# Table of Contents

## About This Book

---

### 1 Monitoring InterSystems IRIS Using the Management Portal

1.1 Monitoring System Dashboard Indicators ................................................................. 3  
1.2 Monitoring System Performance ............................................................................... 6  
  
1.2.1 System Usage Table ............................................................................................. 6  
1.2.2 Generic (Shared) Memory Heap Usage ............................................................... 6  
1.3 Monitoring Locks ...................................................................................................... 8  
1.4 Monitoring InterSystems IRIS Logs ......................................................................... 9  
  
1.4.1 Log Files in the install-dir\mgr Directory ........................................................... 9  
1.4.2 Application and xDBC Error Logs ................................................................. 10  
1.4.3 InterSystems IRIS System Error Log .............................................................. 10  

### 2 Using the InterSystems Diagnostic Report

2.1 Running the Diagnostic Report Task .......................................................................... 13  
2.2 Configuring Diagnostic Report Settings ..................................................................... 13  
2.3 Diagnostic Report Contents ...................................................................................... 15  
  
2.3.1 Basic Information ................................................................................................. 15  
2.3.2 Advanced Information ......................................................................................... 17  

### 3 Using Log Monitor

3.1 System Monitoring Tools ......................................................................................... 19  
3.2 Log Monitor Overview ............................................................................................. 19  
3.3 Using the *MONMGR Utility .................................................................................. 20  
  
3.3.1 Start/Stop/Update Monitor .................................................................................. 20  
3.3.2 Manage Monitor Options .................................................................................... 21  
3.3.3 Manage Email Options ....................................................................................... 22  
3.4 Log Monitor Errors and Traps ................................................................................ 23  

### 4 Using System Monitor

4.1 System Monitor ........................................................................................................ 25  
  
4.1.1 The System Monitor Process .............................................................................. 26  
4.1.2 Tracking System Monitor Notifications .............................................................. 27  
4.1.3 System Monitor Status and Resource Metrics .................................................... 27  
4.1.4 System Monitor Health State ............................................................................. 30  
4.1.5 System Monitor Defaults ................................................................................... 31  
4.1.6 Using the ^%SYSMONMGR Utility .................................................................... 32  
4.1.7 Defining System Monitor Components .............................................................. 36  
4.2 Health Monitor ....................................................................................................... 36  
  
4.2.1 Health Monitor Overview ................................................................................... 37  
4.2.2 Using ^%SYSMONMGR to Manage Health Monitor .......................................... 46  
4.3 Application Monitor ............................................................................................... 48  
  
4.3.1 Application Monitor Overview .......................................................................... 48  
4.3.2 Using ^%SYSMONMGR to Manage Application Monitor .................................. 49  
4.3.3 Application Monitor Metrics ............................................................................ 56  
4.3.4 Writing User-Defined Application Monitor Classes ......................................... 58  

### 5 Gathering Global Activity Statistics Using ^GLOSTAT

5.1 Running ^GLOSTAT ............................................................................................... 61  
5.2 Overview of ^GLOSTAT Statistics ......................................................................... 62
List of Tables

Table 1–1: System Performance Indicators ........................................................................................................ 3
Table 1–2: ECP Indicators .................................................................................................................................. 4
Table 1–3: System Time Indicators .................................................................................................................. 4
Table 1–4: System Usage Indicators ................................................................................................................ 4
Table 1–5: Errors and Alerts Indicators .......................................................................................................... 5
Table 1–6: Licensing Indicators ....................................................................................................................... 5
Table 1–7: Task Manager Upcoming Tasks ...................................................................................................... 5
Table 1–8: System Usage Statistics ................................................................................................................. 6
Table 1–9: Shared Memory Heap Usage ......................................................................................................... 7
Table 4–1: System Monitor Status and Resource Notifications ........................................................................ 28
Table 4–2: System Monitor Health State ......................................................................................................... 30
Table 4–3: Health Monitor Sensor Objects .................................................................................................... 38
Table 4–4: Default Health Monitor Periods .................................................................................................... 43
Table 4–5: Responses to Alert Prompts ......................................................................................................... 55
Table 4–6: Application Monitor System Classes .......................................................................................... 56
Table 5–1: Statistics Produced by ^GLOSTAT .............................................................................................. 62
Table 6–1: Metrics for Custom ^PERFMON Report ...................................................................................... 70
Table 10–1: InterSystems IRIS Performance Data Report for Microsoft Windows Platforms ..................... 104
Table 10–2: InterSystems IRIS Performance Data Report for Apple macOS Platforms .............................. 107
Table 10–3: InterSystems IRIS Performance Data Report for IBM AIX® Platforms .................................. 108
Table 10–4: InterSystems IRIS Performance Data Report for Red Hat Linux/SuSE Linux Platforms ....... 110
Table I–1: InterSystems IRIS SNMP Notification Objects (Traps) ............................................................... 128
Table I–2: InterSystems IRIS-specific Auxiliary Objects Sent in Traps ....................................................... 129
Table IV–1: irisstat Options .......................................................................................................................... 151
About This Book

This book describes several tools available for monitoring InterSystems IRIS® data platform.

