate it, reset the sample interval if desired, and restart of System Monitor.

7.5 See Also

- Introduction to System Monitor
- Using Core System Monitor
- Using Health Monitor
- System Monitoring Tools

- Manage Email Options (information about generating email messages from notifications in the messages log, including those generated by System Monitor)
- Monitoring Log Files (includes information on the log files generated by this tool)

8

Gathering Global Activity Statistics Using ^GLOSTAT

InterSystems IRIS® data platform provides the **^GLOSTAT** utility, which gathers global activity statistics and displays a variety of information about disk I/O operations.

You can also view the statistics reported by **^GLOSTAT** from the Management Portal. Log in to the Portal for the system you are monitoring and navigate to the **System Usage** page (**System Operation** > **System Usage**).

8.1 Running ^GLOSTAT

To run the **^GLOSTAT** routine you must be in the %SYS namespace. The name of the routine is case-sensitive. Type the following command and press **Enter**:

ObjectScript

```
do ^GLOSTAT
```

The **^GLOSTAT** routine displays statistics, as seen in Example A. Each time InterSystems IRIS starts, it initializes the **^GLOSTAT** statistical counters; the initial output of **^GLOSTAT** reflects operations since InterSystems IRIS has started.

The following prompt appears beneath the report:

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

You may enter one of the following:

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

8.2 Overview of ^GLOSTAT Statistics

Each **^GLOSTAT** statistic represents the number of times a type of event has occurred since InterSystems IRIS has started, or per second during a defined interval. You may run **^GLOSTAT** at any time from the system manager's namespace. In most cases, you should run the utility on an active system, not an idle one.

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

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

Table 8–1: Statistics Produced by ^GLOSTAT

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

8.3 Examples of ^GLOSTAT Output

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

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

8.3.1 Example A

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

```
%SYS>do ^GLOSTAT

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

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

8.3.2 Example B

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

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

Counts per Second for 30 Seconds...

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

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

8.3.3 Example C

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

```
%SYS>do ^GLOSTAT

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

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

9

Monitoring System Performance Using ^PERFMON

This page describes how to monitor system performance using the **^PERFMON** utility.

9.1 Introduction

^PERFMON is an ObjectScript utility that controls the **MONITOR** facility.

The **MONITOR** facility provides performance data for the InterSystems IRIS® data platform 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. InterSystems IRIS allocates memory at **MONITOR** startup to create slots for the number of processes, routines, globals, and nodes specified. The first process to trigger an event counter allocates the first slot and continues to add to that set of counters. Once the facility allocates all the available slots to processes, it includes any subsequent process counts in the *Other* slot. It follows the same procedure for globals, routines, and nodes.

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

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

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

9.2 Using ^PERFMON

You can run the **^PERFMON** routine in two ways: interactively in the InterSystems Terminal, or by individual calls to its functions. All the options of **^PERFMON** are available using either method.

^PERFMON contains the following functions:

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

Each function returns a status of success (`1`) or failure (a negative number, followed by a comma and a brief message).

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 an InterSystems IRIS 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.
```

9.2.1 Running ^PERFMON Interactively

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

1. Enter the following command from the **%SYS** namespace:

   ```
   do ^PERFMON
   ```

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

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

   Monitor is Stopped

   Enter the number of your choice:
   ```

3. Each option corresponds directly to a ^PERFMON function, and will prompt you for the necessary parameters. For example, entering `1` corresponds to Start function:

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

   Monitor is Stopped

   Enter the number of your choice: 1


   Processes <24>:
   Routine <200>:
   Globals <100>:
   Databases <10>:
   Network nodes <5>:
   ```

9.3 Start

Turns on collection of the statistics.

Format:

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

Parameters:

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

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

Status Codes:

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

9.4 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

9.5 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

9.6 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

9.7 Sample Counters

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

Format:

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

Parameters:

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

Status Codes:

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

9.8 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

9.9 Report

The report function gathers and outputs a report of metrics.

Format:

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

Parameters:

- *report* — type of report to output; valid values are:
  - `G` – for global activity
  - `R` – for routine activity
  - `N` – for network activity
  - `C` – for a custom report where you select the metrics to report
- *sort* — grouping and sort order of report; valid values are:
  - `P` – to organize the report by Process
  - `R` – to organize the report by Routine
  - `G` – to organize the report by Global
  - `D` – to organize the report by Database

  - `I` – to organize the report by Incoming node
  - `O` – to organize the report by Outgoing node

- *format* — output format; valid values are:
  - `P` – for a printable/viewable report (.txt file, no pagination)
  - `D` – for comma delimited data (.csv file) which can be read into a spreadsheet
  - `X` – for Microsoft Excel XML markup suitable for import into Excel (.xml file)
  - `H` – for an HTML page (.html file)

- *output* — enter a file name, **Return** to accept the default file name displayed, or `0` (zero) for output to the screen
- *list* — (*for Custom report only*) comma-separated list of metric numbers which specify what columns to include in the report; input `?` after specifying a custom report for a list of all possible metrics and their numbers.

  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.

9.10 Collect

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

Format:

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

Parameters:

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

Status Codes:

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

9.11 Report Examples

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

```
%SYS>Do ^PERFMON


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

Enter the number of your choice: 7


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


Press RETURN to continue ...
```

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

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

Enter the number of your choice: 7


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

9.12 See Also

- %Monitor.System package
- Application Monitor
- Examining Routine Performance Using ^%SYS.MONLBL

10

Monitoring Routine Performance Using ^PROFILE

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

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

By default, **^PROFILE** captures metrics for up to 5000 routines; if there is not enough shared memory available for the maximum number of routines, the utility displays a message about the number of pages of memory required to monitor this collection and the number of pages available. Then, the utility captures metrics for as many routines as possible.

10.1 Using ^PROFILE

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

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

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

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

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

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

Column Title (Metric)	Description
RtnLoad	Number of times the routine is loaded. By default, the entry is listed as a percentage of all routine loads.
GloRef	Number of global references by the routine. By default, the entry is listed as a percentage of global references by all routines.
GloSet	Number of global sets by the routine. By default, the entry is listed as a percentage of global sets by all routines.

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

Follow the instructions that are displayed in the Terminal:

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

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

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

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

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

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

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

Option	Description
#	The line number in the code you want to analyze in more detail. After you press **Enter**, the code for the specified label is displayed.
B	Display the previous page of the list.
C	Switch code display between source code and intermediate (INT/MVI) code.
M	Change the page margin and length.
N	Display the next page of the list.
O	Re-sort the page based on different metrics.
Q	Exit the list, returning to the previous level.

Option	Description
R	Refresh the list with the most recent metrics.
S	Switch to the subroutine level display of the routine.

10.2 ^PROFILE Example

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

1. Enter the following command:

```
do ^PROFILE
```

2. The following message appears.

```
WARNING: This routine will start a system-wide collection of
data on routine activity and then display the results. There
may be some overhead associated with the initial collection,
and it could significantly affect a busy system.

The second phase of collecting line level detail activity
has high overhead and should ONLY BE RUN ON A TEST SYSTEM!

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

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

```
 Waiting for initial data collection ...

      RtnLine     Time        CPU         RtnLoad     GloRef      GloSet
1.    41.48%      12.19%      0.00%       28.97%      10.65%      0.00%
      %Library.ResultSet.1.INT (IRISLIB)
2.    35.09%      56.16%      65.22%      9.35%       36.77%      42.55%
      SYS.Database.1.INT (IRISSYS)
3.    10.75%      6.62%       0.00%       43.30%      22.68%      46.81%
      Config.Databases.1.INT (IRISSYS)
4.    7.13%       3.22%       0.00%       6.23%       0.00%       0.00%
      %Library.Persistent.1.INT (IRISLIB)
5.    1.26%       0.71%       0.00%       4.36%       4.12%       4.26%
      PROFILE.INT (IRISSYS)
6.    1.20%       0.00%       0.00%       0.00%       5.15%       6.38%
      %SYS.WorkQueueMgr.INT (IRISSYS)
7.    0.76%       15.08%      34.78%      0.00%       0.00%       0.00%
      %SYS.API.INT (IRISSYS)
8.    0.64%       1.05%       0.00%       0.00%       17.18%      0.00%
      %Library.JournalState.1.INT (IRISLIB)
9.    0.61%       0.31%       0.00%       3.74%       0.00%       0.00%
      %Library.IResultSet.1.INT (IRISLIB)
10.   0.28%       0.93%       0.00%       0.00%       1.72%       0.00%
      %Library.Device.1.INT (IRISLIB)
11.   0.24%       0.71%       0.00%       0.62%       0.00%       0.00%
      Config.CPF.1.INT (IRISSYS)
Select routine(s) or '?' for more options  N =>
```

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

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

```
There are 2 routines selected for detailed profiling. You may now
end the routine level collection and start a detailed profiler collection.

WARNING !!

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

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

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

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

Loading ^%Library.Persistent.1 in ^^c:\intersystems\iris\mgr\irislib\

Detail level Profile collection started.




    RtnLine     Routine Name  (Database)
1.  96.72%      %Library.Persistent.1.INT (IRISLIB)
2.  3.28%       Config.CPF.1.INT (IRISSYS)
Select routine to see details or '?' for more options  R =>
```

7. After you select the routine whose code you want to analyze, the routine displays a page with information about the code.

11

Examining Routine Performance Using ^%SYS.MONLBL

The routine **^%SYS.MONLBL** provides a user interface to the InterSystems IRIS® data platform **MONITOR** facility. This utility provides a way to diagnose where time is spent executing selected code in 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 an InterSystems IRIS instance.

11.1 Invoking the Line-by-line Monitoring Routine

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

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


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

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

1.) Start Monitor
2.) Memory Requirements

Enter the number of your choice:
```

- Enter `1` to begin the dialog to provide the appropriate information to start monitoring.
- Enter `2` to calculate an estimate of how much memory a collection needs before actually starting the monitor. See Estimate Memory Requirements for details.
- Press the `Enter` key to exit the routine.

11.1.1 Start Monitoring

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

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

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

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

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

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

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

The time metrics are clock time and are measured in seconds.

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

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

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

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

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

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

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

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

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

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

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

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

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

Enter PID (press 'Enter' to terminate)

PID: 1640
PID: 2452
PID:
```

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

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

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

```
Monitor started.

Press RETURN to continue ...
```

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

11.1.2 Estimate Memory Requirements

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

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

You can maintain the **gmheap** (generic memory heap) setting from the **Advanced Memory Settings** page of the Management Portal (**System Administration** > **Configuration** > **Additional Settings** > **Advanced Memory**).

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

```
Enter the number of your choice: 2

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

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

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

Enter the number of your choice: <1> 5

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

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

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

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

Press RETURN to continue ...
```

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

11.2 Line-by-line Monitoring Options

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

```
Line-by-Line Monitor

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

Enter the number of your choice:
```

The first three options are fairly self-explanatory:

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

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

The following subsections explain the four report options.

11.2.1 Report Line-by-line Statistics

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

You have four types of reports to choose from:

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

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

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

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

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

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

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

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

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

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

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

```
FileName: monlbl_JRN_dtl.txt
```

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

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

11.3 Sample Line-by-line Detail Report

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

```
Line-by-Line Monitor

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

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

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

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

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

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

```
Routine ^JRNDUMP ...

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

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

This is a partial display of one selected routine.

11.4 Sample Line-by-line Summary Report

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

```
Line-by-Line Monitor

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

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


Press RETURN to continue ...
```

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

```
Routine              Lines   LinesHit    Percent    RtnLine       Time
JRNOPTS                109         60     55.05%        155  14.172230
JRNSWTCH               249         58     23.29%         69   0.926131
JRNDUMP                579        100     17.27%        582  17.265002
JRNSTART               393         23      5.85%         23   0.005541
JRNUTIL                872         39      4.47%         39   0.116995
JRNUTIL2               276          8      2.90%         56   0.006056
JRNCHECK                18          0      0.00%
```

```
JRNCLFOR               416          0      0.00%
JRNCLUREST             193          0      0.00%
JRNCLUREST2            229          0      0.00%
JRNINFO                263          0      0.00%
JRNMARK                195          0      0.00%
JRNRESTB              1315          0      0.00%
JRNRESTC              1245          0      0.00%
JRNRESTC2              540          0      0.00%
JRNRESTCHELP           122          0      0.00%
JRNRESTD               445          0      0.00%
JRNRESTO               859          0      0.00%
JRNROLL                827          0      0.00%
JRNSTAT                 62          0      0.00%
JRNSTOP                119          0      0.00%
JRNWUTL                235          0      0.00%

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

This is the complete sample report.

11.5 Sample Line-by-line Delimited Output Report

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

```
Line-by-Line Monitor

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

Enter the number of your choice: 6

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

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


Press RETURN to continue ...
```

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

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

This is a partial display of one selected routine.

11.6 Sample Line-by-line Procedure Level Report

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

```
Line-by-Line Monitor

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

Enter the number of your choice: 7

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

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


Press RETURN to continue ...
```

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

```
Routine ^JRNDUMP ...

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

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

11.7 Metrics Shown in These Reports

These reports show the following metrics:

- `GloRef` — global references
- `GloSet` — global sets
- `GloKill` — global kills
- `DirBlkRd` — directory block reads
- `UpntBlkRd` — upper pointer block reads
- `BpntBlkRd` — bottom pointer block reads
- `DataBlkRd` — data block reads
- `BdataBlkRd` — big data block reads
- `MapBlkRd` — map block reads
- `OthBlkRd` — other block reads
- `DirBlkWt` — directory block writes
- `UpntBlkWt` — upper pointer block writes
- `BpntBlkWt` — bottom pointer block write
- `DataBlkWt` — data block writes
- `BdataBlkWt` — big data block writes
- `MapBlkWt` — map block writes
- `OthBlkWt` — other block writes
- `DirBlkBuf` — directory block requests satisfied from a global
- `UpntBlkBuf` — upper pointer block requests satisfied from a global buffer
- `BpntBlkBuf` — bottom pointer block requests satisfied from a global buffer
- `DataBlkBuf` — data block requests satisfied from a global buffer

- `BdataBlkBuf` — big data block requests satisfied from a global buffer
- `MapBlkBuf` — map block requests satisfied from a global buffer
- `OthBlkBuf` — other block requests satisfied from a global buffer
- `JrnEntry` — journal entries
- `BlkAlloc` — blocks allocated
- `NetGloRef` — network global refs
- `NetGloSet` — network sets
- `NetGloKill` — network kills
- `NetReqSent` — network requests sent
- `NCacheHit` — network cache hits
- `NCacheMiss` — network cache misses
- `NetLock` — network locks
- `RtnLine` — lines of ObjectScript
- `RtnLoad` — routine loads
- `RtnFetch` — routine fetches
- `LockCom` — lock commands
- `LockSucc` — successful lock commands
- `LockFail` — failed lock commands
- `TermRead` — terminal reads
- `TermWrite` — terminal writes
- `TermChRd` — terminal read chars
- `TermChWrt` — terminal write chars
- `SeqRead` — sequential reads
- `SeqWrt` — sequential writes
- `IJCMsgRd` — local interjob communication (IJC) messages read
- `IJCMsgWt` — local IJC messages written
- `IJCNetMsg` — network IJC messages written
- `Retransmit` — network retransmits
- `BuffSent` — network buffers sent
- `Time` — elapsed time (sum of each time it is hit)
- `TotalTime` — total time used (including subroutines)

11.8 Line-by-line Monitor Programming Interface

Programmers can also interface with the InterSystems IRIS **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 %Monitor.System.LineByLine for more details.

12

Tracing Process Performance with ^TRACE

The ^**TRACE** utility offers functionality to trace the execution of InterSystems IRIS processes. Traced processes write events to a trace file with information about the routine line, where it occurred and, if applicable, the global reference.

12.1 Using ^TRACE

Note: The trace files may contain sensitive information such as global references or parameters passed to subroutines. They will not contain the values of any globals.

The events available for tracing correspond to the metrics reported in performance monitoring tools (for example ^PERFMON or %SYS.MONLBL). Raw data is written to a trace file, iristrace_pid.txt, in a specified directory.

Note: The trace directory must be writable by the processes being traced.

Different sets of trace events can be selected to produce traces for different purposes. Highly detailed application execution tracing can be achieved; this can include tracing all global references (GloRef), all application subroutine calls (RtnLoad), or every line of application code executed (RtnLines). Alternatively, tracing can be limited to less common events such as physical block reads (DataBlkRd, UpntBlkRd, etc), network cache misses (NCacheMiss), or block collisions (BlkWait), in order to find all the locations in the application where these occurrences may be affecting performance.

Note: The ability to configure the trace, start tracing a process, or use the ^**TRACE** utility requires **`%Admin_Manage:USE`**.

13

Monitoring Performance Using ^SystemPerformance

This page describes the **^SystemPerformance** utility, (previously named ^pButtons), a tool for collecting detailed performance data about an InterSystems IRIS® data platform instance and the platform on which it is running. You can send the resulting report to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems. **^SystemPerformance** is similar to Diagnostic Reports (see Using the Diagnostic Report), but focuses on performance data.

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

You can run the profiles in the Terminal or schedule runs in the Management Portal with Task Manager. In addition, you can add, modify, and delete profiles using the API that is included with the utility.

Important: Before using **^SystemPerformance** in any unattended way, use it interactively to be sure that you have the appropriate privileges at the operating-system level. In particular, some environments require a UAC login to get elevated privileges. In such a case, the utility will fail to capture data and will instead writes the error

```
Error: Access is denied. You're running with a restricted token, try running elevated.
```

If this occurs, you can open a Windows CMD as administrator, start an ObjectScript shell from there, and then use **^SystemPerformance**.

If you share the generated report with InterSystems, note that it includes private application information. InterSystems keeps all data strictly confidential.

13.1 Basics

The **^SystemPerformance** utility lets you select one or more profiles to run. (The profiles available vary depending on the product version and any customization that has been performed.) Based on the selected profile(s), it generates a set of log files, which are placed in the output directory. By default, the output directory is the *install-dir*\mgr directory of the InterSystems IRIS instance, but you can change the output directory.

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

- 12hours — 12-hour run sampling every 10 seconds

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

To run the **^SystemPerformance** utility:

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

```
%SYS>do ^SystemPerformance
```

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

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

select profile number to run:
```

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

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

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

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

13.2 Stopping ^SystemPerformance

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

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

To stop the job without deleting log files and produce an HTML performance report from those log files, enter:

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

For more information, see Functions in ^SystemPerformance.

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

13.3 Functions in ^SystemPerformance

The ^**SystemPerformance** utility provides options for the start, collect, preview, and stop functions as described in the following list:

Note: You can run multiple profiles concurrently.

$$run^SystemPerformance("*profile*")

Starts the specified *profile*. If successful, returns the *runid*; if unsuccessful, returns 0.

$$literun^SystemPerformance("*profile*")

Same as the preceding, *except* that it does not include operating-system data.

Note: This option is intended for servers that are running multiple instances of InterSystems IRIS, where the operating-system data would be duplicated.

$$Collect^SystemPerformance("*runid*")

Produces a readable HTML performance report file for the specified *runid*. If successful, returns 1 and the report filename; if unsuccessful, returns 0 followed by a carat and the reason for the failure.

If no *runid* is supplied, `$$Collect` generates a report for each completed *runid* that does not yet have a report. In this mode, the return value is the number of reports generated followed by a carat and the word 'collected'. For example `5^collected` if 5 reports were generated, or `0^collected` if none were. If run at a command prompt, `$$Collect` lists which profiles are still running, if any, and how much time remains for each.

Note: It is not necessary to call `$$Collect` as part of standard operations, because ^**SystemPerformance** does so automatically. In unusual circumstances such as ^**SystemPerformance** being interrupted by an instance restart, if the automatic mechanism does not execute, `$$Collect` may be needed.

$$Preview^SystemPerformance("*runid*")

Produces a readable HTML interim (incomplete) performance report file for the specified *runid*. If successful, returns 1 followed by a carat and the file location. If unsuccessful, returns 0 followed by a carat and the reason for the failure.

$$Stop^SystemPerformance("*runid*",[0])

Stops (aborts) ^**SystemPerformance** from collecting data for a specified *runid* and by default deletes the associated .log files produced by the utility. To stop data collection without deleting the .log files and produce an HTML performance report from those log files, include the 0 parameter following the *runid*.

If unsuccessful, the function returns 0 followed by a carat and the reason for the failure; if successful, it returns: *1:2:3:4_1:2:3:4*. The successful status is made up of two parts separated by an underscore: OS-specific and InterSystems IRIS-specific; within each part, colon-separated values specify:

1. Number of jobs successfully stopped
2. Number of jobs that failed to stop
3. Number of files successfully deleted
4. Number of files not deleted

$$waittime^SystemPerformance("*runid*")

Reports the time until the final HTML file for the specified *runid* will be complete. If the *runid* is finished, returns `ready now`, otherwise returns a string of the form *XX* `hours` *YY* `minutes` *ZZ* `seconds`.

do EnableEnsqcnt^SystemPerformance()

Enables the collection additional data for interoperability productions.

do DisableEnsqcnt^SystemPerformance()

Disables the collection of additional data for interoperability productions.

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

Terminal

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

%SYS>set status=$$Collect^SystemPerformance(runid)
SystemPerformance run 20181004_123815_30mins is not yet ready for collection.

%SYS>write status
0^not ready

%SYS>set status=$$Preview^SystemPerformance(runid)

%SYS>write status
1^c:\intersystems\iris\mgr\USER_IRIS_20181004_123815_30mins_P1.html
%SYS>
```

13.4 Generating ^SystemPerformance Performance Reports

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

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

13.5 Scheduling the ^SystemPerformance Utility with Task Manager

This section provides examples using the Task Manager in the Management Portal to schedule **^SystemPerformance** to run. For general instructions for how to schedule a task, see Schedule Task Manager.

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

Example 1: Weekly 24-Hour Run

In this example, a task is created to schedule the **^SystemPerformance** utility to run a profile named 24hours (which collects performance data for 24 hours) every Thursday at 09:00:

1. From the **Task Manager** page of the Management Portal (**System Operation** > **Task Manager**), choose the **New Task** option to start the **Task Scheduler Wizard**. Then enter the following information in the specified fields:
   - **Task name** — enter `24HourRun`.
   - **Description** — enter `Start 24-hour ^SystemPerformance Run`.
   - **Namespace to run task in** — select **%SYS** from the drop-down list.
   - **Task type** — select **RunLegacyTask** from the drop-down list.

     In the **ExecuteCode** text box, enter the following code:

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

   - **Output file** — leave blank; the task has no output (see Changing the Output Directory for information on customizing the output directory).
2. Click **Next**. Then enter the following information in the specified fields:
   - **How often ...** — choose **Weekly** from the drop-down list.

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

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

Example 2: Daily 30–Minute Run

In this example, a task is created to schedule the **^SystemPerformance** utility to run a profile named 30mins (which collects performance data for 30 minutes) every day at 12:00:

1. From the **Task Manager** page of the Management Portal (**System Operation** > **Task Manager**), choose the **New Task** option to start the **Task Scheduler Wizard**. Then enter the following information in the specified fields:
   - **Task name** — enter `30MinRun`.
   - **Description** — enter `Start 30-minute ^SystemPerformance Run`.
   - **Namespace to run task in** — select **%SYS** from the drop-down list.
   - **Task type** — select **RunLegacyTask** from the drop-down list.

In the **ExecuteCode** text box, enter the following code:

```
do run^SystemPerformance("30mins")
```

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

2. Click **Next**. Then enter the following information in the specified fields:
   - **How often ...** — choose **Daily** from the drop-down list.
   - **Start Date** — enter the start date in the text box.

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

13.6 Changing the Output Directory

The default output directory for both the log files and the resulting HTML report file is the *install-dir*\mgr of the InterSystems IRIS instance for which you are running the **^SystemPerformance** utility. You can change the default directory using the commands described in the following list.

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

`do setlogdir^SystemPerformance("directory")`

Sets the pathname of output directory to *directory*; if the directory does not exist, it is created.

Note: If you do not specify an absolute pathname (for example, `C:\Reports`), the directory is assumed to be relative to the *install-dir*\mgr directory.

`set x = $$getlogdir^SystemPerformance()`

Sets variable *x* equal to the output directory pathname.

`do clrlogdir^SystemPerformance()`

Resets the output directory pathname to the default directory (*install-dir*\mgr).

13.7 Getting Version Information

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

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

13.8 Manipulating Profiles

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

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

13.8.1 Create New Profiles

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

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

You must specify:

- *profilename* — A name for the profile, which must be unique and cannot contain spaces or whitespace characters.
- *description* — A description of the profile that is displayed in the **^SystemPerformance** menu.
- *interval* — The frequency with which to run each sample, in seconds (in the range of 1 second to 300 seconds). An interval of 1 second is only allowed if the profile duration is an hour or less.
- *count* — The number of times to run the profile.

The function returns 1 if successful and 0 if unsuccessful. This is followed by a carat and then the reason for any errors.

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

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

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

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

13.8.1.1 Generate Profile

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

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

Where:

- *duration* — How long the profile should run. The valid formats are "*hh:mm*", "*hh:*", or *mm*
- *interval* (optional) — The frequency with which to run each sample, in seconds (in the range of 1 second to 300 seconds). An interval of 1 second is only allowed if the profile duration is an hour or less.

The function returns 1 if successful. If unsuccessful, it returns 0 followed by a carat and the reason for any errors.

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

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

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

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

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

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

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

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

13.8.2 Edit Profiles

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

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

Where:

- *profilename* — The name of the existing profile you want to edit.
- *description* — A description of the profile that is displayed in the **^SystemPerformance** menu.
- *interval* (optional) — The frequency with which to run each sample, in seconds (in the range of 1 second to 300 seconds). An interval of 1 second is only allowed if the profile duration is an hour or less.
- *count* (optional) — The number of times to run the profile.

The function returns 1 if successful and 0 if unsuccessful. This is followed by a carat and then the reason for any errors.

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

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

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

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

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

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

13.8.3 Copy Profiles

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

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

You must specify:

- *sourceprofilename* — The name of an existing profile
- *targetprofilename* — The name of the profile you want to create. This must be unique and cannot contain spaces or whitespace characters. This must be double-quoted.

The function returns 1 if successful. If unsuccessful, it returns 0 followed by a carat and the reason for any errors.

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

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

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

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

You can now edit the new profile as described in Edit Profiles.

13.8.4 Delete Profiles

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

```
set rc=$$delprofile^SystemPerformance("profilename")
```

You must specify *profilename*, the name of the profile you want to delete. This must be double-quoted.

The function returns 1 if successful. If unsuccessful, it returns 0 followed by a carat and the reason for any errors.

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

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

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

13.9 Performance Report Details

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

```
Configuration

IRISTEST3 on machine testsystem

Customer: InterSystems Development
License : 123456

InterSystems IRIS Version String: InterSystems IRIS for Windows (x86-32) 2021 (Build 508) Fri Jan 26
2018 17:51:22 EDT
------------------------------------------------------------
Profile
```

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

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

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

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

- InterSystems IRIS Performance Data Report for Microsoft Windows Platforms
- InterSystems IRIS Performance Data Report for Apple macOS Platforms
- InterSystems IRIS Performance Data Report for IBM AIX® Platforms
- InterSystems IRIS Performance Data Report for Linux Platforms

Note: In all of the following tables, data marked with * is collected once per run.

Table 13–1: InterSystems IRIS Performance Data Report for Microsoft Windows Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
Configuration *	InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.

Section	Description
irisstat -c	Four samples taken at even intervals over the course of the run using the command **.\bin\irisstat -s -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all IRIS.DAT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility.
irisstat -D	Eight samples taken at even intervals over the course of the run using the command **irisstat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility. For information about monitoring block collisions using the **^BLKCOL** utility, see Monitoring Block Collisions Using ^BLKCOL.
license *	InterSystems IRIS license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
mgstat	InterSystems IRIS-specific data taken over the course of the run using the **^mgstat** utility. See Monitoring Performance Using ^mgstat.

Section	Description

perfmon	Output from the Microsoft Windows **perfmon** utility.
	
	The default presentation of Microsoft Windows **perfmon** data is raw format. The format can be switched to processed, which removes the repeated server name and splits the datetime column into separate columns, to improve readability.
	
	The following functions allow the querying and updating of the flag that determines whether the **perfmon** data is manipulated or not:
	
	`set rc=$$setperfmonpostproc^SystemPerformance(<onoroff>)`
	
	where `onoroff` can be 1 (on) or 0 (off), or the non-case-sensitive words `"on"` or `"off"`.
	
	A return code of 1 indicates successful update of the flag, 0 indicates a failed update, and `-1` indicates a non-Windows platform.
	
	To determine the current format (raw or processed):
	
	`set status=$$getperfmonpostproc^SystemPerformance()`
	
	A return code of 1 indicates processed format, 0 indicates raw format.
	
	In addition, the current status of the flag is reported prior to the profile menu display in the interactive run of **^SystemPerformance**.
	
	By default, **perfmon** monitors the counter definitions specified in the default pbctrs.txt file. To monitor previously defined **perfmon** counters, import the definition into **^SystemPerformance** using:
	
	`write $$importctrs^SystemPerformance(`*`WindowsCtrName`*
	`[,`*`SystemPerformanceCtrName`*` [,`*`SystemPerformanceFileName`*`]])`
	
	A return code of 0 indicates success and a negative number followed by a reason string indicates failure. Duplicate SystemPerformance counter names are not allowed. If necessary, **^SystemPerformance** generates both the internal counter name and file name.
	
	To change the default SystemPerformance counter definition to an exiting definition, use:
	
	`write $$setctrdefault(`*`SystemPerformanceCtrName`*`)`
	
	Return code of 1 indicates success and 0 followed by a reason string indicates failure. If an invalid counter is specified, the builtin defaualt is set.
	
	To reset the default SystemPerformance counter definition, use:
	
	`do clrctrdefault^SystemPerformance()`
	
	To associate a specific SystemPerformance counter definition with an existing profile, use:
	
	`write $$addctrtoprofile(`*`ProfileName`*`,`*`SystemPerformanceCtrName`*`)`
	
	Return code of 1 indicates success and 0 followed by a reason string indicates failure. If either the profile or the counter definition do not exist, the command is not run.

Section	Description
Profile *	Information about the **^SystemPerformance** profile that created this log.
tasklist	Four outputs of the **tasklist -V** command, taken at even intervals over the course of the run. The **tasklist -V** command provides a list of all processes running on the system.
Windows info *	Output from the **systeminfo** command, including the Windows version (excluding hotfix information) and hardware information; for example, processor count, memory installed, and memory used.

Table 13–2: InterSystems IRIS Performance Data Report for Apple macOS Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
Configuration *	InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.
irisstat -c	Four samples taken at even intervals over the course of the run using the command **irisstat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all IRIS.DAT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility.

Section	Description
irisstat -D	Eight samples taken at even intervals over the course of the run using the command **irisstat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility. For information about monitoring block collisions using the **^BLKCOL** utility, see Monitoring Block Collisions Using ^BLKCOL.
ipcs *	Interprocess communication configuration information, including shared memory, semaphores, and message queues; output from **ipcs -a** command.
license *	InterSystems IRIS license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
macOS Info *	OS version and hardware information. Output from the **sw_vers**, **uname -a**, **mount**, and **netstat** commands.
mgstat	InterSystems IRIS-specific data taken over the course of the run using the **^mgstat** utility. See Monitoring Performance Using ^mgstat.
Profile *	Information about the **^SystemPerformance** profile that created this log.
ps:	Four samples taken at even intervals over the course of the run using the command **ps -eflv**.
sar -d	Disk (block) device throughput and latency statistics.
sar -g	Page out rates.
sar -n DEV	Network device throughput.
sar -n EDEV	Network device error rates.
sar -p	Page in and page fault rates.
sar -u	CPU usage statistics.
sysctl -a *	Kernel and system parameter settings.
vm_stat *	memory page information.

Table 13–3: InterSystems IRIS Performance Data Report for IBM AIX® Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.
AIX info *	Output from the **oslevel**. **uname -a**, **prtconf**, and **lspv** commands

Section	Description
Configuration *	InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.
cpu type *	Information on processors installed and whether or not SMT is enabled; output from **lsattr -El proc0**.
irisstat -c	Four samples taken at even intervals over the course of the run using the command **irisstat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all IRIS.DAT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility.
irisstat -D	Eight samples taken at even intervals over the course of the run using the command **irisstat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility. For information about monitoring block collisions using the **^BLKCOL** utility, see Monitoring Block Collisions Using ^BLKCOL.
df -k *	Information about mounted file systems, including mount points, logical volumes, and free space; output from **df -k** command.
filesystems *	Current /etc/filesystems file.
ioo -a *	Current values of I/O tunable parameters; output from **ioo -a** command. Included *only* if the user initiating the **^SystemPerformance** profile run has root access.

Section	Description
iostat -DIT	Long listing of extended disk/device statistics with sample time for IBM AIX® 5.3 and newer; output from **iostat -DIT** command. Information varies for releases before IBM AIX® 5.3.
ipcs *	Interprocess communication configuration information, including shared memory, semaphores, and message queues; output from **ipcs -a** command.
license *	InterSystems IRIS license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
mount *	Information on all file systems and their mount options.
mgstat	InterSystems IRIS-specific data taken over the course of the run using the **^mgstat** utility. See Monitoring Performance Using ^mgstat.
Profile *	Information about the **^SystemPerformance** profile that created this log.
ps:	Four samples taken at even intervals over the course of the run using the command **ps aux**.
sar -d	Included *only* if the user initiating the**^SystemPerformance** profile run has root access and /usr/sbin/sar exists.
sar -r	Included *only* if the user initiating the**^SystemPerformance** profile run has root access and /usr/sbin/sar exists.
sar -u	CPU statistics that includes micropartitioning information if used. Included *only* if the user initiating the**^SystemPerformance** profile run has root access and /usr/sbin/sar exists.
vmo –a	Current values of virtual memory tunable parameters; output from **vmo -a** command. Included *only* if the user initiating the**^SystemPerformance** profile run has root access.
vmstat -s *	Absolute counts of virtual memory statistics, including total page ins and page outs.
vmstat -t	Virtual memory and CPU (paging, queuing, and CPU) statistics with timestamps.
vmstat -v *	Samples virtual memory statistics, including free pages, **pbuf** usage, and **fsbuf** usage.

Table 13–4: InterSystems IRIS Performance Data Report for Linux Platforms

Section	Description
%SS	Four samples taken over the course of the run using the **ALL^%SS** command.