The following chapters describe how to monitor InterSystems IRIS with tools included with InterSystems IRIS:

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

The following chapters describe how to monitor InterSystems IRIS with system routines:

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

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

- **Monitoring InterSystems IRIS Using SNMP**
- **Monitoring InterSystems IRIS Using Web Services**
- **Monitoring InterSystems IRIS Using REST API**
- **Monitoring InterSystems IRIS Using the irisstat Utility**

For detailed information, see the Table of Contents.
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.

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

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globals/Second</td>
<td>Most recently measured number of global references per second.</td>
</tr>
<tr>
<td>Global Refs</td>
<td>Number of global references since system startup.</td>
</tr>
<tr>
<td>Global Sets</td>
<td>Number of global Set and Kill operations since system startup.</td>
</tr>
<tr>
<td>Routine Refs</td>
<td>Number of routine loads and saves since system startup.</td>
</tr>
</tbody>
</table>
### Table 1–2: ECP Indicators

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Servers</td>
<td>Summary status of ECP (Enterprise Cache Protocol) application servers connected to this system.</td>
</tr>
<tr>
<td>Application Server Traffic</td>
<td>Most recently measured ECP application server traffic in bytes per second.</td>
</tr>
<tr>
<td>Data Servers</td>
<td>Summary status of ECP data servers to which this system is connected.</td>
</tr>
<tr>
<td>Data Server Traffic</td>
<td>Most recently measured ECP data server traffic in bytes per second.</td>
</tr>
</tbody>
</table>

For more information on the ECP indicators, see the “Horizontally Scaling Systems for User Volume with InterSystems Distributed Caching” chapter of the Scalability Guide.

### Table 1–3: System Time Indicators

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Up Time</td>
<td>Elapsed time since this system was started.</td>
</tr>
<tr>
<td>Last Backup</td>
<td>Date and time of last system backup.</td>
</tr>
</tbody>
</table>

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

### Table 1–4: System Usage Indicators

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Space</td>
<td>Indicates whether there is a reasonable amount of disk space available for database files. Clicking ... more details displays the System Operation &gt; Databases page.</td>
</tr>
<tr>
<td>Database Journal</td>
<td>Indicates the current status of the database journal. Clicking ... more details displays the System Operation &gt; Journals page.</td>
</tr>
<tr>
<td>Journal Space</td>
<td>Indicates whether there is a reasonable amount of disk space available for journal files. Clicking ... more details displays the System Operation &gt; Journals page.</td>
</tr>
<tr>
<td>Journal Entries</td>
<td>Number of entries written to the system journal. Clicking ... more details displays the System Operation &gt; Journals page.</td>
</tr>
<tr>
<td>Lock Table</td>
<td>Current status of the system Lock Table. Clicking ... more details displays the System Operation &gt; Locks &gt; Manage Locks page.</td>
</tr>
<tr>
<td>Write Daemon</td>
<td>Current status of the system Write daemon.</td>
</tr>
</tbody>
</table>
**Definition Indicator**

Current status of open local and remote (ECP) transactions. If there are no open transactions, status is **Normal**; status may also be **Warning** (if the duration of the longest open local or remote transaction is greater than 10 minutes) and **Troubled** (if greater than 20 minutes). Clicking **more details** displays the **Transactions** page.

**Processes**

Most recent number of running processes. Clicking **more details** displays the **Processes** page (**System Operation > Processes**).

**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**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serious Alerts</td>
<td>Number of serious alerts that have been raised. Clicking <strong>more details</strong> displays the <strong>View Messages Log</strong> page (<strong>System Operation &gt; System Logs &gt; Messages Log</strong>).</td>
</tr>
<tr>
<td>Application Errors</td>
<td>Number of application errors that have been logged. Clicking <strong>more details</strong> displays the <strong>View Application Error Log</strong> page (<strong>System Operation &gt; System Logs &gt; Application Error Log</strong>).</td>
</tr>
</tbody>
</table>

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

**Table 1–6: Licensing Indicators**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>License Limit</td>
<td>Maximum allowed license units for this system.</td>
</tr>
<tr>
<td>Current License Use</td>
<td>License usage as a percentage of available license units.</td>
</tr>
<tr>
<td>Highest License Use</td>
<td>Highest license usage as a percentage of available license units.</td>
</tr>
</tbody>
</table>

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

**Table 1–7: Task Manager Upcoming Tasks**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upcoming Tasks</td>
<td>Lists the next five tasks scheduled to run.</td>
</tr>
<tr>
<td>Task</td>
<td>Name of the upcoming task.</td>
</tr>
<tr>
<td>Time</td>
<td>Time the task is scheduled to run.</td>
</tr>
<tr>
<td>Status</td>
<td>Task status—one of: scheduled, completed, running.</td>
</tr>
</tbody>
</table>

Click the **more details** link in the bottom details box to display the **System Operation > Task Manager > Upcoming Tasks** page. For details on the Task Manager, see the **Using the Task Manager** section of the “Managing InterSystems IRIS” chapter of the **System Administration Guide**.
1.2 Monitoring System Performance

System performance metrics are described in the following tables:

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

1.2.1 System Usage Table

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

**Table 1–8: System Usage Statistics**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global references (all)</td>
<td>Logical count of accesses to globals, including Sets, Kills, $Data, $Order, $Increment, $Query, and global references in expressions.</td>
</tr>
<tr>
<td>Global update references</td>
<td>Logical count of global references that are Set, Kill, or $Increment operations.</td>
</tr>
<tr>
<td>Routine calls</td>
<td>Number of calls to a routine.</td>
</tr>
<tr>
<td>Routine buffer loads and saves</td>
<td>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).)</td>
</tr>
<tr>
<td>Logical block requests</td>
<td>Number of database blocks read by the globals database code. (In a well-tuned environment, many of these reads are satisfied without disk access.)</td>
</tr>
<tr>
<td>Block reads</td>
<td>Number of physical database blocks read from disk for both global and routine references.</td>
</tr>
<tr>
<td>Block writes</td>
<td>Number of physical database blocks written to disk for both global and routine references.</td>
</tr>
<tr>
<td>WIJ writes</td>
<td>Number of blocks written to the write image journal file.</td>
</tr>
<tr>
<td>Journal entries</td>
<td>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.</td>
</tr>
<tr>
<td>Journal block writes</td>
<td>Number of 64-KB journal blocks written to the journal file.</td>
</tr>
<tr>
<td>Routine lines</td>
<td>Number of routine lines executed since system startup.</td>
</tr>
<tr>
<td>Last update</td>
<td>Date and time stamp of the displayed statistics.</td>
</tr>
</tbody>
</table>

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

1.2.2 Generic (Shared) Memory Heap Usage

To view the InterSystems IRIS’s generic memory heap (gmheap) usage, referred to on this page as shared memory heap (SMH) usage, navigate to the System Usage page (System Operation > System Usage), and click the Shared Memory Heap Usage link.
To learn how to change the size of the generic memory heap or gmheap (sometimes known as the shared memory heap or SMH), see the Changing This Parameter section of the “gmheap” entry in Configuration Parameter File Reference.

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

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

### Table 1–9: Shared Memory Heap Usage

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miscellaneous</td>
<td>Shared memory allocated for the static memory table (SMT) and general string table (GST).</td>
</tr>
<tr>
<td>Audit System</td>
<td>Shared memory used for system auditing.</td>
</tr>
<tr>
<td>Classes Instantiated</td>
<td>Shared memory allocated/available/used for the class hash table and control blocks.</td>
</tr>
<tr>
<td>Database Encryption Key Change</td>
<td>Shared memory allocated/available/used for database encryption key changes.</td>
</tr>
<tr>
<td>Semaphore Objects</td>
<td>Shared memory allocated/available/used for semaphore objects.</td>
</tr>
<tr>
<td>Event System</td>
<td>Shared memory allocated/available/used for the event system.</td>
</tr>
<tr>
<td>Global Mapping</td>
<td>Shared memory allocated/available/used for global mapping and subscript-level mapping (SLM).</td>
</tr>
<tr>
<td>License Upgrade</td>
<td>Shared memory allocated/available/used for license upgrades.</td>
</tr>
<tr>
<td>Lock Table</td>
<td>Shared memory allocated/available/used for the lock system.</td>
</tr>
<tr>
<td>Performance Monitor</td>
<td>Shared memory allocated/available/used for the Performance Monitor (^PERFMON).</td>
</tr>
<tr>
<td>Process Table</td>
<td>Shared memory allocated/available/used for the Process ID (PID) table.</td>
</tr>
<tr>
<td>Routine Buffer in Use Table</td>
<td>Shared memory allocated/available/used for routine buffer-in-use tables.</td>
</tr>
<tr>
<td>Security System</td>
<td>Shared memory allocated/available/used for the security system.</td>
</tr>
<tr>
<td>Shared Library</td>
<td>Shared memory allocated/available/used for shared libraries.</td>
</tr>
<tr>
<td>TTY Hash Table</td>
<td>Shared memory allocated/available/used for TTY hash tables.</td>
</tr>
<tr>
<td>DB Name &amp; Directory</td>
<td>Shared memory allocated/available/used for database names and directories.</td>
</tr>
<tr>
<td>iKnow Language Model Data</td>
<td>Shared memory allocated/available/used for iKnow language models.</td>
</tr>
</tbody>
</table>
### 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. The article *Locking and Concurrency Control* discusses InterSystems IRIS locks in detail.

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.

The Lock Table has the following column entries.

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

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

For a more in-depth description of the LOCK command and its features, see the LOCK entry of the ObjectScript Reference. You may need to enlarge the size of the lock table if your system uses a large number of locks. You can do this using the Management Portal; for instructions, see the locksiz entry in Configuration Parameter File Reference.

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

### 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 xDBC 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 the article “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 the “Using Log Monitor” chapter of this guide for more information.

**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 the “Journaling” chapter of the Data Integrity Guide for information about 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 in the Configuration Parameter File Reference), 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 in the Configuration Parameter File Reference 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 the chapter “Using System Monitor” in this guide) are written to the System Monitor log, install-dir\mgr\SystemMonitor.log.