Section	Description
Configuration *	InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.
cpf file *	A copy of the currently active configuration file.
irisstat -c	Four samples taken at even intervals over the course of the run using the command **irisstat cache -p-1 -c-1 -e1 -m8 -n2 -N127**. Following is a brief description of each argument: • **-p-1**: samples the process table to include process and global state information. • **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics. • **-e1**: the SYSLOG error table. • **-m8**: the file table, which includes all IRIS.DAT files and their attributes. • **-n2**: the network structures table, including local-to-remote database mappings. • **-N127**: ECP statistics for both client and server connections. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility.
irisstat -D	Eight samples taken at even intervals over the course of the run using the command **irisstat cache --f1 -D10,100**. Following is a brief description of each argument: • **-fl**: basic flags. • **-D10,100**: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds. For more information about the **irisstat** utility, see Monitoring InterSystems IRIS Using the irisstat Utility. For information about monitoring block collisions using the **^BLKCOL** utility, see Monitoring Block Collisions Using ^BLKCOL.
df -m *	Information about mounted file systems, including mount points, logical volumes, and free space; output from **df -m** command.
free -m	Memory usage statistics in MB (**-m**).
iostat	CPU and disk throughput.
license *	InterSystems IRIS license usage information using **Decode^%LICENSE** and **counts^%LICENSE**.
mgstat	InterSystems IRIS-specific data taken over the course of the run using the **^mgstat** utility. See Monitoring Performance Using ^mgstat.
Profile *	Information about the **^SystemPerformance** profile that created this log.

Section	Description
ps:	Four samples taken at even intervals over the course of the run using the command **ps -efly**.
sar -d	Disk (block) device throughput and latency statistics.
sar -u	CPU usage statistics include **iowait** percentage.
vmstat -n	CPU, queuing, paging statistics. Only one header is printed (**-n**) .
CPU *	Information gathered from **lscpu** and **/proc/cpuinfo**
Linux info *	General OS and hardware information; includes output from uname -a, lsb_release -a, id, and ulimit -a commands as well as information gathered from /etc/issue.net, /proc/partitions, and /dev/mapper.
ipcs *	Interprocess communication configuration information, including shared memory, semaphores, and message queues; output from ipcs -a command.
mount *	Information on all file systems and their mount options.
fdisk -l *	Partition tables for all devices mentioned in /proc/partitions.Included *only* if the user initiating the ^*SystemPerformance* profile run has root access.
ifconfig *	Status information of currently active network interfaces.
sysctl -a *	Kernel and system parameter settings.

13.10 See Also

- Using the Diagnostic Report
- WRC distribution site (**Tools**)

14

Monitoring Performance Using ^mgstat

This topic describes the **^mgstat** utility, a tool for collecting basic performance data.

Note: This utility may be updated between releases. Contact the InterSystems Worldwide Response Center (WRC) for information about downloading newmgstat.xml from ftp://ftp.intersys.com/pub/performance/.

14.1 Running ^mgstat

You must call **^mgstat** in the **%SYS** namespace. You can use the following positional arguments:

1. *sample time* — This argument specifies the frequency (in seconds) for sampling counters. If not specified, the default is 2 seconds.

   Note: If you specify a *sample time* greater than 10 seconds, **^mgstat** reduces it to 10 seconds. See the *number of samples* argument in this table.

2. *number of samples* — This argument specifies the maximum number of samples to be obtained. If not specified, the default is 10 iterations.

   Note: If **^mgstat** reduces the *sample time*, it increases the specified *number of samples* to ensure that the duration (*sample time* * *number of samples*) of the run is effectively the same as it would have been if none of the arguments were modified.

3. *filename* — This arguments specifies the filename for the `.mgst` file that **^mgstat** generates, relative to *install-dir*\mgr. If not specified, the default filename is *ServerName_InstanceName_Date_Time*`.mgst`.
4. *page length* — If you run **^mgstat** interactively, this argument specifies the number of lines to display before the header rows are repeated. The default is 0, which displays the header once at the beginning of the page; if you specify a value less than 5 lines (other than 0), **^mgstat** increases it to 5.

   Note: This argument is ignored when you run **^mgstat** as a background job.

For example, if running **^mgstat** as a background job, to specify that file samples be obtained every 5 seconds until 17280 samplings are obtained (in the Terminal, in the %SYS namespace), enter the following:

```
%SYS>JOB ^mgstat(5,17280)
```

Alternatively, if running **^mgstat** interactively, to specify the same samplings redisplay the headings after each 10 rows of data, enter the following:

```
%SYS>DO ^mgstat(5,17280,,10)
```

By default **^mgstat** generates a filename based on the *server name*, *configuration name*, and *date and time*, with the `mgst` extension, which is recognized by an analyzer tool written in Microsoft Excel that aids graphing of the data. By default, the file is located in the *install-dir*\mgr directory of the InterSystems IRIS® data platform instance; however, if the output directory has been changed through the **^SystemPerformance** utility (see Change Output Directory), **^mgstat** uses that output directory.

Note: The `.mgst` file is also generated when you run the **^SystemPerformance** utility and included in the HTML performance report (see Monitoring Performance Using ^SystemPerformance).

To ensure minimal impact on system performance, the **^mgstat** utility extracts various counter information from shared memory. If the utility is running and an apparent performance issue occurs, data is available to help you investigate the problem; for assistance with your analysis, contact the InterSystems Worldwide Response Center (WRC), which can provide tasks that automate both the running of **^mgstat** and the purging of files.

14.2 Data Provided by ^mgstat

Most of the reported data is averaged in per-second values, except as noted in the table below. The generated output file is in a readable, comma-separated value (CSV) format, which is more easily interpreted with a spreadsheet tool such as Microsoft Excel. The first line of the file is a header line which includes the filename and the utility version, as well information about buffer allocation and the version of the product being monitored. The number of columns of data depends on the version of the product: the first two columns are the date and time; the remaining columns are:

Column	Description	Notes
Glorefs	Global references (database accesses). Indicates the amount of work that is occurring on behalf of the current workload; although global references consume CPU time, they do not always require physical reads because of the buffer pool.	
RemGrefs *	Remote global references (database accesses). Indicates the number of global references that are generated on behalf of distributed cache cluster application servers.	
GRratio	Ratio of global references to remote global references.	
PhyRds	Physical reads from disk. A high number of physical reads may indicate a performance problem; you can improve the performance by increasing the number of database (global) buffers.	
Rdratio	Ratio of logical block reads to physical block reads, but zero if physical block reads is zero.	
Gloupds	Global updates (sets or kills).	

Column	Description	Notes
RemGupds *	Remote global updates.	
Rourefs	Routine references (includes **tag^routine**).	
RemRrefs *	Remote routine references.	
RouLaS	Routine loads and saves (fetch from or save to disk). A high number of routine loads/saves may indicate a performance problem; you can improve the performance by increasing the number of routine buffers.	
RemRLaS *	Remote routine loads and saves.	
PhyWrs	Physical writes to disk.	
WDQsz	write daemon Queue size (in blocks).	Not per second.
WDtmpq	Updated blocks in IRISTEMP.	Not per second.
WDphase	Phase of the write daemon. The most common phases are: • 0: Idle (WD is not running) • 5: WD is updating the Write Image Journal (WIJ) file. • 7: WD is committing WIJ and Journal. • 8: Databases are being updated.	Not per second.
Wijwri	Number of 256-KB blocks written to the WIJ. This is non-zero when the WD is writing data to the WIJ.	
RouCMs	Number of Routine Cache Misses.	
Jrnwrts	Number of blocks written to journals.	
GblSz	Number of seizes on the global resource; see Considering Seizes, ASeizes, and NSeizes.	
pGblNsz	Percentage of NSeizes on the global resource; see Considering Seizes, ASeizes, and NSeizes.	
pGblAsz	Percentage of **ASeizes** on the global resource; see Considering Seizes, ASeizes, and NSeizes.	
RouSz	Number of seizes on the routine resource; see Considering Seizes, ASeizes, and NSeizes.	
pRouAsz	Percentage of **ASeizes** on the routine resource; see Considering Seizes, ASeizes, and NSeizes.	

Column	Description	Notes
ObjSz	Number of seizes on the object resource; see Considering Seizes, ASeizes, and NSeizes.	
pObjAsz	Percentage of **ASeizes** on the object resource; see Considering Seizes, ASeizes, and NSeizes.	
ActECP *	Number of active ECP connections.	Not per second.
Addblk *	Number of blocks added to ECP Client's cache.	
PrgBufL *	Number of blocks purged from ECP Client's cache due to global buffer shortage (on the ECP Client). A high number may indicate a performance problem on the ECP client; you can improve performance by increasing the number of global buffers on the ECP Client.	
PrgSrvR *	Number of blocks purged from ECP Client's cache by ECP server.	
BytSnt *	Number of bytes sent as an ECP Client.	
BytRcd *	Number of bytes received as an ECP Client.	
WDPass	WD cycle since startup.	Not per second.
IJUCnt	Number of jobs that WD is waiting for to continue this cycle.	Not per second.
IJULock	Indicates whether or not the IJULock flag is set . If IJULock is set, all updates are locked out while the WD finalizes the write cycle.	Not per second.
PPGrefs	The count of all process private global accesses.	
PPGupds	The count of all process private global updates.	

* 0 is displayed unless this is an ECP configuration.

14.3 Considering Seizes, ASeizes, and NSeizes

Note: Seize statistics are included if the underlying system is collecting them; more information can be found in the Config.Miscellaneous class; see the CollectResourceStats property.

A *Seize* occurs whenever a job needs exclusive access on a given resource to guarantee that an update occurs without interference from other processes. If the Seize is not immediately satisfied, the update is postponed until it is satisfied.

On a single-CPU system, the process immediately hibernates (because it cannot do anything until the process holding the resource relinquishes it, which does not occur until after its own update completes).

On a multiple-CPU system, the process enters a holding loop in the "hope" that it will gain the resource in a reasonable time, thus avoiding the expense of hibernating. If the process gets access to the resource during the hold loop, the loop immediately exits and the process continues with its update; upon completing the update, the process relinquishes the resource for other processes that may be waiting for it; this is an *Aseize*. If, at the end of the hold loop, the resource is still held by another process, the process continues to hibernate and wait to be woken up when the resource is released; this is an *Nseize*.

Nseizes are a natural consequence of running multiple processes on a single-CPU system; Aseizes are a natural consequence of running multiple processes on a multi-CPU system. The difference is that Nseizes incur system, or privileged, CPU time because the operating system must change the context of the running process, whereas an Aseize incurs user time on the CPU because it continues to run until the resource is gained and released, or until it gives up and hibernates. In general, on multi-CPU systems it is more expensive for the operating system to do the context switch than to loop a few times to avoid this operation because there is both CPU overhead and memory latency associated with context switching on multi-CPU systems.

14.4 See Also

- Monitoring Performance Using ^SystemPerformance

15

History Monitor

The History Monitor maintains a historical database of performance and system usage metrics. Its primary purposes are to:

- Provide a performance baseline and help with analysis of performance issues.
- Help analyze system usage over time for capacity planning.

This database is defined in the SYS.History class package and kept in the %SYS namespace. All of the details of the database structure are published there and the data is accessible through SQL or the normal persistent object access. The class documentation in SYS.History also contains descriptions of all the individual properties, methods and queries that are available.

The data is generally organized into performance (see SYS.History.Performance) and system usage (see SYS.History.SystemUsage) data. The performance metrics are intended to be sampled at short intervals (by default, 30 seconds), and the system usage data at longer intervals (by default, 5 minutes). At the beginning of each day, the individual interval samples are summarized into hourly and daily tables as averages, maximums, minimums, standard deviation, medians and totals. You can select which, if any, of the summary functions are kept for each metric class. The interval and hourly data may be purged automatically after a defined number of days (by default, seven (7) days and 60 days, respectively); the daily summary is intended for long-term analysis and can be purged manually.

15.1 Base Metrics

All of the collected metrics are defined in four %SerialObject classes in SYS.History. These same classes are used as the basis for the Interval, Hourly, and Daily databases, so all of the properties are defined as %Numeric types to allow for decimal values in the summaries.

The performance related metrics are defined in:

- SYS.History.Performance — The properties in this class are general performance metrics like global references and routine calls.

  Note: These properties are all "counter" types and the interval data is collected as deltas, which represent the change in the counter over the last interval. When this data is summarized into hourly and daily values, the data is normalized to per-second rates

- SYS.History.WriteDaemon — The properties in this class describe the performance of write daemon cycles. The system automatically keeps track of the last 20 write daemon cycles, and the History Monitor stores the data for the cycles that occurred in each interval. Typically, there are multiple cycles within each interval.

The system usage metrics are defined in:

- SYS.History.SystemUsage — The properties in this class track how busy the system is but do not tend to change as quickly or dramatically as the performance data, such as the number of processes in InterSystems IRIS® data platform and license information.
- SYS.History.Database — This class tracks the database growth, file size and free space, for each local database.

15.2 Collecting Data

To begin collecting data, you must do the following:

- Use the System Monitor **^%SYSMONMGR** utility in the %SYS namespace to activate the desired monitor classes (%Monitor.System.HistoryPerf or %Monitor.System.HistorySys) in Application Monitor (which is part of System Monitor). These classes are registered in the %SYS namespace by default.
- Restart System Monitor in the %SYS namespace.

See Using ^%SYSMONMGR to Manage Application Monitor and Start/Stop System Monitor for information about these procedures.

The detailed interval collection of data is defined in two persistent classes:

- SYS.History.PerfData — Includes the performance and write daemon classes as embedded objects.
- SYS.History.SysData — Includes the system usage and database classes.

The corresponding %Monitor classes must be activated in Application Monitor in order to collect data and build the history data:

- %Monitor.System.HistoryPerf — Collects instances of SYS.History.PerfData samples.
- %Monitor.System.HistorySys — Collects SYS.History.SysData samples.

System Monitor, including Application Monitor, starts by default in the %SYS namespace when the InterSystems IRIS instance starts. You can configured other startup namespaces, however. The %Monitor classes are provided by default only in %SYS, but can be added to other configured startup namespaces using **^%SYSMONMGR**.

15.3 Summaries

The %Monitor.System.HistoryPerf and %Monitor.System.HistorySys classes, as executed by Application Monitor, also create the hourly and daily summaries at the end of each day. The summaries are defined as the persistent classes SYS.History.Hourly and SYS.History.Daily; they include all four of the base classes as embedded objects.

For each metric property, the system may calculate the average, maximum (high-water mark), standard deviation, minimum, median, or total for each hour and for the whole day. The summary functions are selectable (or may be disabled) for each base class (SYS.History.Performance, SYS.History.WriteDaemon, SYS.History.SystemUsage, or SYS.History.Database) and for each summary period class, using the **SetSummary()** method of each of the base classes. By default, the History Monitor calculates average, maximum and standard deviation for each class for both hourly and daily summaries.

Note: The counter properties of the SYS.History.Performance class are normalized to per second rates for these calculations (except Total).

Purging Data

After creating the summaries, Application Monitor automatically purges the interval and hourly databases. The default is seven (7) days for interval data and 60 days for hourly data, but these may be changed using the **SetPurge()** method in SYS.History.PerfData and SYS.History.Hourly classes. The SYS.History.Daily data is not automatically purged, but can be done manually using the **SYS.History.Daily:Purge()** method.

15.4 Accessing the Data

Since the database is defined as persistent classes, the data is available using standard SQL or persistent object access. Using the SQL browser in the Management Portal is a quick and easy way to see the various SQL schemas/tables that are created, including the individual property values.

There are several basic queries implemented in each of the persistent classes in SYS.History (SYS.History.PerfData, SYS.History.SysData, SYS.History.Hourly, and SYS.History.Daily) that can be used to access the individual tables for a date range; for more information about the queries, see the class reference documentation.

The are also several **Export()** methods provided for each persistent class so that the individual tables can be exported to files in CSV format, suitable for use with a spreadsheet such as Microsoft Excel. In particular, the **SYS.History.PerfData:Export()** method creates a file that is very similar in format to that created by the **^mgstat** utility (for more information, see Monitoring Performance Using ^mgstat).

15.5 Adding User-Defined Metrics

You can add user-defined metrics to the History Monitor (SYS.History package):

1. Create a class, or multiple classes, that inherit from SYS.History.Adaptor and add %Numeric properties to define the metrics.

   Note: User-written classes must be in the %SYS namespace, and should begin with "Z" or "z" to prevent naming conflicts with system classes and problems during upgrades.

2. Code the **Sample()** method to instantiate the class and provide periodic values for each property. This method is called when the interval data is collected.

3. When you compile your class, it is added as an embedded object to an interval persistent class in SYS.History. You can choose where and when it is collected using the *INTERVAL* parameter provided in SYS.History.Adaptor class. This selects which interval class it is added to and which %Monitor class does the collection, as shown in the following table:

INTERVAL Selected	Interval Class Used	%Monitor Class Used
"User" (default)	SYS.History.User	%Monitor.System.HistoryUser
"UserPerf"	SYS.History.UserPerf	%Monitor.System.HistoryPerf
"UserSys"	SYS.History.UserSys	%Monitor.System.HistorySys

   Selecting "UserPerf" or "UserSys" lets you collect data at the same interval and with the same timestamp as SYS.History.PerfData or SYS.History.SysData, which makes it easier to correlate your data with the system data. "User" gives you a choice of a third (unrelated) time interval.

Note: There are several parameters in the SYS.History.Adaptor class that provide options for how properties are collected and summarized; for more information, see the SYS.History.Adaptor class reference documentation.

4. User-defined classes are also added as embedded objects to the SYS.History.UserHourly and SYS.History.UserDaily summary classes. The user-defined metrics are summarized and automatically purged just like the system metrics.

   Important: User-defined metric classes become embedded objects in persistent data. You should not change definitions after data collection has started: deleting objects can result in orphaned data; re-defining existing classes or properties can cause already stored data to be misinterpreted.

   However, because of the *schema evolution* feature, you can safely add new objects and properties. See Schema Evolution.

16

Monitoring Block Collisions Using ^BLKCOL

A block collision occurs when a process is forced to wait for access to a block. Excessive block collisions slow application performance. This page describes how to monitor block collisions by using the **^BLKCOL** utility.