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

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 xDBC 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 ODBC and JDBC 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 in Configuration Parameter File Reference.

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 the “Using the Diagnostic Report” chapter of this guide.

- Run the `irisstat` command with the `-e1` option, as described in Running `irisstat with Options` in the “Monitoring InterSystems IRIS Using the `irisstat Utility” appendix of this guide.

- Run the `IRISHung` script, as described in `IRISHung Script` in the “Monitoring InterSystems IRIS Using the `irisstat Utility” appendix.

You can configure InterSystems IRIS to write the system error log to the `messages log` during shutdown using the `ShutDownLogErrors` setting (see `ShutDownLogErrors` in `Configuration Parameter File Reference`).
Using the InterSystems Diagnostic Report

InterSystems provides a mechanism to run a Diagnostic Report on your InterSystems IRIS® data platform instance and send the results to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems.

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, 2019 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. The Configuring Diagnostic Report Settings section describes these settings. If you click Close, your changes are discarded and the report task does not run.

Viewing the Diagnostic Report Task History

Click Task History at the top of the Diagnostic Report page to display the history for the Diagnostic Report task. (See Using the Task Manager in the “Managing InterSystems IRIS” chapter of the System Administration Guide 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 `WRCHChealthCheck@InterSystems.com`.

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

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

- **Name of IP address of server for outgoing mail** — address of your outgoing SMTP (Simple Mail Transfer Protocol) mail server.

- **Username for authenticated SMTP and Password** — only required for SMTP authentication with the SMTP server. See RFC 2554 for details.

- **Address for the “From:” field in outgoing mail** — email address to appear in the sender field. Required if you enter SMTP server information; defaults to `DefaultDiagnosticReport@InterSystems.com`.

- **Address for the “Reply-To:” field in outgoing mail** — a valid email address at your company able to receive automated configuration messages from InterSystems.

- **Addresses for the “CC:” field in outgoing mail** — additional email addresses to receive the report.

- **Enable automatic WRC HealthCheck updates** — select this check box to send periodic reports to the WRC. InterSystems highly recommends that you enable the WRC HealthCheck feature. If selected, the Diagnostic Report task runs at regular intervals and sends the report to the WRC. These regular reports allow the WRC to better assist you. Selecting this feature requires you to enter the SMTP server information.

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

- **Run the automatic WRC HealthCheck updates every** number of **days at this time** — if you enable WRC HealthCheck, the task manager saves the frequency (defaults to 7 days) and time (defaults to the 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 in the “Monitoring InterSystems IRIS Using the Management Portal” chapter for more information.

**Security**
Displays a listing of the following security information:
- Security parameters
- Services
- Resources
- Roles
- Applications
- System users
- Current login failures
- Domains
- SSL configurations

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

**messages**
Displays the contents of the messages.log (if its size does not exceed 5MB).
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 -q1 -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 the Monitoring InterSystems IRIS Using the irisstat Utility appendix of this guide.

**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).
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. This chapter discusses the following topics:

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 the “Monitoring InterSystems IRIS Using the Management Portal” chapter of this guide. 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 chapter, 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.

- 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. System Monitor (including Application Monitor and Health Monitor) is managed using the ^%SYSMONMGR utility. See the “System Monitor” chapter of this guide for detailed information about using System Monitor, Application Monitor and Health 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.

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

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

### 3.3 Using the ^MONMGR Utility

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

1. To start the Log Monitor Manager, enter the following command in the Terminal:
   ```
   ^SYS>do ^MONMGR
   ```

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

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

### 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:

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

### 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Set Monitor Interval</td>
<td>Lets you change the interval at which the messages log is scanned. InterSystems recommends an interval no longer than the default of 10 seconds.</td>
</tr>
</tbody>
</table>
| 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. |
| 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
| 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 in the “Monitoring InterSystems IRIS Using the Management Portal” chapter) or switch journaling to a new device.
- Error logging in journal
- Journaling Error x reading attributes after expansion
- ECP client daemon/connection is hung
- Cluster Failsoft failed, couldn't determine locksysid for failed system - all cluster systems are suspended
- enqpijstop failed, declaring a cluster crash
- enqpijchange failed, declaring a cluster crash
• Failure during WIJ processing - Declaring a crash
• Failure during PIJ processing - Declaring a crash
• Error reading block – recovery read error
• Error writing block – recovery write error
• WIJ expansion failure: System Frozen - The system has been frozen because WIJ expansion has failed for too long. If space is created for the WIJ, the system will resume otherwise you need to shut it down with 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 slave write daemon
• ENQDMN exiting due to reason #
• Becoming primary mirror server
4

Using System Monitor

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. As provided, System Monitor incorporates the following InterSystems IRIS instance 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. System Monitor and Application Monitor can optionally be run in other namespaces under namespace-specific configurations and settings. You can define and configure your own components to extend the capabilities of System Monitor in each namespace as your needs require.

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

4.1 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.

The remainder of this section discusses the following topics:

- The System Monitor Process
- Tracking System Monitor Notifications
- System Monitor Status and Resource Metrics
4.1.1 The System Monitor Process

In each namespace in which it is configured to run, System Monitor gathers and delivers system metric information in three stages using three types of classes (or System Monitor components). 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 the Default System Monitor Components section in this chapter.

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

### 4.1.2 Tracking System Monitor Notifications

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

System Monitor alerts, 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 in the “Monitoring InterSystems IRIS Using the Management Portal” chapter). Bear in mind that when a sequence of System Monitor alerts is generated for a given sensor within a short period of time, only the first is written to the 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.

### 4.1.3 System Monitor Status and Resource Metrics

The following table lists the system status and resource usage metrics sampled by System Monitor, and the notification thresholds and rules for each that result in warnings (severity 1), alerts (severity 2), and “status OK” (severity 0) notifications.
### Table 4–1: System Monitor Status and Resource Notifications

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Notification Rules</th>
</tr>
</thead>
</table>

Using System Monitor
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Notification Rules</th>
</tr>
</thead>
</table>
| 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            | 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) |
### 4.1.4 System Monitor Health State

Based on notifications posted to the messages log (see Monitoring Log Files in the “Monitoring InterSystems IRIS Using the Management Portal” chapter of this guide), 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 4–2: System Monitor Health State**

<table>
<thead>
<tr>
<th>State</th>
<th>Set at startup when ...</th>
<th>Set following startup when ...</th>
<th>Cleared to ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GREEN (0)</td>
<td>no system alerts are posted during startup</td>
<td>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</td>
<td>n/a</td>
</tr>
<tr>
<td>YELLOW (1)</td>
<td>up to four system alerts are posted during startup</td>
<td>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</td>
<td>GREEN when 30 minutes have elapsed since the last system alert or System Monitor alert or warning was posted</td>
</tr>
<tr>
<td>RED (2)</td>
<td>five or more system alerts are posted during startup</td>
<td>• 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</td>
<td>YELLOW when 30 minutes have elapsed since the last system alert or System Monitor alert or warning was posted</td>
</tr>
</tbody>
</table>
Note: A fourth state, **HUNG**, can occur when global updates are blocked. Specifically, the following events change the state to **HUNG**:

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

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

You can view the System Monitor health state using:

- The **View System Health** option on the **View System Data** menu of `^%SYSTEMMGR` (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**.

### 4.1.5 System Monitor Defaults

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

#### 4.1.5.1 Default System Monitor Components

Five classes are provided with 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 the chapter “Monitoring InterSystems IRIS Using the Management Portal” in this guide for information on these log files.)

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

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

### 4.1.5.2 Default System Monitor Namespace

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

Health Monitor runs only in the %SYS namespace.

### 4.1.5.3 Default System Monitor Settings

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

By default, the System Monitor:

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

These settings can be changed using ^%SYSMONMGR.

### 4.1.6 Using the ^%SYSMONMGR Utility

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

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

%SYS>do ^%SYSMONMGR

The main menu appears.

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

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

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

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

4.1.6.1 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.
Using System Monitor

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 Files in the “Monitoring InterSystems IRIS Using the Management Portal” chapter for information about the messages log.

4.1.6.2 Set System Monitor Options

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

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

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

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

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

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

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

4.1.6.3 Configure System Monitor Components

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

Configure System Monitor Classes

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

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

Enter 1 to display the options for configuring classes:

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

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

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

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

Note: Configuring or deleting a class affects only the namespace in which you started ^%SYSMONMGR.
Configure System Monitor Namespaces

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

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

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

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

Enter 2 to display the options for configuring namespaces:

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

Enter 1 to list the currently configured startup namespaces.

Enter 2 to add a startup namespace.

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

4.1.6.4 View System Monitor State

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

<table>
<thead>
<tr>
<th>Component</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Monitor</td>
<td>OK</td>
</tr>
<tr>
<td>%SYS.Monitor.AppMonSensor</td>
<td>None</td>
</tr>
<tr>
<td>SYS.Monitor.SystemSensors</td>
<td>OK</td>
</tr>
<tr>
<td>SYS.Monitor.Health.Control</td>
<td>Running: Period is Thursday 09:00 - 11:30</td>
</tr>
<tr>
<td>SYS.Monitor.SystemSubscriber</td>
<td>OK</td>
</tr>
<tr>
<td>SYS.Monitor.SystemNotifier</td>
<td>OK</td>
</tr>
</tbody>
</table>

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

4.1.6.5 Manage Application Monitor

See Using ^%SYSMONMGR to Manage Application Monitor.

4.1.6.6 Manage Health Monitor

See Using ^%SYSMONMGR to Manage Health Monitor.

4.1.6.7 View System Data

Enter 7 at the main menu (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.

4.1.7 Defining System Monitor Components

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

4.1.7.1 Sensor Classes

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

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

4.1.7.2 Subscriber Classes

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

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

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

4.1.7.3 Notifier Classes

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

4.2 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.