16.1 Using ^BLKCOL

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

Note: The **irisstat -D** option, as described in Running irisstat with Options, also samples block collisions, but identifies only the blocks involved.

The output of **irisstat -D** is included in the reports generated by the **^SystemPerformance** utility, as described in Monitoring Performance Using ^SystemPerformance.

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

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

16.2 ^BLKCOL Ouput

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

```
%SYS>do ^BLKCOL

Block Collision Analysis

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

Sampling ... (any key to interrupt)


625 block collisions in 735 samples.

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

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

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

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

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

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

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

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

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

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

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

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

Block Collision Analysis

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

Sampling ... (any key to interrupt)


571 block collisions in 768 samples.


Sorted by routine/line that waits for block ownership
_____________________________________________________

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

Sorted by routine that owns the block
_____________________________________

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

17

Monitoring Processes Using ^PERFSAMPLE

This topic describes the **^PERFSAMPLE** utility, a tool for analyzing InterSystems IRIS® data platform processes.

The utility performs high frequency sampling of selected processes on the system, and analyzes the data to determine where the processes are spending most of their time. It presents an easily navigable breakdown of the sampled activity, which can provide insights into your system. For example, you may discover application bottlenecks by checking ECP requests, or identify overall system bottlenecks by reviewing the types of wait events.

To get started, run **^PERFSAMPLE** in the %SYS namespace:

Terminal

```
USER>set $namespace = "%SYS"

%SYS>do ^PERFSAMPLE
```

17.1 Collecting Samples

The following message appears as soon as you run **^PERFSAMPLE**:

Terminal

```
This utility performs high frequency sampling of processes on the system,
analyzing and counting data points in different ways to understand where
processes are spending most of their time. On ECP Data Servers, this also
offers sampling of the current request being processed and the states of
the ECPSvrW daemons doing the processing.

1) Sample Local Process Activity
2) Sample ECP Server Requests

Option?
```

If the instance has no incoming ECP connections from ECP clients (application servers), `option 1` above is automatically selected.

You are then prompted to enter the following information:

1. What processes or ECP connections to sample.
2. Whether to ignore samples where the process is in any of the following states: `READ`, `READW`, `EVTW`, `HANG`, `SLCT`, `SLCTW`, and `RUNW`. When sampling ECP connections, only events where the `ECPSvrW` process is non-idle are recorded.

Selecting YES (the default) reduces the number of events that **^PERFSAMPLE** records. When monitoring many processes, this speeds analysis and uses less memory.

3. Number of samples to collect per second.
4. Total number of seconds to collect samples.

In the Terminal, the prompts look like:

Terminal

```
Enter a list of PIDs, * for all, or ? for ^%SS display: *
Ignore samples where the process appears idle (READ, HANG, etc)?
  Yes =>
Sample rate per second: 1000 =>
Number of seconds to sample: 30 =>
```

17.2 Examining and Analyzing Samples

After collecting the samples, you may view an analysis. An analysis is a summary of one or more *dimensions*, or components, of the sampled processes. That is, an analysis sorts the sampled information according to the selected dimensions.

This section includes the following:

- An example of using a predefined analyses.
- Information about creating a custom analyses.
- A description of the available analysis dimensions.

Use the following keys to navigate within the analyzer:

Key Input	Navigation Action
Up arrow or **U**	Move the selector up
Down arrow or **D**	Move the selector down
Right arrow or **Enter**	Select the current item
Left arrow or **Backspace**	Go back to the previous level
C	Cycles through the following count displays: • Percent of total • Raw count • Percent of the current subset • (If multiple jobs are sampled) Average number of jobs found concurrently in this state
N or **CTRL-D**	Next page (if multiple pages)
P or **CTRL-U**	Previous page (if multiple pages)
Q	Quit **^PERFSAMPLE**

The main landing page looks like:

Terminal

```
— PERFSAMPLE for Local Process Activity. 1.710949s at 11/17/2020 15:58:31
28479 samples | CPULoad* 0.91
Multiple jobs included: 1290 samples per job
----------------------------'?' for help----------------------------
Select an analysis to view:
New Analysis (press '+' any time)
Using CPU? -> PID -> Process State
Using CPU? -> Routine -> Namespace -> Process State
Process State -> Routine -> PID
Kernel Wait State -> Routine -> PID
```

17.2.1 Predefined Analysis Example

Below is an example of an analysis that begins with the `Process State` dimension.

In this example, **^PERFSAMPLE** found `76755` samples of processes in a sample-able state (non-idle if the option to ignore idle was selected) out of `319994` total samples:

Terminal

```
– PERFSAMPLE for Local Process Activity. 3.89s at 11/17/2020 16:59:59
76755 events in 319994 samples [24.0 %-total] | CPULoad* 8.22
Multiple jobs included: 2191 samples per job
----------------------------'?' for help----------------------------
Process State [24.0 %-total]
GGET [8.46 %-total]
RUN [5.88 %-total]
GDEF [3.16 %-total]
GSETW [1.63 %-total]
BSETW [1.21 %-total]
GDEFW [1.18 %-total]
GGETW [0.931 %-total]
SEMW [0.685 %-total]
GSET [0.311 %-total]
LOCKW [0.144 %-total]
LOCK [0.0644 %-total]
INCRW [0.0641 %-total]
BSET [0.0513 %-total]
```

Initially, the values appear as a percent of the total number of samples. The most common `Process State` value sampled in this case was `GGET`, which represents 8.46% of the total `319994` samples.

Pressing **c** cycles through how this count is displayed. For example, you can display the above information as a raw count of samples:

Terminal

```
Process State [76755]
> GGET [27083]
RUN [18823]
GDEF [10121]
```

You can also view the information as a percent of qualifying samples (in this case, samples that had a non-idle `Process State`):

Terminal

```
Process State [24.0 %-total]
> GGET [35.3 %-subset]
RUN [24.5 %-subset]
GDEF [13.2 %-subset]
```

Finally, you can view the average number of jobs concurrently each state:

Terminal

```
Process State [24.0 %-total]
GGET [12.4 jobs]
RUN [8.59 jobs]
GDEF [4.62 jobs]
```

Selecting GGET with the **Right Arrow** key moves to the next dimension, ordering the values of that dimension for samples where the first dimension had value GGET. You can navigate freely between the dimensions using the arrow keys.

17.2.2 Creating a Custom Analysis

Select the New Analysis option from the main landing page to create a custom analysis. You can also create an custom analysis using one of these shortcuts:

Key Input	Shortcut
+ key	Add a dimension to the current analysis (when in an analysis)
* key	Begin a new analysis with the current item as the first dimension

Adding a new analysis brings you to the following screen:

Terminal

```
New Analysis:

Specify a comma-delimited list of dimensions upon which to analyze samples.
For example, "state,ns,rou" means first count each unique state the sampled
processes were in; then for each state, count the namespace from the samples
in that state; and finally for each state->namespace pair, count each unique
routine name. In other words, report on routines by namespace by state.

The following dimensions are available:
cpu - Using CPU? (process state indicates expected CPU use)
ns - Namespace (current namespace)
pid - PID (process ID)
rou - Routine (name of current routine)
state - Process State (process state string, e.g. GSETW)
trace - Kernel Trace (alternative to 'state' w/ kernel-level detail)
waits - Kernel Wait State (kernel-level condition that delayed the process)
wtrace - Reverse Kernel Trace (revese kernel trace, stop at any wait state)

Enter dimension list:
```

From here, enter the list of dimensions you would like to analyze as described by the prompt. Once you press **Enter**, you may navigate the analysis as described above.

17.2.3 Analysis Dimensions

The dimensions for analyses are described within the **^PERFSAMPLE** tool. This section provides some additional information.

- cpu - Using CPU? (process state indicates expected CPU use)

  Note: The yes or no value for cpu is not a true measure of on-cpu time, but an estimate. **^PERFSAMPLE** infers CPU use from the process state, and InterSystems IRIS state tracking may not directly correlate to CPU use.

  If the process is waiting for the OS scheduler to make the CPU available to it as a result of (instantaneous or persistent) over-utilization of the CPU, this can also lead to inaccuracy in cpu.

- ns - Namespace (current namespace)

- `pid` - PID (process ID)
- `rou` - Routine (name of current routine)
- `state` - Process State (process state string, e.g. GSETW)
- `waits` - Kernel Wait State (kernel-level condition that delayed the process). See the following section for more information.

In general, the following dimensions are only useful when troubleshooting with the InterSystems Worldwide Response Center (WRC):

- `trace` - Kernel Trace (alternative to 'state' w/ kernel-level detail)
- `wtrace` - Reverse Kernel Trace (revese kernel trace, stop at any wait state)

The `trace` and `wtrace` dimensions have a hierarchical organization. Selecting an ancestor, denoted with an ellipsis (`...`), moves down a hierarchy level. Selecting a non-ancestral item goes to the next dimension of analysis. The **h** key toggles between this hierarchical view and a flattened view. Pressing the **a** key on an ancestor aggregates subsequent dimensions for all its descendants.

17.2.3.1 The waits Dimension

The `waits` dimension is null if the process was not found to be waiting on anything internal to the InterSystems IRIS kernel. A non-null value indicates a condition which required the process to wait (to block internally).

It's important to note that these are internal conditions leading to the process waiting outside of the application's direct control. As such, waiting due to a conflicting LOCK command, a $SYSTEM.Event, and the like do not count here.

Nonetheless, many of the values, particularly the more common ones, are things that the application can influence indirectly. For example, if samples show that a key application process is often waiting for `diskio`, this indicates that the process is waiting to read database blocks from disk and could possibly benefit from parallelization, prefetching, or more database cache. Similarly, a process that samples show is often waiting on `inusebufwt` is encountering database block collisions that may need investigation at the application level (with the help of the ^BLKCOL utility). The values in this dimension take on the following mnemonic values, which are subject to change in the future:

- `diskio`: waiting for database physical block read
- `inusebufwt`: waiting due to block collision (^BLKCOL utility may help identify application cause)
- `expand`: waiting for database expansion
- `ecpwait`: waiting for an answer from the ECP server
- `jrniowait`: no space in journal buffers, waiting for journal I/O
- `jrnsyncblk`: waiting for journal data to be committed
- `jrnlckwait`: waiting to access journal buffer
- `mirrorwait`: waiting for active backup mirror member
- `mirrortrouble`: blocked due to mirror trouble state
- `globwait`: waiting because of an internal condition blocking global updates
- `aiowait`: waiting for asynchronous disk I/O to complete
- `wdqwait`: waiting for a write cycle to complete
- `freebuf`: global buffers are completely exhausted and waiting for database writes
- `gfownwait`: access to database is blocked

- `resenqXYZ`: waiting on an internal resource XYZ

Note: While many of these correspond to a canonical process state that includes the `W` letter flag (e.g. `GSETW`, `GORDW`, etc) and not all do – `diskio` is a very common example – and not all cases of the `W` state flag have an internal reason reflected here (e.g. `LOCKW` as mentioned above).

17.3 Save Analysis

After viewing the samples, you may save them for future analysis. To do so, press the **Left Arrow** from the analysis landing page. This returns you to the initial Collecting Samples page, but with the additional option to `Save Samples to File`. Select this option and enter the desired filename, such as `perfsample001.txt`. **^PERFSAMPLE** saves the file to the *install-dir*\mgr directory.

To open a saved analysis, launch **^PERFSAMPLE** using the LOAD tag and specify the file to open. For example:

Terminal

```
USER>set $namespace = "%SYS"
%SYS>do LOAD^PERFSAMPLE
File: C:\MyIRIS\mgr\perfsample001.txt
```

^PERFSAMPLE loads the file, allowing you to analyze and examine the saved samples.

17.4 See Also

- Controlling InterSystems IRIS Processes (general information on InterSystems processes)
- ^BLKCOL utility (for monitoring block collisions, which occur when a process is forced to wait for access to a block)

A

Monitoring InterSystems IRIS Using SNMP

This topic describes the interface between InterSystems IRIS® data platform 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 InterSystems IRIS: 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 InterSystems IRIS uses to define its management information for use by other applications.

A.1 Using SNMP with InterSystems IRIS

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 InterSystems IRIS, this is the ISC-IRIS.mib file, located in the *install-dir*\SNMP directory.

A.2 InterSystems IRIS as a Subagent

The SNMP client connects to the SNMP agent which is listening on a well-known address, port `161`. Since the client expects to connect on this particular port, there can only be one SNMP agent on a computer system. To allow access to multiple applications on the system, developers can implement *master agents*, which may be extended or connected to multiple subagents. InterSystems has implemented the InterSystems IRIS SNMP interface as a subagent, designed to communicate through an SNMP master agent.

Most operating systems that InterSystems IRIS 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. InterSystems IRIS implements its subagent using the Agent Extensibility (AgentX) protocol, an IETF-proposed standard as described in RFC 2741.

Some of the standard SNMP master agents support AgentX. If the SNMP master agent supplied by an operating system is not AgentX-compatible, you can replace it with the public domain Net-SNMP agent.

Note: The exception is the Windows standard agent which does not support AgentX and for which the Net-SNMP version may not be adequate. For this exception, InterSystems supplies a Windows extension agent DLL, iscsnmp.dll, which handles the connection between the standard Windows SNMP service extension API and the InterSystems IRIS AgentX server.

A.3 Managing SNMP in InterSystems IRIS

Since SNMP is a standard protocol, the management of the InterSystems IRIS subagent is minimal. The most important task is to verify that the SNMP master agent on the system is compatible with the Agent Extensibility (AgentX) protocol (see InterSystems IRIS as a Subagent) and it is active and listening for connections on the standard AgentX TCP port `705`. On Windows systems, the system automatically installs a DLL to connect with the standard Windows SNMP service. Verify that the Windows SNMP service is installed and started either automatically or manually.

Important: Some SNMP master agents, notably Net-SNMP on Linux, do not enable AgentX by default and do not use TCP port `705` by default once they are enabled. For Net-SNMP you must modify the snmpd.conf file to enable communications with the InterSystems IRIS subagent. Recent versions of Net-SNMP also implement VACM (View-based Access Control Model) security and, by default, only allow access to the mib-2.system subtree; the InterSystems IRIS subagent starts and runs without error, but no SNMP requests are forwarded to InterSystems IRIS. You must expand the "views" defined in snmpd.conf to include the InterSystems IRIS MIB subtree.

Next, enable the monitoring service using the following steps:

1. Navigate to the **Services** page in the Management Portal (**System Administration** > **Security** > **Services**).
2. Click the **`%Service_Monitor`** service.
3. Select the **Service enabled** check box and click **Save**.
4. Return to the list of services page and ensure that the **`%Service_Monitor`** service is enabled.

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

1. Navigate to the **Monitor Settings** page in the Management Portal (**System Administration** > **Configuration** > **Additional Settings** > **Monitor**).
2. Select `Yes` for the **Start SNMP Agent at System Startup** setting and click **Save**.
3. When you edit this setting, the InterSystems IRIS end of the SNMP interface immediately stops and starts.

You can also start and stop the InterSystems IRIS SNMP subagent manually or programmatically using the **^SNMP** routine:

```
Do start^SNMP(<port>,<timeout>)
Do stop^SNMP
```

where <*port*> is the TCP port for the connection (default is `705`) and <*timeout*> is the TCP port read timeout value (default is 20 seconds). Until the <*timeout*> value is reached, InterSystems IRIS logs any problems encountered while establishing a connection or answering requests in the SNMP.LOG file in the *install-dir*\mgr directory.

Note: When the SNMP master agent is restarted, it may be necessary to manually restart the InterSystems IRIS SNMP subagent using the **^SNMP** routine, as described in the foregoing.

A.4 SNMP Troubleshooting

The InterSystems IRIS subagent (running the **^SNMP** routine) depends on the correct installation and configuration of the SNMP master agent supplied by the operating system. As noted in InterSystems IRIS as a Subagent, there are two main ways in which the **^SNMP** routine communicates with this master agent:

- Primarily, **^SNMP** uses the AgentX protocol on TCP port 705.
- On Windows, **^SNMP** uses a Windows extension agent DLL installed as iscsnmp.dll.

Detailed instructions for configuring the SNMP agent should be supplied with the operating system, and system managers should take some time to understand how to do this. The following are some basic guidelines and tips for troubleshooting if problems are encountered in getting InterSystems IRIS to communicate with the SNMP agent.

A.4.1 All Systems

- Make sure the SNMP agent is working independently of InterSystems IRIS and you can at least query the `mib-2.system` tree for general system information. If this fails, on Windows check the Windows SNMP Service; on UNIX®/Linux see if the SNMP daemon (`snmpd`) is running.
- If you can successfully query the SNMP system information but not the InterSystems IRIS MIB, then check for a background process in InterSystems IRIS running the **^SNMP** routine. Try starting it using the **$$start^SNMP()** function. If the routine starts but does not continue running, check for errors in the messages.log and SNMP.log log files in the InterSystems IRIS *install-dir*/mgr directory. On Windows, iscsnmp.dll logs any errors it encounters in Windows\System32\snmpdbg.log (on a 64–bit Windows system, the file is in the SysWOW64 subdirectory).
- Make sure the InterSystems IRIS `%Service_Monitor` service is enabled.
- More information can be logged to the SNMP.log file if you set **^SYS("MONITOR","SNMP","DEBUG")=1** in the %SYS namespace and restart the **^SNMP** InterSystems IRIS subagent process. This logs details about each message received and sent.

A.4.2 Windows Systems

- Not all Windows versions install the Windows SNMP service by default. You may need to do this as an additional step. Make sure the **Security** tab of the **Properties** dialog for the service has at least a **public** community with **READ** rights. To send SNMP traps, you must define a **Community Name** and **Destination** on the **Trap** tab of the properties dialog.
- InterSystems IRIS expects the SNMP service to be installed *before* you install InterSystems IRIS, so it can add iscsnmp.dll to the proper Registry keys. Once InterSystems IRIS is installed, the SNMP service must be restarted so that it properly loads iscsnmp.dll and can find and communicate with the new InterSystems IRIS instance.

Note: If InterSystems IRIS is installed before the SNMP service, iscsnmp.dll cannot be properly registered, and you must use the **set myStatus=$$Register^SNMP()** function to do this after the Windows SNMP service is installed. Once this is done the SNMP service must be restarted.

- On Windows, the **$$start^SNMP()** function only signals the SNMP service, and the InterSystems IRIS **^SNMP** process is actually started by a callback from the SNMP service into InterSystems IRIS. It may take a few seconds for the process to start, and a few more seconds before it can respond to queries.

A.4.3 UNIX® Systems

Many UNIX operating systems (IBM AIX®) do *not* support the AgentX protocol at this time. If your system does not support AgentX, you must install a separate SNMP agent which supports AgentX, such as Net-SNMP.

A.4.4 Linux and macOS with Net-SNMP

- AgentX support is *not* enabled by default, and the default port is not 705. You must modify the snmpd.conf file and add **master agentx** and **agentXSocket TCP:localhost:705**, or use **snmpd -x TCP:localhost:705** on the command line.
- Basic system information like `syslocation`, `syscontact` and `sysservices` must be defined in snmpd.conf to enable successful startup of the snmpd daemon.
- Recent versions of Net-SNMP also implement VACM (View-based Access Control Model) security and, by default, allow access to the `mib-2.system` subtree only; as a result, the InterSystems IRIS subagent starts and runs without error, but no SNMP requests are forwarded to InterSystems IRIS. You must expand the "views" defined in snmpd.conf to include the InterSystems IRIS MIB subtree.
- To send SNMP traps, you must define a destination using the **trapsink** parameter in snmpd.conf, for example **trapsink 192.16.61.36 public**.

A.5 InterSystems IRIS MIB Structure

All of the managed object data available through the InterSystems IRIS SNMP interface is defined in the InterSystems IRIS MIB file, ISC-IRIS.mib, which is located in the *install-dir*\SNMP directory. Typically an SNMP management application must load the MIB file for the managed application to understand and appropriately display the information. Since this procedure varies among applications, consult your management application documentation for the appropriate way to load the InterSystems IRIS MIB.

The specific data defined in the InterSystems IRIS MIB is documented in the file itself and, therefore, is not repeated here. However, it may be valuable to understand the overall structure of the InterSystems IRIS MIB tree, especially as it relates to multiple instances on the same system.

Note: The best way to view the MIB tree is to load the MIB into a management application or MIB browser. These tools display the MIB as a tree with object IDs (OIDs), matching text representations of the objects, and descriptions of the objects.

SNMP defines the Structure of Management Information (SMI), a specific, hierarchical tree structure for all managed objects, 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 InterSystems IRIS does, an application uses the *enterprise* branch which is defined as:

```
iso.org.dod.internet.private.enterprises (1.3.6.1.4.1)
```

The Internet Assigned Numbers Authority (IANA) assigns each organization a private enterprise number as the next level in the hierarchy. For InterSystems IRIS this is `16563`, which represents *intersystems*.

Below this, InterSystems IRIS implements its enterprise private subtree as follows:

- The level below *intersystems* is the "product" or application ID level. For InterSystems IRIS this is `.4` (`iscIris`). This serves as the MIB module identity.
- The next level is the "object" level, which separates data objects from notifications. For InterSystems IRIS, these are `.1` (`irisObjects`) and `.2` (`irisTraps`). By convention, the *intersystems* tree uses a brief lowercase prefix added to all data objects and notification names. For InterSystems IRIS this is *iris*.
- 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 InterSystems IRIS instances on one machine. All tables use the InterSystems IRIS instance name as the first index of the table. The tables may also have one or more additional indexes.
- The next level, which is always `.1`, is the "conceptual row" for the table (as required by the SNMP SMI).
- 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". For more information, see InterSystems IRIS SNMP Traps.
- InterSystems IRIS-specific auxiliary objects sent via notifications (traps) are defined as individual entries at the same hierarchical level as the "table". For more information, see InterSystems IRIS SNMP Traps.

For example, encode the size of a database as:

```
1.3.6.1.4.1.16563.4.1.3.1.6.4.84.69.83.84.1
```

This translates to:

```
iso.org.dod.internet.private.enterprises.intersystems.isciris.irisObjects
irisDBTab.irisDBRow.irisDBSize.TEST(instname).1(DBindex)
```

A.5.1 Extending the InterSystems IRIS MIB

Application programmers can add managed object definitions and extend the MIB for which the InterSystems IRIS subagent provides data. This is not intended to be a complete MIB editor or SNMP toolkit; rather, it is a way to add simple application metrics that you can browse or query through SNMP.

Note: The objects must follow the basic InterSystems IRIS SNMP structure, there is limited support for SNMP table structures (only integer-valued indexes are supported), and SNMP traps are not created (see the %Monitor.Alert class). A basic understanding of SNMP structure of management information is helpful.

To create these objects do the following:

1. Create InterSystems IRIS object definitions in classes that inherit from the %Monitor.Adaptor class. See the *InterSystems Class Reference* for details about adding managed objects to the %Monitor package.
2. Execute an SNMP class method to enable these managed objects in SNMP and create a MIB definition file for management applications to use. The method to accomplish this is **MonitorTools.SNMP.CreateMIB()**.

See the MonitorTools.SNMP class documentation 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 InterSystems IRIS subagent uses this to register the MIB subtree, walk the tree for **GETNEXT** requests, and reference specific objects methods for gathering the instance data in **GET** requests.

All the managed object definitions use the same general organization as the InterSystems IRIS enterprise MIB tree, that is: `application.objects.table.row.item.indices`. The first index for all tables is the InterSystems IRIS 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 InterSystems IRIS subagent no longer registers or answers requests for that private enterprise MIB subtree.

For an example of defining a user Monitor class, see Sample User-Defined SNMP Monitor Class.

A.5.2 InterSystems IRIS SNMP Traps

In addition to the object data and metrics available through SNMP queries, InterSystems IRIS can send asynchronous alerts or SNMP traps. The following table describes the InterSystems IRIS-specific SNMP traps.

Table I–1: InterSystems IRIS SNMP Notification Objects (Traps)

Trap Name (Number)	Description
irisStart (1)	The InterSystems IRIS instance has been started.
irisStop (2)	The InterSystems IRIS instance is in the process of shutting down.
irisDBExpand (3)	An InterSystems IRIS database has expanded successfully.
irisDBOutOfSpace (4)	Future expansion of an InterSystems IRIS database may be limited; there is not enough free space on the file system for 10 more expansions or there is less than 50 MB of free space.
irisDBStatusChange (5)	The read/write status of an InterSystems IRIS database has been changed.
irisDBWriteFail (6)	A write to an InterSystems IRIS database has failed. It includes the InterSystems IRIS error code for the failed write.
irisWDStop (7)	The write daemon for an InterSystems IRIS instance has stalled.
irisWDPanic (8)	The write daemon for an InterSystems IRIS instance has entered “panic” mode; that is, the write daemon is out of buffers and must write database blocks directly to disk without first committing them to the Write Image Journal (WIJ) file.
irisLockTableFull (9)	The lock table for an InterSystems IRIS instance is full, which causes subsequent Locks to fail.
irisProcessFail (10)	A process has exited InterSystems IRIS abnormally (due to an access violation). For detailed information, see the messages.log file.
irisECPTroubleDSrv (11)	A connection to this ECP Data Server for an InterSystems IRIS database has encountered a serious communication problem. For detailed information, see the messages.log file.
irisECPTroubleASrv (12)	A connection from this ECP Application Server to a remote InterSystems IRIS database has encountered a serious communication problem. For detailed information, see the messages.log file.

Trap Name (Number)	Description
irisAuditLost (13)	InterSystems IRIS has failed to record an Audit event. The most likely cause is a problem with space for the Audit database, which requires operator assistance.
irisLoggedError (14)	A “severe ” error has been logged in the messages.log file. This trap includes the error message defined in irisSysErrorMsg.
irisLicenseExceed (15)	A request for a license has exceeded the number of licenses currently available or allowed.
irisEventLogPost (16)	An entry posted in the Interoperability Event Log.
irisAppAlert (100)	This is a generic trap that can be used by InterSystems IRIS applications to generate alerts via SNMP. For detailed information about how this trap can be used, see the **%Monitor.Alert.External** class method.

The following table describes the InterSystems IRIS-specific auxiliary objects that can be sent in the traps described in the preceding table.

Table I–2: InterSystems IRIS-specific Auxiliary Objects Sent in Traps

Auxiliary Object Name (Number)	Description
irisDBWriteError (1)	The InterSystems IRIS-specific error code for a failed database write. Possible values are: <DATABASE>, <DISKHARD>, <BLOCKNUMBER>, <FILEFULL> or <DATABASE MAP LABEL>.
irisApp (2)	A short text string (maximum of 20 characters) that identifies the application that generated (or was the source of) an irisAppAlert trap.
irisAppSeverity (3)	A code that indicates the severity of the problem for a irisAppAlert trap. The code can be `0` (info), `1` (warning), `2` (severe), or `3` (fatal).
irisApptext (4)	A text string description (maximum of 1024 characters) of the problem, error, or event that caused the irisAppAlert trap.

A.6 Sample User-Defined SNMP Monitor Class

This section describes an example of how to define a user Application Monitor class (see Application Monitor) that you can query via SNMP. The Application Monitor includes only properties with %Monitor data types in the SNMP data.

Example Sample Class

The following is the sample class for this example:

Class Definition

```
Class SNMP.Example Extends %Monitor.Adaptor
{

  /// Give the application a name. This allows you to group different
  /// classes together under the same application level in the SNMP MIB.
  /// The default is the same as the Package name.
  Parameter APPLICATION = "MyApp";

  /// This groups a set of properties together at the "table" level of the
  /// SNMP MIB hierarchy. The default is the Class name.
  Parameter GROUPNAME = "MyTable";
```

```
  /// An integer metric counter
  Property Counter1 As %Monitor.Integer;

  /// Another integer metric counter
  Property Counter2 As %Monitor.Integer;

  /// A status indicator as a string data type
  Property Status As %Monitor.String;

  /// The method is REQUIRED. It is where the Application Monitor
  /// calls to collect data samples, which then get picked up by the
  /// ^SNMP server process when requested.
  Method GetSample() As %Status
  {
    set ..Counter1=$r(200)
    set ..Counter2=200+$r(100)
    set n=$r(4)
    set ..Status=$s(n=1:"Crashed",n=2:"Warning",n=3:"Error",1:"Normal")
    Quit $$$OK
  }

}
```

Before compiling this class in a user namespace, InterSystems IRIS must load supporting classes into the namespace; these classes are required to store the data samples for SNMP.

To load the classes, run **^%SYSMONMGR**, as described in Using ^%SYSMONMGR to Manage Application Monitor, and do the following:

- Select option 2, `Manage Monitor Classes`.
- Select option 3, `Register Monitor System Classes`.

When you compile the sample class, it creates the SNMP.Sample.Example class to store the sample data.

Important: Do not delete generated sample classes explicitly; if you select both the Application Monitor and generated sample classes for deletion, the sample class routines remain although the monitor class routines are deleted, which causes an error. To ensure that all sample class routines are properly removed, delete *only* the Application Monitor class that generated it; when you delete the monitor class both the monitor class and generated sample class, as well as related routines for both classes, are deleted. For example, to delete a sample class (for example, SNMP.Sample.Example), use the Management Portal to delete the monitor class from which it is generated (that is, SNMP.Example).

Run **^%SYSMONMGR** to activate the sample class and start the Application Monitor to collect samples:

1. Select option 2, `Manage Monitor Classes`.
2. Select option 1, `Activate/Deactivate a Monitor Class`.
3. To see an numbered list of registered Monitor Classes, enter `?`.
4. Enter the number of Monitor Class you want to activate; for example, to activate a user-defined class named SNMP.Example, enter the number next to the class name.
5. Select option 6, `Exit` (to return to the Application Monitor main menu).
6. Select option 1, `Manage Application Monitor`.
7. Select option 1, `Start Application Monitor`.
8. Select option 5, `Exit` (to return to the Application Monitor main menu).
9. Select option 6, `Exit` (to exit from Application Monitor main menu).

Note: For information about configuring and using Application Monitor, see Application Monitor.

Example of Creating a User MIB

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

The input parameters for the method are similar to the following:

```
CreateMIB("MyApp","USER",99990,1,"mycorp","myapp","mc","MC-MYAPP","Unknown",1)
```

Important: Do not use 99990 as the Enterprise ID for production; each organization should register with the IANA for their own ID.

```
USER>set $namespace = "%SYS"

%SYS>Do ##class(MonitorTools.SNMP).CreateMIB("MyApp","USER",99990,1,"mycorp",
"myapp","mc","MC-MYAPP","Unknown",1)


Create SNMP structure for Application - MyApp

     Group - MyTable
          Counter1 = Integer
          Counter2 = Integer
          Status = String


Create MIB file for MyApp

     Generate table MyTable
          Add object Counter1
          Add object Counter2
          Add object Status


%SYS>
```

This creates the MC-MYAPP.MIB file in your default directory (*install-dir*\mgr\User), which you can load into your SNMP management application.

Note: You may need to restart the SNMP master agent and the InterSystems IRIS ^**SNMP** service on your system before each recognizes this MIB.

```
--
-- MIB file generated for mcMyApp product.
--
-- Sep 16, 2008
--

MC-MYAPP DEFINITIONS ::= BEGIN

IMPORTS

  MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
  Counter32, Gauge32, Integer32
  FROM SNMPv2-SMI
  DisplayString
  FROM SNMPv2-TC
  enterprises
  FROM RFC1155-SMI
  cacheSysIndex
  FROM ISC-IRIS;


mcMyApp MODULE-IDENTITY
    LAST-UPDATED "200809161700Z"
    ORGANIZATION "mycorp"
    CONTACT-INFO "
                  Unknown"
    DESCRIPTION ""
    ::= { mycorp 1 }

mycorp OBJECT IDENTIFIER ::= { enterprises 16563 }

myappObjects OBJECT IDENTIFIER ::= { mcMyApp 1 }
```

```
--
-- Begin tables
--

-- Table myappMyTable

myappMyTable         OBJECT-TYPE
       SYNTAX        SEQUENCE OF myappMyTableR
       MAX-ACCESS    not-accessible
       STATUS        current
       DESCRIPTION
       ""
   ::= { myappObjects 1 }

myappMyTableR        OBJECT-TYPE
       SYNTAX        myappMyTableR
       MAX-ACCESS    not-accessible
       STATUS        current
       DESCRIPTION
       "Conceptual row for MyTable table."
       INDEX  { cacheSysIndex }
   ::= { myappMyTable 1 }

myappMyTableR ::=
       SEQUENCE {
        myappCounter1   Integer32
        myappCounter2   Integer32
        myappStatus   DisplayString
       }

myappCounter1        OBJECT-TYPE
       SYNTAX        Integer32
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       ""
       ::= { myappMyTableR 1 }

myappCounter2        OBJECT-TYPE
       SYNTAX        Integer32
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       ""
       ::= { myappMyTableR 2 }

myappStatus          OBJECT-TYPE
       SYNTAX        DisplayString
       MAX-ACCESS    read-only
       STATUS        current
       DESCRIPTION
       "Status"
       ::= { myappMyTableR 3 }