This section covers the following topics:

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

### 4.2.1 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

#### 4.2.1.1 Health Monitor Process Description

By default, Health Monitor does not start automatically when the instance starts; for this to happen, you must enable Health Monitor within 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, later in this chapter.

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.

  The Sensors and Sensor Objects section below contains a list of all the sensor objects Health Monitor uses, a brief description of each, and their default values.

- 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. The Periods section below 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 the Charts section below.

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 the Notification Rules section below), 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.

### 4.2.1.2 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 the Configure Health Monitor Classes section below). 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 4–3: Health Monitor Sensor Objects

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUsage</td>
<td></td>
<td>System CPU usage (percent).</td>
<td>50</td>
<td>85</td>
<td>—</td>
<td>75</td>
<td>—</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>CSPSessions</td>
<td>IP_address:port</td>
<td>Number of active web sessions on the listed Web Gateway server.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>CSPActivity</td>
<td>IP_address:port</td>
<td>Requests per minute to the listed Web Gateway server.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>CSPActualConnections</td>
<td>IP_address:port</td>
<td>Number of connections created on the listed Web Gateway server.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>CSPInUseConnections</td>
<td>IP_address:port</td>
<td>Number of currently active connections to the listed Web Gateway server.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>CSPPrivateConnections</td>
<td>IP_address:port</td>
<td>Number of private connections to the listed Web Gateway server.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>CSPUrlLatency</td>
<td>IP_address:port</td>
<td>Time (milliseconds) required to obtain a response from CSP sensor objects.</td>
<td>1000</td>
<td>5000</td>
<td>—</td>
<td>3000</td>
<td>—</td>
</tr>
<tr>
<td>CSPGatewayLatency</td>
<td>IP_address:port</td>
<td>Time (milliseconds) required to obtain a response from the listed Web Gateway server when fetching the metrics represented by the CSP sensor objects.</td>
<td>1000</td>
<td>2000</td>
<td>—</td>
<td>1000</td>
<td>—</td>
</tr>
<tr>
<td>DBLatency</td>
<td>database directory</td>
<td>Milliseconds to complete a random read from the listed mounted database.</td>
<td>1000</td>
<td>3000</td>
<td>—</td>
<td>1000</td>
<td>—</td>
</tr>
<tr>
<td>DBReads</td>
<td>database directory</td>
<td>Reads per minute from the listed mounted database.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>DBWrites</td>
<td>database directory</td>
<td>Writes per minute to the listed mounted database.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>ECPAppServerKBPerMinute</td>
<td></td>
<td>KB per minute sent to the ECP data server.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>ECPConnections</td>
<td></td>
<td>Number of active ECP connections.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>ECPDataServerKBPerMinute</td>
<td></td>
<td>KB per minute received as ECP data server.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>ECPLatency</td>
<td></td>
<td>Network latency (milliseconds) between the ECP data server and this ECP application server.</td>
<td>1000</td>
<td>3000</td>
<td>—</td>
<td>3000</td>
<td>—</td>
</tr>
<tr>
<td>ECPTransOpenCount</td>
<td></td>
<td>Number of open ECP transactions</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ECPTransOpenSecsMax</td>
<td></td>
<td>Duration (seconds) of longest currently open ECP transaction</td>
<td>60</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>GlobalRefsPerMin</td>
<td></td>
<td>Global references per minute.</td>
<td>1024</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>GlobalSetKillPerMin</td>
<td></td>
<td>Global sets/kills per minute.</td>
<td>1024</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>JournalEntriesPerMin</td>
<td></td>
<td>Number of journal entries written per minute.</td>
<td>1024</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>JournalGrowthRate</td>
<td></td>
<td>Number of KB per minute written to journal files.</td>
<td>1024</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>LicensePercentUsed</td>
<td></td>
<td>Percentage of authorized license units currently in use.</td>
<td>50</td>
<td></td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LicenseUsedRate</td>
<td></td>
<td>License acquisitions per minute.</td>
<td>20</td>
<td></td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LockTablePercentFull</td>
<td></td>
<td>Percentage of the lock table in use.</td>
<td>50</td>
<td>99</td>
<td></td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>LogicalBlockRequestsPerMin</td>
<td></td>
<td>Number of logical block requests per minute.</td>
<td>1024</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>MirrorDatabaseLatencyBytes</td>
<td>mirror_name</td>
<td>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).</td>
<td>$2 \times 10^7$</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>MirrorDatabaseLatencyFiles</td>
<td>mirror_name</td>
<td>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).</td>
<td>3</td>
<td></td>
<td>2</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>MirrorDatabaseLatencyTime</td>
<td>mirror_name</td>
<td>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).</td>
<td>1000</td>
<td>4000</td>
<td>—</td>
<td>3000</td>
<td>—</td>
</tr>
<tr>
<td>MirrorJournalLatencyBytes</td>
<td>mirror_name</td>
<td>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).</td>
<td>$2^{10^7}$</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>MirrorJournalLatencyFiles</td>
<td>mirror_name</td>
<td>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).</td>
<td>3</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>MirrorJournalLatencyTime</td>
<td>mirror_name</td>
<td>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).</td>
<td>1000</td>
<td>4000</td>
<td>—</td>
<td>3000</td>
<td>—</td>
</tr>
<tr>
<td>PhysicalBlockReadsPerMin</td>
<td></td>
<td>Number of physical block reads per minute.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>PhysicalBlockWritesPerMin</td>
<td></td>
<td>Number of physical block writes per minute.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>ProcessCount</td>
<td></td>
<td>Number of active processes for the InterSystems IRIS instance.</td>
<td>100</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>RoutineCommandsPerMin</td>
<td></td>
<td>Number of routine commands per minute.</td>
<td>1024</td>
<td>—</td>
<td>2</td>
<td>—</td>
<td>1.6</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>RoutineLoadsPerMin</td>
<td>Number of routine loads per minute.</td>
<td>1024 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoutineRefsPerMin</td>
<td>Number of routine references per minute.</td>
<td>1024 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMHPercentFull</td>
<td>Percentage of the shared memory heap (generic memory heap) in use.</td>
<td>50 98 — 85 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TransOpenCount</td>
<td>Number of open local transactions (local and remote).</td>
<td>100 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TransOpenSecondsMax</td>
<td>Duration (seconds) of longest currently open local transaction.</td>
<td>60 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDBuffers</td>
<td>Average number of database buffers updated per write daemon cycle.</td>
<td>1024 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDCycleTime</td>
<td>Average number of seconds required to complete a write daemon cycle.</td>
<td>60 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDWIJTime</td>
<td>Average number of seconds spent updating the write image journal (WIJ) per cycle.</td>
<td>60 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDDWriteSize</td>
<td>Average number of bytes written per write daemon cycle.</td>
<td>1024 — 2 — 1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 the “Mirroring” chapter of the *High Availability Guide*), the following special conditions apply to Health Monitor:

- No sensors are sampled while the mirror is restarting (for example, just after the backup failover member has taken over as primary) or if the member’s status in the mirror is indeterminate.
- If a sensor is in analysis mode for a period and the member’s status in the mirror changes during the period, no chart is created and the sensor remains in analysis mode.
- Only the `MirrorDatabaseLatency` and `MirrorJournalLatency` sensors are sampled on the backup failover mirror member.
- All sensors except the `MirrorDatabaseLatency` and `MirrorJournalLatency` sensors are sampled on the primary failover mirror member.
4.2.1.3 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 4-4: Default Health Monitor Periods

<table>
<thead>
<tr>
<th>Time Interval 1</th>
<th>Time Interval 2</th>
<th>Time Interval 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:15 a.m. – 02:45 a.m.</td>
<td>03:00 a.m. – 06:00 a.m.</td>
<td>06:15 a.m. – 08:45 a.m.</td>
</tr>
<tr>
<td>09:00 a.m. – 11:30 a.m.</td>
<td>11:45 a.m. – 01:15 p.m.</td>
<td>01:30 p.m. – 04:00 p.m.</td>
</tr>
<tr>
<td>04:15 p.m. – 06:00 p.m.</td>
<td>06:15 p.m. – 08:45 p.m.</td>
<td>09:00 p.m. – 11:59 p.m.</td>
</tr>
</tbody>
</table>

You can list, add and delete periods using the Configure Periods option in the ^%SYSMONMGR utility (see the Configure Health Monitor Classes section below). 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.

4.2.1.4 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.

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.
## 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 the **Analysis Mode** section above. 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.

### 4.2.1.5 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

<table>
<thead>
<tr>
<th>Chart Type</th>
<th>Sensor Maximum Value</th>
<th>Sensor Warning Value</th>
<th>Active When</th>
</tr>
</thead>
<tbody>
<tr>
<td>base, maximum value, warning value</td>
<td>none</td>
<td>sensor object maximum value</td>
<td>System Monitor running</td>
</tr>
<tr>
<td>base, maximum multiplier, warning multiplier</td>
<td>generated</td>
<td>sensor object maximum multiplier times greater of:</td>
<td>System Monitor running, Health Monitor enabled</td>
</tr>
<tr>
<td>base only</td>
<td>generated</td>
<td>greater of:</td>
<td></td>
</tr>
<tr>
<td>(n/a if user-created chart exists)</td>
<td>usercreated</td>
<td>chart alert value</td>
<td>chart warning value</td>
</tr>
</tbody>
</table>

#### 4.2.1.6 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 multiplier 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 \( \times (\text{chart mean} + (3 \times \text{chart sigma})) \)
  
  \[ 2 \times (2145 + (3 \times 141)) = 5136 \]

- maximum multiplier \( \times (\text{chart maximum value} + \text{chart sigma}) \)
  
  \[ 2 \times (2327 + 141) = 4936 \]

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 \( \times (\text{the chart mean} + \text{three times the sigma}) \)
  
  \[ 1 \times (2145 + (3 \times 141)) = 2568 \]

- maximum multiplier \( \times (\text{the highest value in the chart} + \text{one sigma}) \)
  
  \[ 1 \times (2327 + 141) = 2468 \]
4.2.2 Using ^%SYSMONMGR to Manage Health Monitor

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

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

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

Option?