--  End of MyTable table

myappTraps  OBJECT IDENTIFIER ::= { mcMyApp 2 }

-------------------------------------------------------------
 END
```

B

Monitoring InterSystems IRIS Using Web Services

This topic introduces and briefly describes how to use InterSystems IRIS® data platform support for the WS-Management specification, which enables you to remotely monitor an InterSystems IRIS instance via SOAP.

B.1 Overview of InterSystems IRIS Support for WS-Monitoring

Following the WS-Management specification, the SYS.WSMon package provides a web service that you can use to remotely monitor an InterSystems IRIS instance. It is functionally similar to the SNMP interface (see Monitoring InterSystems IRIS Using SNMP), but uses the built-in InterSystems IRIS web service services support.

The support for WS-Management includes the following elements:

- The *Log Monitoring Web Service* (SYS.WSMon.Service) that provides methods that return information about an InterSystems IRIS instance.
- An InterSystems IRIS web service client (SYS.WSMon.Client) that can invoke methods in this Monitoring Web Service or in the Monitoring Web Service of another InterSystems IRIS instance.

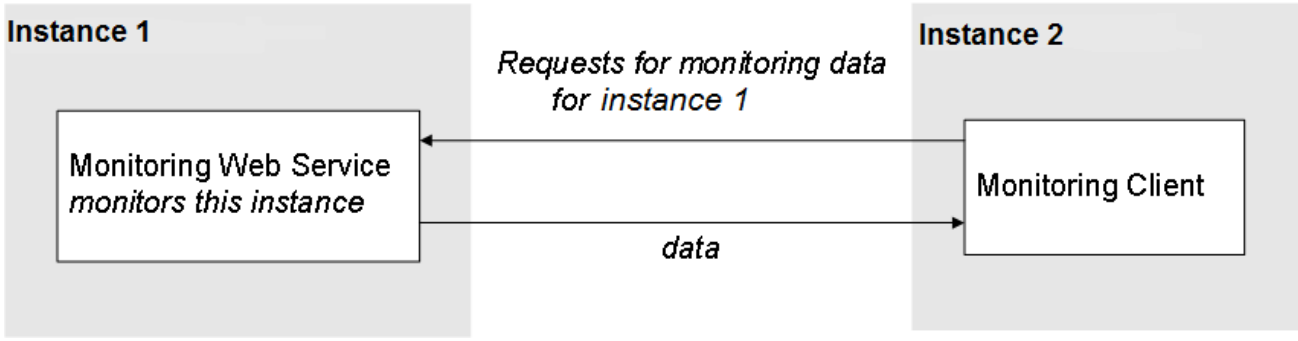
  Instead of using this web client, you can create your own web client, possibly using third-party technology.
- Several XML-enabled classes that this web service and client use to represent monitoring information.

  These classes include SYS.WSMon.wsEvent, which can represent events.
- A *sample event sink web service* (SYS.WSMon.EventSink) that can receive and process events. Via a SOAP call, you can subscribe to this sample event sink service so that it will receive events from any Monitoring Web Service.

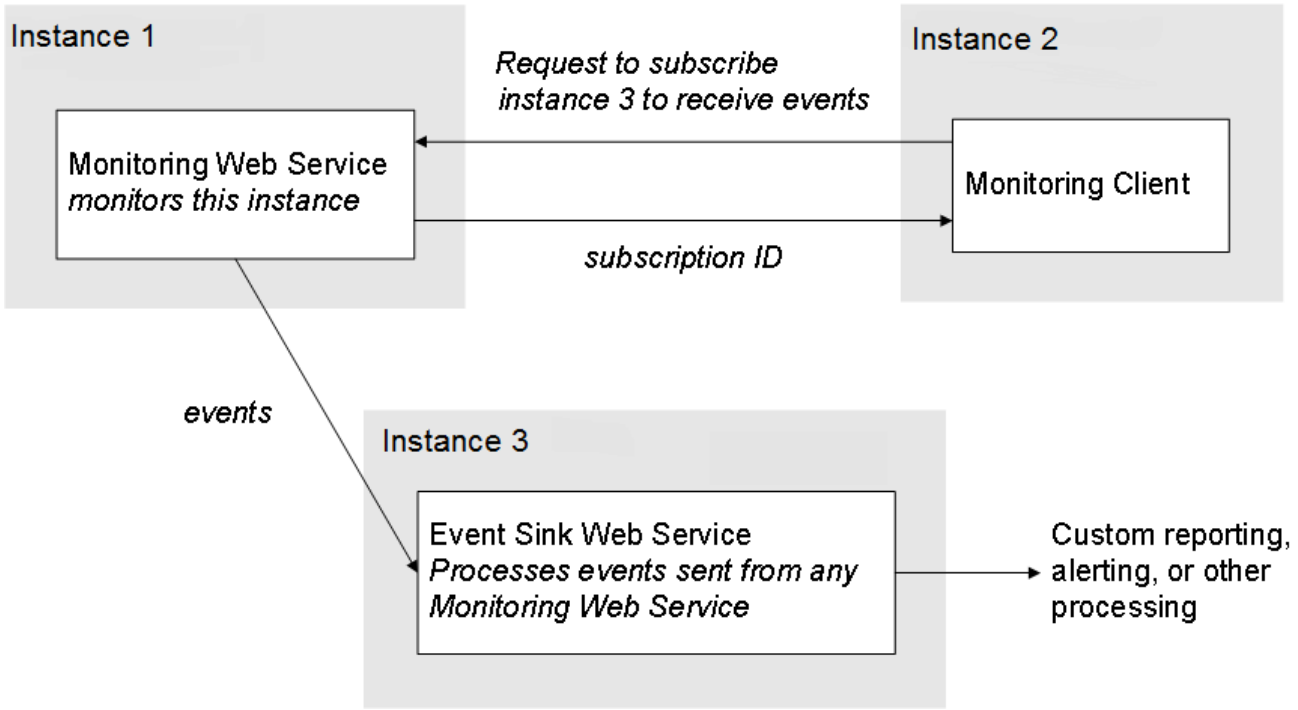
  Instead of using this sample, you can create your own, possibly using third-party technology.

These classes are available only in the %SYS namespace.

For basic monitoring, you can use the Log Monitoring Web Service with a web client in another instance. The image below shows a monitoring client requesting monitoring data from a target instance. The Log Monitoring Web Service on the target instance responds, sending the client monitoring data.



In more advanced cases, the web client subscribes an event sink service, possibly running on another InterSystems IRIS instance. For example, in the image below the monitoring client sends a request to the primary instance to subscribe a third instance. The primary instance's Monitoring Web Service responds by sending subscription ID to the client, and thereafter sends events to the third instance. The third instance can process these events in a number of ways, such as to create custom reporting or alerting.



Your event sink web service can perform any processing needed by the business.

For information on generating web services and web clients from a WSDL, see Creating Web Services and Web Clients.

B.2 Support Details

InterSystems supports the following parts of the WS-Management specification:

- wxf:Get
- wsen:Enumerate
- wsen:Pull
- wsen:Release

- wse:Subscribe
- wse:Renew
- wse:Unsubscribe

For more information, see the WS-Management specification (https://www.dmtf.org/standards/published_documents/DSP0226_1.1.pdf).

B.3 URL for the Monitoring Web Service

For a given InterSystems IRIS instance, the Log Monitoring Web Service is available at the following URL, using the <baseURL> for your instance:

```
http://<baseURL>/csp/sys/SYS.WSMon.Service.cls
```

For example:

```
http://localhost:8000/csp/sys/SYS.WSMon.Service.cls
```

Similarly, the WSDL for this web service is available at the following URL, again using the <baseURL> for your instance:

```
http://<baseURL>/csp/sys/SYS.WSMon.Service.cls?WSDL=1
```

B.4 Web Methods of the Monitoring Web Service

The SYS.WSMon.Service class provides the following web methods:

EnumBuffer()

```
method EnumBuffer() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates the statistics for all buffer sizes. For this instance, the dataset uses the Sample() class query of the SYS.Stats.Buffer class.

For information on working with %XML.DataSet, see Using Datasets in SOAP Messages or see the class reference for %XML.DataSet.

Also see the class reference for SYS.Stats.Buffer.

EnumDatabase()

```
method EnumDatabase() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates all databases for this instance. For this instance, the dataset uses the List() class query of the SYS.WSMon.wsDatabase class.

See the comments for **EnumBuffer()** and see the class reference for SYS.WSMon.wsDatabase.

EnumResource()

```
method EnumResource() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates statistics for all system resource seizes. For this instance, the dataset uses the Sample() class query of the SYS.Stats.Resource class.

See the comments for **EnumBuffer()** and see the class reference for SYS.Stats.Resource.

EnumWriteDaemon()

```
method EnumWriteDaemon() as %XML.DataSet
```

Returns an instance of %XML.DataSet that enumerates statistics for all write daemons. For this instance, the dataset uses the **Sample()** class query of the SYS.Stats.WriteDaemon class.

See the comments for **EnumBuffer()** and see the class reference for SYS.Stats.WriteDaemon.

EventCancel()

```
method EventCancel(id) as %Integer
```

Cancels the subscription for a given web service; see **EventSubscribe()**.

EventSubscribe()

```
method EventSubscribe(location) as %String
```

Subscribes the given web service to receive information about events in this InterSystems IRIS instance. This can be your own web service or can be the SYS.WSMon.EventSink web service, which is provided as an example. If you create your own web service, it must follow the WSDL of the SYS.WSMon.EventSink web service.

For *location*, specify the URL needed to invoke the **EventSink()** method of the web service, using the <baseURL> for your instance. For SYS.WSMon.EventSink, you might specify *location* as the following:

```
http://<baseURL>/csp/sys/SYS.WSMon.EventSink.cls
```

Where *server* is the server on which InterSystems IRIS is running, and *port* is the port that InterSystems IRIS uses.

For each event, InterSystems IRIS will attempt to call **EventSink()** method of the given web service, sending an instance of SYS.WSMon.wsEvent.

This method returns an ID that you can use to cancel the subscription; see **EventCancel()**.

GetDisk()

```
method GetDisk() as SYS.Stats.Disk
```

Returns an instance of SYS.Stats.Disk that contains metrics of disk usage for globals for this instance.

See the class reference for SYS.Stats.Disk.

GetECPAppSvr()

```
method GetECPAppSvr() as SYS.Stats.ECPAppSvr
```

Returns an instance of SYS.Stats.ECPAppSvr that contains ECP application server metrics for this instance.

See the class reference for SYS.Stats.ECPAppSvr.

GetECPDataSvr()

```
method GetECPDataSvr() as SYS.Stats.ECPDataSvr
```

Returns an instance of SYS.Stats.ECPDataSvr that contains ECP database server metrics for this instance.

See the class reference for SYS.Stats.ECPDataSvr.

GetGlobal()

```
method GetGlobal() as SYS.Stats.Global
```

Returns an instance of SYS.Stats.Global that contains global metrics for this instance.

See the class reference for SYS.Stats.Global.

GetRoutine()

```
method GetRoutine() as SYS.Stats.Routine
```

Returns an instance of SYS.Stats.Routine that contains routine metrics for this instance.

See the class reference for SYS.Stats.Routine.

GetSystem()

```
method GetSystem() as SYS.WSMon.wsSystem
```

Returns an instance of SYS.WSMon.wsSystem that contains system information about the InterSystems IRIS instance.

See the class reference for SYS.WSMon.wsSystem.

B.5 Monitoring Web Client

The SYS.WSMon.Client class and related classes are an InterSystems IRIS web service client that can invoke methods of SYS.WSMon.Server web service in the same InterSystems IRIS instance or another InterSystems IRIS instance.

This web client class uses the following LOCATION parameter, using the <baseURL> for your instance:

```
Parameter LOCATION = "http://<baseURL>/csp/sys/SYS.WSMon.Service.cls"
```

Where *server* is the server on which InterSystems IRIS is running and *port* is the port that the InterSystems IRIS web service server uses.

Use this web client in the same way that you use other InterSystems IRIS web service clients:

1. Create an instance of the web client class.
2. Set its Location property if needed.

   This is necessary if the SYS.WSMon.Server web service that you want to use is on a different machine than the client, or if it uses a port other than 52773.
3. Set other properties if needed.

   See Creating Web Services and Web Clients.
4. Invoke a web method.
5. Examine the value returned by the web method.

   The details depend on the web method you invoke; see Web Methods of the Monitoring Web Service and see the class reference for the return types.

The following shows an example Terminal session:

```
USER>set $namespace = "%SYS"

%SYS>set client=##class(SYS.WSMon.Client).%New()

%SYS>set client.Location="http://localhost:8000/csp/sys/SYS.WSMon.Service.cls"

%SYS>set myroutinestats=client.GetRoutine()

%SYS>write myroutinestats.RtnCallsLocal
19411581
%SYS>write myroutinestats.RtnCallsRemote
0
%SYS>write myroutinestats.RtnCommands
432764817
%SYS>
```

More typically, you create and use the client programmatically, perhaps to retrieve data for display in a user interface.

Note: Remember that the SYS.WSMon package is available only in %SYS namespace, which means that you must be in that namespace to perform the steps described here.

B.6 Processing Events

InterSystems IRIS provides a sample web service (SYS.WSMon.EventSink) that can receive and process events sent by any Log Monitoring Web Service. You can use this web service or create and use your own.

B.6.1 Using the Sample Event Sink Web Service

SYS.WSMon.EventSink is a sample InterSystems IRIS web service service that can receive and process events.

For a given InterSystems IRIS instance, the Log Monitoring Web Service is available at the following URL, using the <baseURL> for your instance:

```
http://<baseURL>/csp/sys/SYS.WSMon.EventSink.cls
```

Where *server* is the server on which InterSystems IRIS is running and *port* is the port that the InterSystems IRIS web service server uses.

This web service has one method:

CacheEventSink()

```
Method CacheEventSink(event As SYS.WSMon.wsEvent) As %Integer
```

On Windows platforms, this sample method displays a popup window when an event occurs; for other platforms, it adds an entry to `^SYS("MONITOR","WSMON","EVENT_RECEIVED",$h)`.

This method always returns 1.

To subscribe this sample service so that it will receive events from the Monitoring Web Service, do the following in the Terminal:

```
USER>set $namespace = "%sys"

%SYS>set client=##class(SYS.WSMon.Client).%New()

%SYS>set eventsinklocation="http://localhost:8000/csp/sys/SYS.WSMon.EventSink.cls"

%SYS>set subscriptionid=client.EventSubscribe(eventsinklocation)

%SYS>write subscriptionid
CacheEventSubscription_2
```

Here *eventsinklocation* is the URL for the event sink web service that will process events.

B.6.2 Creating Your Own Event Sink Web Service

To create your own event sink web service, generate a web service from the following WSDL, using the <baseURL> for your instance:

```
http://<baseURL>/csp/sys/SYS.WSMon.EventSink.cls?WSDL=1
```

Where *server* is the server on which InterSystems IRIS is running and *port* is the port that the InterSystems IRIS web service server uses.

For details, see Creating Web Services and Web Clients.

Then modify the **CacheEventSink()** method in the generated web service to include your custom logic.

C

Monitoring InterSystems IRIS via REST

Every InterSystems IRIS® data platform instance contains a REST interface that provides statistics about the instance. This REST API provides a way to gather information from multiple machines running InterSystems IRIS, allowing you to monitor in detail all instances that comprise your application. The API follows the OpenMetrics standard.

This topic describes the metrics provided by the /api/monitor service. These metrics are compatible with Prometheus, an open-source monitoring and alerting tool. Configuring Prometheus to scrape multiple connected InterSystems IRIS instances provides a cohesive view of your entire system, making it easier to evaluate whether the system is behaving properly and efficiently.

C.1 Introduction to /api/monitor Service

The /api/monitor service provides information about the InterSystems IRIS Instance on which it runs. By default, the /api/monitor web application is enabled with "Unauthenticated" access. For information about setting up authentication for this service, see Securing REST Services.

This API has the following two endpoints:

- /api/monitor/metrics, which returns all instance metrics, and can be configured to return specific application metrics.
- /api/monitor/alerts, which returns any system alerts that have been posted since the endpoint was last scraped.

Note: InterSystems IRIS logs any errors in the SystemMonitor.log file, which is located in the install-dir/mgr directory.

C.2 /api/monitor/metrics

The /api/monitor/metrics endpoint returns a list of metrics, which are described in Metric Descriptions. You can also enable the collection of additional metrics about active interoperability productions, as described in Interoperability Metrics. Create Application Metrics contains instructions for how to define custom metrics.

To configure Prometheus to scrape an instance of InterSystems IRIS, follow the instructions in *First Steps With Prometheus* (https://prometheus.io/docs/introduction/first_steps/).

C.2.1 Metric Descriptions

The metrics are returned in a text-based format, described in the *Exposition Formats* page of the Prometheus documentation (https://prometheus.io/docs/instrumenting/exposition_formats/). Each metric is listed on a single line with only one space, which separates the name from the value.

InterSystems IRIS metrics are listed in the table below. Metric names with a label appear here with line breaks to improve readability.

Note: This table contains metrics for the version of InterSystems IRIS documented here. As metrics may be added in newer versions, be sure this documentation matches your version of InterSystems IRIS.

Metric Name	Description
`iris_cpu_pct {id="ProcessType"}`	Percent of CPU usage by InterSystems IRIS process type. *ProcessType* can be any of the following: `ECPWorker, ECPCliR, ECPCliW, ECPSrvR, ECPSrvW, LICENSESRV, WDAUX, WRTDMN, JRNDMN, GARCOL, CSPDMN, CSPSRV, ODBCSRC, MirrorMaster, MirrorPri, MirrorBack, MirrorPre, MirrorSvrR, MirrorJrnR, MirrorSK, MirrorComm` (see Secure InterSystems Processes and Operating System Resources)
`iris_cpu_usage`	Percent of CPU usage for all programs on the operating system
`iris_csp_activity {id="IPaddress:port"}`	Number of web requests served by the Web Gateway Server since it was started
`iris_csp_actual_connections {id="IPAddress:port"}`	Number of current connections to this server by the Web Gateway Server
`iris_csp_gateway_latency {id="IPaddress:port"}`	Amount of time to obtain a response from the Web Gateway Server when fetching `iris_csp_` metrics, in milliseconds
`iris_csp_in_use_connections {id="IPaddress:port"}`	Number of current connections to this server by the Web Gateway Server that are processing a web request
`iris_csp_private_connections {id="IPaddress:port"}`	Number of current connections to this server by the Web Gateway Server that are reserved for state-aware applications (Preserve mode 1)
`iris_csp_sessions`	Number of currently active web session IDs on this server
`iris_cache_efficiency`	Ratio of global references to physical reads and writes, as a percent
`iris_db_expansion_size_mb {id="database"}`	Amount by which to expand *database*, in megabytes

Metric Name	Description
`iris_db_free_space {id="database"}`	Free space available in *database*, in megabytes (This metric is only updated once per day, and may not reflect recent changes.)
`iris_db_latency {id="database"}`	Amount of time to complete a random read from *database*, in milliseconds
`iris_db_max_size_mb {id="database"}`	Maximum size to which *database* can grow, in megabytes
`iris_db_size_mb {id="database",dir="path"}`	Size of *database*, in megabytes
`iris_directory_space {id="database",dir="path"}`	Free space available on the *database* directory's storage volume, in megabytes
`iris_disk_percent_full {id="database",dir="path"}`	Percent of space filled on the *database* directory's storage volume
`iris_ecp_conn`	Total number of active client connections on this ECP application server
`iris_ecp_conn_max`	Maximum active client connections from this ECP application server
`iris_ecp_connections`	Number of servers synchronized when this ECP application server synchronizes with its configured ECP data servers
`iris_ecp_latency`	Latency between the ECP application server and the ECP data server, in milliseconds
`iris_ecps_conn`	Total active client connections to this ECP data server per second
`iris_ecps_conn_max`	Maximum active client connections to this ECP data server
`iris_glo_a_seize_per_sec`	Number of Aseizes on the global resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_glo_n_seize_per_sec`	Number of Nseizes on the global resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_glo_ref_per_sec`	Number of references to globals located on local databases per second
`iris_glo_ref_rem_per_sec`	Number of references to globals located on remote databases per second
`iris_glo_seize_per_sec`	Number of seizes on the global resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_glo_update_per_sec`	Number of updates (**SET** and **KILL** commands) to globals located on local databases per second

Metric Name	Description
`iris_glo_update_rem_per_sec`	Number of updates (**SET** and **KILL** commands) to globals located on remote databases per second
`iris_jrn_block_per_sec`	Journal blocks written to disk per second
`iris_jrn_free_space {id="JournalType",dir="path"}`	Free space available on each journal directory's storage volume, in megabytes. *JournalType* can be `WIJ`, `primary`, or `secondary`
`iris_jrn_size {id="JournalType"}`	Current size of each journal file, in megabytes. *JournalType* can be `WIJ`, `primary`, or `secondary`
`iris_license_available`	Number of licenses not currently in use
`iris_license_consumed`	Number of licenses currently in use
`iris_license_percent_used`	Percent of licenses currently in use
`iris_log_reads_per_sec`	Logical reads per second
`iris_obj_a_seize_per_sec`	Number of Aseizes on the object resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_obj_del_per_sec`	Number of objects deleted per second
`iris_obj_hit_per_sec`	Number of object references per second, in process memory
`iris_obj_load_per_sec`	Number of objects loaded from disk per second, not in shared memory
`iris_obj_miss_per_sec`	Number of object references not found in memory per second
`iris_obj_new_per_sec`	Number of objects initialized per second
`iris_obj_seize_per_sec`	Number of seizes on the object resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_page_space_percent_used`	Percent of maximum allocated page file space used
`iris_phys_mem_percent_used`	Percent of physical memory (RAM) currently in use
`iris_phys_reads_per_sec`	Physical database blocks read from disk per second
`iris_phys_writes_per_sec`	Physical database blocks written to disk per second
`iris_process_count`	Total number of active InterSystems IRIS processes
`iris_rtn_a_seize_per_sec`	Number of Aseizes on the routine resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_rtn_call_local_per_sec`	Number of local routine calls per second to globals located on remote databases per second
`iris_rtn_call_miss_per_sec`	Number of routines calls not found in memory per second
`iris_rtn_call_remote_per_sec`	Number of remote routine calls per second

Metric Name	Description
`iris_rtn_load_per_sec`	Number of routines locally loaded from or saved to disk per second
`iris_rtn_load_rem_per_sec`	Number of routines remotely loaded from or saved to disk per second
`iris_rtn_seize_per_sec`	Number of seizes on the routine resource per second (see Considering Seizes, ASeizes, and NSeizes)
`iris_sam_get_db_sensors_seconds`	Amount of time it took to collect `iris_db*` sensors, in seconds
`iris_sam_get_jrn_sensors_seconds`	Amount of time it took to collect `iris_jrn*` sensors, in seconds
`iris_sam_get_sql_sensors_seconds`	Amount of time it took to collect `iris_sql*` sensors, in seconds
`iris_sam_get_wqm_sensors_seconds`	Amount of time it took to collect `iris_wqm*` sensors, in seconds
`iris_smh_available {id="purpose"}`	Shared memory available by *purpose*, in kilobytes (For more information, including a list of identifiers for *purpose,* see Generic (Shared) Memory Heap Usage.)
`iris_smh_percent_full {id="purpose"}`	Percent of allocated shared memory in use by *purpose* (For more information, including a list of identifiers for *purpose,* see Generic (Shared) Memory Heap Usage.)
`iris_smh_total`	Shared memory allocated for current instance, in kilobytes
`iris_smh_total_percent_full`	Percent of allocated shared memory in use for current instance
`iris_smh_used {id="purpose"}`	Shared memory in use by *purpose*, in kilobytes (For more information, including a list of identifiers for *purpose,* see Generic (Shared) Memory Heap Usage.)
`iris_sql_active_queries {id="namespace"}`	The number of SQL statements currently executing
`iris_sql_active_queries_95_percentile {id="namespace"}`	For the current set of active SQL statements, the 95th percentile elapsed time since a statement began executing
`iris_sql_active_queries_99_percentile {id="namespace"}`	For the current set of active SQL statements, the 99th percentile elapsed time since a statement began executing
`iris_sql_commands_per_second {id="namespace"}`	Average number of ObjectScript commands executed to perform SQL queries, per second
`iris_sql_queries_avg_runtime {id="namespace"}`	Average SQL statement runtime, in seconds

Metric Name	Description
`iris_sql_queries_avg_runtime_std_dev {id="namespace"}`	Standard deviation of the average SQL statement runtime
`iris_sql_queries_per_second {id="namespace"}`	Average number of SQL statements, per second
`iris_system_alerts`	The number of alerts posted to the messages log since system startup
`iris_system_alerts_log`	The number of alerts currently located in the alerts log
`iris_system_alerts_new`	Whether new alerts are available on the /api/monitor/alerts endpoint, as a Boolean
`iris_system_state`	A number representing the system monitor health state (see System Monitor Health State.)
`iris_trans_open_count`	Number of open transactions on the current instance
`iris_trans_open_secs`	Average duration of open transactions on the current instance, in seconds
`iris_trans_open_secs_max`	Duration of longest currently open transaction on the current instance, in seconds
`iris_wd_buffer_redirty`	Number of database buffers the write daemon wrote during the most recent cycle that were also written in prior cycle
`iris_wd_buffer_write`	Number of database buffers the write daemon wrote during its most recent cycle
`iris_wd_cycle_time`	Amount of time the most recent write daemon cycle took to complete, in milliseconds
`iris_wd_proc_in_global`	Number of processes actively holding global buffers at start of the most recent write daemon cycle
`iris_wd_size_write`	Size of database buffers the write daemon wrote during its most recent cycle, in kilobytes
`iris_wd_sleep`	Amount of time that the write daemon was inactive before its most recent cycle began, in milliseconds
`iris_wd_temp_queue`	Number of in-memory buffers the write daemon used at the start of its most recent cycle
`iris_wd_temp_write`	Number of in-memory buffers the write daemon wrote during its most recent cycle
`iris_wdwij_time`	Amount of time the write daemon spent writing to the WIJ file during its most recent cycle, in milliseconds
`iris_wd_write_time`	Amount of time the write daemon spent writing buffers to databases during its most recent cycle, in milliseconds
`iris_wij_writes_per_sec`	WIJ physical block writes per second

Metric Name	Description
`iris_wqm_active_worker_jobs {id="category"}`	Average number of worker jobs running logic that are not blocked
`iris_wqm_commands_per_sec {id="category"}`	Average number of commands executed in this Work Queue Management category, per second
`iris_wqm_globals_per_sec {id="category"}`	Average number of global references run in this Work Queue Management category, per second
`iris_wqm_max_active_worker_jobs {id="category"}`	Maximum number of active workers since the last log entry was recorded
`iris_wqm_max_work_queue_depth {id="category"}`	Maximum number of entries in the queue of this Work Queue Management category since the last log
`iris_wqm_waiting_worker_jobs {id="category"}`	Average number of idle worker jobs waiting for a group to connect to and do work for

C.2.2 Interoperability Metrics

In addition to the metrics described in the previous section, an InterSystems IRIS instance can also record metrics about active interoperability productions and include them in the output of the /metrics endpoint. The recording of these interoperability metrics is disabled by default. To enable it, you must perform the following steps for each interoperability production you want to monitor:

1. Open a Terminal session for the InterSystems IRIS instance running the production you want to monitor. If necessary, switch to the namespace associated with the production by executing the following command:

   ```
   set $namespace = "[interopNS]"
   ```

   where *[interopNS]* is the namespace name.

2. In the Terminal, execute the following command to enable the collection of metrics for the active production within the current namespace (SAM refers to System Alerting and Monitoring, the InterSystems monitoring solution):

   ```
   do ##class(Ens.Util.Statistics).EnableSAMForNamespace()
   ```

   Note: If the recording of metrics is enabled for a namespace but the corresponding production is not active, the /metrics endpoint does not return any metrics.

   The Ens.Util.Statistics class provides methods for customizing the output of the /metrics endpoint. For example, invoking the method **DisableSAMIncludeHostLabel** will provide aggregated metrics for the entire production instead of providing them for each host individually.

   The metrics available after completing this step are described in the *Basic Interoperability Metrics* table below.

3. To collect additional metrics for a production, enable activity monitoring by invoking the class method **Ens.Util.Statistics.EnableStatsForProduction** in the corresponding namespace using the Terminal. You must also add the Ens.Activity.Operation.Local business operation to the production. This process is detailed in Enabling Activity Monitoring.

   The additional metrics available after completing this step are described in the *Activity Volume Metrics* table below.

4. To collect additional HTTP transmission metrics for EnsLib.HTTP.OutboundAdapter or the EnsLib.SOAP.Outbound-Adapter (used in productions), enable the reporting of HTTP metrics for the corresponding business operation by performing the following steps:

   a. Open the Management Portal for the InterSystems IRIS instance containing the web client you want to monitor.

   b. Select **Interoperability** and choose the namespace containing the web client.

   c. Select **Configure** > **Production** to open the **Production Configuration** page.

   d. Select the operation which uses the HTTP or SOAP outbound adapter.

   e. In the **Alerting Control** section of the **Production Settings** > **Settings** panel, select the **Provide Metrics for SAM** check box.

   f. Select **Apply** to save your settings.

   The additional metrics available after completing this step are described in the *HTTP Metrics* table below.

   Note: Currently, HTTP transmission metrics are only collected for business operations which invoke actors using the **Queue** style (not **inProc**). For more information on the difference between these invocation styles, see Defining a Business Operation Class.

InterSystems IRIS interoperability metrics are listed in the tables below. Metric names with a label appear here with line breaks to improve readability.

Note: These tables contain metrics for the version of InterSystems IRIS documented here. As metrics may be added in newer versions, be sure this documentation matches your version of InterSystems IRIS.

Table III–1: Basic Interoperability Metrics

Metric Name	Description
`iris_interop_alert_delay {id="namespace",host="host",production="production"}`	Number of hosts within the *production* and *namespace* that have triggered a Queue Wait Alert. If output has been configured to include host labels, the hosts that have triggered Queue Wait Alerts are provided separately and the value will be `1`.
`iris_interop_hosts {id='namespace',status='status',host='host',production='production'}`	Number of hosts within the *production* and *namespace* which currently have the specified *status*. If output has been configured to include host labels, the status of each host is provided separately and the value will be `1`. *status* can be `OK`, `Error`, `Retry`, `Starting`, `Inactive`, or `Unconfigured`.
`iris_interop_messages {id="namespace",host="host",production="production"}`	Number of messages processed since the *production* started. If output has been configured to include host labels, the number of messages processed by each host is provided separately
`iris_interop_messages_per_sec {id="namespace",host="host",production="production"}`	Average number of messages processed within the *production* and *namespace* in a second over the most recent sampling interval. If output has been configured to include host labels, the number of messages processed by each host is provided separately
`iris_interop_queued {id="namespace",host="host",production="production"}`	Number of messages currently queued within the *production* and *namespace*. If output has been configured to include host labels, the number of messages currently queued for each host is provided separately.

Table III–2: Activity Volume Metrics

Metric Name	Description
`iris_interop_avg_processing_time` `{id="area",hostType="hostType",host="host",production="production",messageType="messageType"}`	Average length of time required to process a message of the specified *MessageType* within the *production* and *namespace*, in seconds. *HostType* can be `service`, `operation`, or `actor` (that is, process). *MessageType* is user-defined; if no *MessageType* is specified,"`-`" is returned. If output has been configured to include host labels, the message processing time for each host is provided separately.
`iris_interop_avg_queueing_time` `{id="area",hostType="hostType",host="host",production="production",messageType="messageType"}`	Average duration that a message of the specified *MessageType* spent in the queue while being processed by a host of *HostType* within the *production* and *namespace*, in seconds. *HostType* can be `service`, `operation`, or `actor` (that is, process). *MessageType* is user-defined; if no *MessageType* is specified,"`-`" is returned. If output has been configured to include host labels, the queueing time for each host is provided separately.
`iris_interop_sample_count` `{id="area",hostType="hostType",host="host",production="production",messageType="messageType"}`	Number of messages of the specified *MessageType* processed by a host of *HostType* within the *production* and *namespace* over the most recent sampling interval. *HostType* can be `service`, `operation`, or `actor` (that is, process). *MessageType* is user-defined; if no *MessageType* is specified,"`-`" is returned. If output has been configured to include host labels, the number of messages processed by each host is provided separately.
`iris_interop_sample_count_per_sec` `{id="area",hostType="hostType",host="host",production="production",messageType="messageType"}`	Number of messages of the specified *MessageType* processed per second by a host of *HostType* within the *production* and *namespace*, averaged over the most recent sampling interval. *HostType* can be `service`, `operation`, or `actor` (that is, process). *MessageType* is user-defined; if no *MessageType* is specified,"`-`" is returned. If output has been configured to include host labels, the number of messages processed by each host is provided separately.

Table III–3: HTTP Metrics

Metric Name	Description
`iris_interop_avg_http_received_chars {id="namespace",host="host",production="production"}`	Average number of characters received per HTTP or SOAP response within the *production* and *namespace* over the most recent sampling interval. If output has been configured to include host labels, the average number of characters received by each host is provided separately.
`iris_interop_avg_http_sent_chars {id="namespace",host="host",production="production"}`	Average number of characters sent per HTTP or SOAP request within the *production* and *namespace* over the most recent sampling interval. If output has been configured to include host labels, the average number of characters sent by each host is provided separately.
`iris_interop_avg_http_ttfc {id="namespace",host="host",production="production"}`	*Time to First Character* (TTFC): average length of time between the start of an HTTP or SOAP request and the first character of the corresponding response, in seconds. If output has been configured to include host labels, the TTFC for each host is provided separately
`iris_interop_avg_http_ttlc {id="namespace",host="host",production="production"}`	*Time to Last Character* (TTLC): average length of time between the start of an HTTP or SOAP request and the last character of the corresponding response. If output has been configured to include host labels, the TTLC for each host is provided separately.
`iris_interop_http_sample_count {id="namespace",host="host",production="production"}`	Number of HTTP or SOAP transmissions sent within the *production* and *namespace* over the most recent sampling interval. If output has been configured to include host labels, the number of transmissions sent by each host is provided separately.
`iris_interop_http_sample_count_per_sec {id="namespace",host="host",production="production"}`	Number of HTTP or SOAP transmissions sent per second within the *production* and *namespace*, averaged over the most recent sampling interval. If output has been configured to include host labels, the number of transmissions sent by each host per second is provided separately.

C.2.3 Create Application Metrics

To add custom application metrics to those returned by the /metrics endpoint:

1. Create a new class that inherits from %SYS.Monitor.SAM.Abstract.
2. Define the `PRODUCT` parameter as the name of your application. This can be anything except for `iris`, which is reserved for the InterSystems IRIS metrics.
3. Implement the **GetSensors()** method to define the desired custom metrics, as follows:

- The method must contain one or more calls to the **SetSensor()** method. This method sets the name and value for an application metric. The values should be integers or floating point numbers to ensure compatibility with Prometheus and InterSystems SAM.

  You can optionally define a label for the metric, though if you do, you must always define a label for that particular metric.

  Note: For best practices when choosing metric and label names, see *Metric and Label Naming* in the Prometheus documentation (https://prometheus.io/docs/practices/naming/).

- The method must return `$$$OK` if successful.

Important: A slow implementation of **GetSensors()** can negatively impact system performance. Be sure to test that your implementation of **GetSensors()** is efficient, and avoid implementations that could time out or hang.

4. Compile the class. An example is shown below:

Class Definition

```
/// Example of a custom class for the /metric API
Class MyMetrics.Example Extends %SYS.Monitor.SAM.Abstract
{

Parameter PRODUCT = "myapp";

/// Collect metrics from the specified sensors
Method GetSensors() As %Status
{
   do ..SetSensor("my_counter",$increment(^MyCounter),"my_label")
   do ..SetSensor("my_gauge",$random(100))
   return $$$OK
}

}
```

5. Use the **AddApplicationClass()** method of the SYS.Monitor.SAM.Config class to add the custom class to the /metrics configuration. Pass as arguments the name of the class and the namespace where it is located.

   For example, enter the following in the Terminal from the %SYS namespace:

```
%SYS>set status = ##class(SYS.Monitor.SAM.Config).AddApplicationClass("MyMetrics.Example", "USER")

%SYS>w status
status=1
```

   Note: When you upgrade your InterSystems IRIS system, you will need to redo this step.

6. Ensure that /api/monitor web application has the necessary Application Roles to access the custom metrics. For details on how to edit application roles, see Edit an Application: The Application Roles Tab.

   This step grants /api/monitor access to the data needed for the custom metric. For example, if the custom metric class is located in the USER database (protected by the %DB_USER resource), grant /api/monitor the %DB_USER role.

7. Review the output of the /metrics endpoint by pointing your browser to a URL with the following form, using the <baseURL> for your system: *http://<baseURL>/api/monitor/metrics*. The metrics you defined should appear after the InterSystems IRIS metrics, such as:

```
[...]
myapp_my_counter{id="my_label") 1
myapp_my_gauge 92
```

The /metrics endpoint now returns the custom metrics you defined. The InterSystems IRIS metrics include an `iris_` prefix, while your custom metrics use the value of `PRODUCT` as a prefix.

C.3 /api/monitor/alerts

The /api/monitor/alerts endpoint fetches the most recent alerts from the alerts.log file and returns them in JSON format, such as:

```
{"time":"2019-08-15T10:36:38.313Z","severity":2,\
"message":"Failed to allocate 1150MB shared memory using large pages.  Switching to small pages."}
```

When /api/monitor/alerts is called, it returns the alerts that have been generated since the previous time /api/monitor/alerts was called. The `iris_system_alerts_new` metric is a Boolean that indicates whether new alerts have been generated.

For more information about when and how alerts are generated, see Using Log Monitor.

C.4 See Also

- Securing REST Services
- Introduction to Creating REST Services
- Developing Rest Interfaces
- https://prometheus.io/docs/

D

Monitoring InterSystems IRIS Using the irisstat Utility

This topic provides an overview of how to use the **irisstat** utility. It is intended as an introduction for new users and a reference for experienced users.

Important: When using this utility, you should consult with the InterSystems Worldwide Response Center (WRC) for guidance about specifying appropriate **irisstat** options and assistance in interpreting the data produced by the utility.

irisstat is a C executable that is distributed with InterSystems IRIS® data platform. It is a diagnostic tool for system level problems, including InterSystems IRIS hangs, network problems, and performance issues. When run, **irisstat** attaches to the shared memory segment allocated by InterSystems IRIS at start time, and displays an InterSystems IRIS instance's internal structures and tables in a readable format. The shared memory segment contains the global buffers, lock table, journal buffers, and a wide variety of other memory structures which need to be accessible to all InterSystems IRIS processes. Processes also maintain their own process private memory for their own variables and stack information. The basic display-only options of **irisstat** are fast and non-invasive to InterSystems IRIS.

CAUTION: More advanced (undocumented) options may alter shared memory and should be used with care. These advanced options should be used only at the direction of InterSystems Support personnel; for information, contact the InterSystems Worldwide Response Center (WRC).

D.1 Basics of Running irisstat

In the event of a system problem, the **irisstat** report is often the most important tool that InterSystems has to determine the cause of the problem. Use the following guidelines to ensure that the **irisstat** report contains all of the necessary information:

- Run **irisstat** at the time of the event.
- Use the Diagnostic Report task or IRISHung script unless directed otherwise by InterSystems support personnel.
- Check the contents of the **irisstat** report to ensure it is valid.

Since **irisstat** is a separate executable file included with InterSystems IRIS, it is run outside of InterSystems IRIS, at an operating system prompt. Therefore, the details of running it depend on the operating system:

- Running irisstat on Windows

- Running irisstat on UNIX®

Running **irisstat** with no options is not a common way to run it, but doing so produces a basic report which is the equivalent of running it with the following default options:

- `-f` (global module flags)
- `-p` (PID table)
- `-q` (semaphores)

For information about **irisstat** options, see Running irisstat with Options.

D.1.1 Running irisstat on Windows

The **irisstat** executable is located in the *install-dir*\bin directory. Starting with a Windows command prompt running as Administrator, you can run it as follows:

```
C:\>cd install-dir\bin
C:\install-dir\bin>irisstat
```

If you run **irisstat** from a directory other than *install-dir*\bin or *install-dir*\mgr, you must include the `-s` argument to specify the location of the *install-dir*\mgr directory. For example:

```
C:\Users>\install-dir\bin\irisstat -s\install-dir\mgr
```

D.1.2 Running irisstat on UNIX®

The **irisstat** executable is located in the *install-dir*/bin directory. If you run **irisstat** from a directory other than *install-dir*\bin or *install-dir*\mgr, you must include the `-s` argument to specify the location of the *install-dir*\mgr directory.

Starting with a UNIX® command prompt running as root, change to the *install-dir*/bin directory or the *install-dir*/mgr directory and run the **irisstat** command:

```
bash-3.00$ ./irisstat
```

From the InterSystems IRIS installation directory, the command would be as follows:

```
bash-3.00$ ./bin/irisstat -smgr
```

You can also invoke **irisstat** via the **iris** command, which can be run from any directory as shown in the following example:

```
bash-3.00$ iris stat iris_instance_name
```

where *iris_instance_name* is the name of the InterSystems IRIS instance on which you are running **irisstat**.

D.2 Running irisstat with Options

Running **irisstat** without options produces a basic report. Generally, you run **irisstat** to obtain specific information. To specify target information, you can include or exclude options as follows:

- To include (turn on) an option, specify a flag followed by a `1` (or other level).
- To exclude (turn off) an option, specify a flag followed by a `0`.

For example, to include the Global File Table (GFILETAB) section in the **irisstat** report, use the `-m1` option:

```
C:\iris-install-dir\Bin\irisstat -m1
```

or, to turn off the default basic options, use the `-a0` option:

```
C:\iris-install-dir\bin\irisstat -a0
```

Many options have more detailed levels than `0` and `1`. These additional levels are described as having bits, which are displayed in decimal as powers of two and control specific types of information about the option. For example, the basic `-p` option, which displays the PID table, is turned on with a `1`; however, using a `2` adds a **swcheck** column, a `4` adds a **pstate** column. and so on. These bits can be combined; for example, if you want to see the information displayed by both the `2` and `4` bits, specify `-p6`. To ask for all bits, use `-1`, as follows:

```
bash-3.00$ ./irisstat -p-1
```

In addition, multiple flags can be combined in a single **irisstat** command. For example, the following command turns off the basic options, then turns on all bits for the global module flags and PID table, as well as a detailed level for the GFILETAB:

```
bash-3.00$ ./irisstat -a0 -f-1 -p-1 -m3
```

It is common for **irisstat** commands to have many flags when you start diagnosing a complex problem; however, the options that make modifications are typically used alone. For example, the `-d` option requests a process dump; before using this option, you might run **irisstat** with multiple options to identify the process to dump, but when using `-d`, typically no other options are selected.

The irisstat Options table describes the options that you can use with the **irisstat** command.

Note: For assistance in interpreting the data produced by the **irisstat** options described in this table, contact the InterSystems Worldwide Response Center (WRC).

Table IV–1: irisstat Options

Option	Description
–a[0/1]	Displays all information described in this table.
–b[bits]	Displays information about global buffer descriptors blocks (BDBs). You can specify a combination of the following bits: • `1` (all) • `2` (cluster) • `4` (ECP server) • `8` (ECP client) • `16` (block contents) • `64` (check block integrity) • `128` (block and LRU summary) **Note:** Running **irisstat -b64** may require extra time.

Option	Description
`-c[bits]`	Displays counters, which are statistics on system performance. You can specify a combination of the following bits: • `1` (global) • `2` (network) • `4` (lock) • `8` (optim) • `16` (terminal) • `32` (symtab) • `64` (journal) • `128` (disk i/o) • `256` (cluster) • `262144` (bshash) • `2097152` (job cmd) • `4194304` (sem) • `8388608` (async disk i/o) • `16777216` (fsync) • `33554432` (obj class) • `67108864` (wd) • `134217728` (bigstr) • `268435456` (swd) • `536870912` (sort) • `1073741824` (symsave) • `2147483648` (freeblkpool)
`-d[pid,opt]`	Creates dump of InterSystems IRIS processes. You can specify the following options: • `0` (full); default • `1` (partial)
`-e[0/1/2]`	Displays the InterSystems IRIS system error log (see InterSystems IRIS System Error Log); `-e2` displays additional process information (in hex).

Option	Description
`-f[bits]`	Displays global module flags. You can specify a combination of the following bits: • `1` (basic) • `64` (resources) • `128` (with detail) • `256` (account detail) • `512` (incstrtab) • `1024` (audit)
`-g[0/1]`	Displays **^GLOSTAT** information; for information see Gathering Global Activity Statistics Using ^GLOSTAT.
`-h`	Displays **irisstat** usage information.
`-j[0/1/2/3/4/5/6]`	Displays the journal system master structure, which lists information about journaling status. `-j32` displays mirror server information.
`-k`	Displays information about prefetch daemons used by the **$PREFETCHON** function; see $PREFETCHON
`-l[bits]`	Displays information about least recently used (LRU) global buffer descriptor block (BDB) queue, but not the contents of the BDBs. You can specify a combination of the following bits: • `1` (all) • `2` (cluster) • `4` (ECP server) • `8` (ECP client) • `16` (block contents) • `32`, but not `1` (most recently used (MRU) order) **Note:** See also `-b`.
`-m[0/1/3/4/8/16]`	Displays Global File Table (GFILETAB), which contains information about all databases, listed by SFN, that have been mounted since the instance of InterSystems IRIS started up. You can specify a combination of the following bits: • `3` (additional details) • `4` (volume queues) • `8` (disk device id table) • `16` (systems remotely mounting this database)
`-n[0/1]`	Displays information about network structures and local/remote SFN translations; `irisstat -n-1` also displays namespace structures.

Option	Description
-o1	Clears the resource statistics displayed by **irisstat -c** to reestablish a base situation without rebooting InterSystems IRIS. No output is produced.
-p[bits]	Displays information about processes that are running in InterSystems IRIS. The information is obtained from the process ID table (PIDTAB). You can specify a combination of the following flags: • 2 (swcheck) • 4 (pstate and %SS) • 5 (NT mailbox locks); Windows only • 8 (js sum) • 16 (js list) • 32 (grefcnt info) • 64 (gstatebits) • 128 (gstate summary) • 256 (jrnhib) • 512 (transaction summary) • 1024 (pidflags) • 2048 (pgbdbsav); additionally dumps pgshared table • 4096 (freeblk table)
-q[0/1]	Displays information about hibernation semaphores.
-s[dir]	Specifies the directory containing the **irisstat** executable when running the command from other than the mgr or bin directories.
-t[seconds]	Runs **irisstat** repeatedly in a loop every *seconds* seconds until halted. Only the global module flags section is displayed, as when -f1 is specified.
-u[bits]	Displays information about InterSystems IRIS locks stored in the lock table (see Monitoring Locks). You can specify a combination of the following bits: • 1 (summary) • 2 (waiters) • 4 (intermediate) • 8 (detail) • 16 (watermark) • 32 (buddy memory) • 64 (resource info)
-v1	Ensures that the InterSystems IRIS executable associated with the shared memory segment **irisstat** is being run on and the **irisstat** executable are from the same version; if not, **irisstat** will not run.

Option	Description
`-w[bits]`	Displays information about BDBs in write daemon queues.
`-B[0/1]`	Displays, in hex, the contents of blocks held in GBFSPECQ.
`-C[0/1]`	Displays configuration information for inter-job communication (IJC) devices.
`-D[secs],[msecs][,0]`	Displays resource statistics over an interval of 'secs' seconds. Sample block collisions ever 'msec' milliseconds. **Note:** Resource information same as `-c`. The **^BLKCOL** utility, described in Monitoring Block Collisions Using ^BLKCOL, provides more detailed information about block collisions.
`-E[bits]`	Displays status of cluster on platforms that support clustering. You can specify a combination of the following bits: • `1` (vars) • `2` (write daemon locks) • `4` (enqinuse) • `8/16` (allenq)
`-G[bdb]`	Displays, in hex, the contents of the global buffer descriptors and the global buffer for a specific buffer descriptor block (BDB). **Note:** Same as `-H` except that the information is displayed by BDB.
`-H[sfn],[blk]`	Displays, in hex, the contents of the global buffer descriptors and the global buffer for a specific system file number (sfn) and block number (blk) pair. **Note:** Same as `-G` except that the information is displayed by system file number and block number pair. The block must be in the buffer pool.
`-I[0/1]`	Displays the incremental backup data structures.
`-L[0/1]`	Displays the license. **Note:** Same as **^CKEY** and **%SYSTEM.License.CKEY** method.
`-M[0/1]`	Displays the mailbox log. **Note:** Disabled by default. A special build is required to capture and log the mailbox messages; additional logging may be required.

Option	Description
`-N[value]`	Displays ECP network information. You can specify a combination of the following values: • `1` (client) • `2` (server) • `4` (client buffers) • `8` (server buffers) • `16` (client buffers, in detail) • `32` (user jobs awaiting answer) • `64` (server answer buffers details) • `128` (request global) • `256` (server send answer buffer details; not -1) • `1024` (dump server received request buffers) • `2048` (client trans bitmap) • `4096` (client GLO Q) • `8192` (request global reference dump, in hex) • `65536` (ECP blocks downloaded to clients) • `131072` (client released request buffer details; not -1)

Option	Description
`-R[value]`	Displays information about routine buffers in use (or changing), class control blocks (CCB), and least recently used (LRU) queues. You can specify a combination of the following values: • `1` (routine buffers in use) • `4` (RCT – changed routine table) • `8` (RCT detail) • `16` (`0x10`=all routine buffers) • `32` (`0x20`=LRU Q) • `64` (`0x40`=all CCB's) • `128` (`0x80`=invalidated CCB's) • `0x100` (invalidated subclasses) • `0x200` (buffer address) • `0x400` (buffer descriptors) • `0x800` (procedure table and cached routines buffer number) • `0x1000` (process cached routine names) • `0x2040` (CCB's and CCB details • `0x4000` (cls NS cache) • `0x6000` (cls NS cache details) • `0x8000` (validate shm cls cache) • `0x10000` (dump all class hierarchy) • `0x20000` (dump all class hierarchy details) • `0x40000` (dump process class and routine statistics) • `0x80000` (process cached class names)
`-S[bits]`	Displays information about the cause of a hang based on a self diagnosis of whether or not the system is hung. You can specify a combination of the following bits: • `1` (display diagnosis) • `2` (partial process dump for suspect jobs) • `4` (full process dump for first suspect job and partial dumps for other suspect jobs) **Note:** In a cluster, this option should be run all cluster members.
`-T[0/1]`	Displays hex values of many in-memory tables, including National Language Settings (NLS) tables.

Option	Description
`-V[pid]`	Displays variables that are part of the process memory structures; of limited value unless you have access to the source code. **Note:** Windows only. Run from the directory that contains the pid.dmp file.
`-W`	Performs the same function as the **Backup.General.ExternalThaw()** classmethod, and may be used to resume the write daemon after **Backup.General.ExternalFreeze()** has been called in cases when a new InterSystems IRIS session cannot be started. (See External Backup for information on the use of these methods.) This option will not unfreeze the write daemon from any hang or suspension caused by anything other than a backup. Use of this option is recorded in the messages log.
`-X[0/1]`	Displays the contents of the device translation table. It is organized by device number and shows both the numeric and plaintext class identifiers.

D.3 Viewing irisstat Output

irisstat data can be viewed immediately (via a terminal) or redirected to an output file for later analysis. The most common methods for viewing the data are:

- irisstat Text File
- Diagnostic Report Task
- IRISHung Script
- ^SystemPerformance Utility

Note: When InterSystems IRIS is forcibly shut down, **irisstat** is run in order to capture the current state of the system. The output is added to the messages log as part of the emergency shutdown procedure.

D.3.1 irisstat Text File

irisstat reports can be redirected to a file instead of the terminal, which might be useful if you want to collect a set of **irisstat** options that are not provided by one of the InterSystems IRIS tools (Diagnostic Report Task, IRISHung Script, ^SystemPerformance Utility) or if you are having trouble running those tools.

D.3.2 Diagnostic Report Task

The Diagnostic Report task creates an HTML log file containing both basic and advanced information, which can be used by the InterSystems Worldwide Response Center (WRC) to resolve system problems. For information about the Diagnostic Report task, including the **irisstat** options that it uses, see Using the InterSystems Diagnostic Report.

Note: The Diagnostic Report task cannot be run on a hung system; if your system is hung, see IRISHung Script.

D.3.3 IRISHung Script

The **IRISHung** script is an OS tool used to collect data on the system when an InterSystems IRIS instance is hung. The name of the script, which is located in the *install-dir*\bin directory, is platform-specific, as specified in the following table:

Platform	Script name
Microsoft Windows	**IRISHung.cmd**
UNIX®/Linux	**IRISHung.sh**

The **IRISHung** script should be run with Administrator privileges. Like the Diagnostic Report Task, the **IRISHung** script runs **irisstat** twice, 30 seconds apart, in case the status is changing, and bundles the reports into an html file together with the other collected data. The **irisstat** reports taken from **IRISHung** use the following options:

```
irisstat -e2 -f-1 -m-1 -n3 -j5 -g1 -L1 -u-1 -v1 -p-1 -c-1 -q1 -w2 -E-1 -N65535
```

IRISHung also runs a third **irisstat** using only the `-S2` option, which it writes to a separate section of output called Self-Diagnosis. The `-S2` option causes suspect processes to leave mini-dumps; therefore, running **IRISHung** is likely to collect information about the specific processes responsible for the hang, whereas simply forcing the instance down does not collect this information.

In addition, **IRISHung** generates **irisstat** output files that are often very large, in which case they are saved to separate .txt files. Remember to check for these files when collecting the output.

D.3.4 ^SystemPerformance Utility

The **^SystemPerformance** utility collects detailed performance data about an InterSystems IRIS instance and the platform on which it is running. It runs inside InterSystems IRIS for a configurable amount of time, collects samples over the that interval, and generates a report when it finishes. For information about the **^SystemPerformance** utility, including the **irisstat** options that it uses, see Monitoring Performance Using ^SystemPerformance.