**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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1) Enable/Disable Health Monitor | • Enable Health Monitor (if it is disabled, as by default), so that it starts when System Monitor starts. Health Monitor does not begin collecting sensor reading until after the configured startup wait time is complete.  
  • Disable Health Monitor (if it is enabled), so that it does not start when System Monitor starts. |
| 2) View Alert Records           | • View alert records for one or all sensors objects over a specified date range.                                                             |
| 3) Configure Health Monitor Classes | • List notification rules.  
  • List and delete existing periods and add new ones.  
  • List, examine, edit, create and recalibrate charts.  
  • List sensor objects and edit their settings.  
  • Reset Health Monitor elements to their defaults. |
| 4) Set Health Monitor Options   | • Set startup wait time.  
  • Specify when alert records should be purged. |
Note: When the utility asks you to specify a single element such as a sensor, rule, period or chart, you can enter ? (question mark) at the prompt for a numbered list, then enter the number of the element you want.

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

### 4.2.2.1 View Alerts Records

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

### 4.2.2.2 Configure Health Monitor Classes

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

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

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Set Startup Wait Time</td>
<td>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.</td>
</tr>
<tr>
<td>2) Set Alert Purge Time</td>
<td>Specify when an alert record should be purged (deleted); the default is five days after the alert is generated.</td>
</tr>
</tbody>
</table>

### 4.3 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.

This section covers the following topics:

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

#### 4.3.1 Application Monitor Overview

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 the “History Monitor” chapter of this guide for more information about these classes and the historical database.

### 4.3.2 Using ^%SYSMONMGR to Manage Application Monitor

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

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

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

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

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

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

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

#### 4.3.2.1 Manage Application Monitor

The options in the main menu let you manage Application Monitor as described in the following table:
Option | Description
--- | ---
1) Set Sample Interval | Sets the interval at which metrics are sampled; the default is 30 seconds. This setting can be overridden for an individual class by setting a class-specific interval (using the 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.

4.3.2.2 Manage Monitor Classes

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

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

Option?

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

Option | Description
--- | ---
1) Activate / Deactivate Monitor Class | Application Monitor samples active classes only. This option lets you activate an 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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3) Register Monitor System Classes</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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. |

**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:

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Enable Debug</td>
<td>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.</td>
</tr>
<tr>
<td>2) Disable Debug</td>
<td>Lets you disable debugging.</td>
</tr>
</tbody>
</table>
### 4.3.2.3 Change Default Notification Method

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

Option? 3

Notify Action (0=none,1=email,2=method)? 0 ->

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

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not take action when an alert is triggered.</td>
</tr>
<tr>
<td>1</td>
<td>Send an email message to the configured recipients when an alert is triggered. For information about configuring email, see Manage Email Options.</td>
</tr>
<tr>
<td>2</td>
<td>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 <code>packagename.classname.method</code>. This method must exist in the local namespace.</td>
</tr>
</tbody>
</table>

### 4.3.2.4 Manage Email Options

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

Option? 4

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

Option?

The options in this submenu let you manage the email notifications for the Application Monitor as described in the following table:
Option | Description
--- | ---
1) Enable / Disable Email | Enabling email makes it possible for Application Monitor to send email notifications when alerts are triggered, if configured. Disabling email prevents Application Monitor from sending email notifications when an alert is triggered.

**Note:** It is not necessary to reconfigure email options when you disable and then reenable email.

2) Set Sender | This option is required. Enter text identifying the sender of the email. Depending on the specified outgoing mail server, this may or may not have to be a valid email account.

3) Set Server | This option is required. Enter the name of the server that handles outgoing email for your site. If you are not sure, your IT staff should be able to provide this information.

4) Manage Recipients | This option displays a submenu that lets you list, add, or remove valid email addresses of recipients:

- 1) List Recipients
- 2) Add Recipient
- 3) Remove Recipient
- 4) Exit

When adding or removing recipients, email addresses must be entered individually, one per line. Addresses of invalid format are rejected.

5) Set Authorization | Lets you specify the authorization username and password if required by your email server. Consult your IT staff to obtain this information. If you do not provide entries, the authorization username and password are set to NULL.

6) Test Email | Sends a test message to the specified recipients using the specified email server. If the attempt fails, the resulting error message may help you fix the problem.

### 4.3.2.5 Manage Alerts

An alert specifies:

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

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

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

The definition of a condition based on properties is called an *evaluation expression*; after specifying the properties of the sample class you want to use, you specify the evaluation expression. Properties are indicated in the expression by 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 \( \frac{\%1}{\%2} > .9 \), but if you enter the properties in the reverse order, the expression would be \( \frac{\%2}{\%1} > .9 \).

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:

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Create Alert</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 2) Edit Alert      | Lets you modify an existing alert. Enter the name of the alert to edit, or enter ? for a list of existing alerts and then enter a number or name.  
      
      **Note:** You must respond to all prompts including those that you do not want to modify; that is, you must re-enter information for fields that you do not want to modify as well as the revised information for the fields you are modifying. For a description of the prompts and responses, see the Responses to Alert Prompts. |
| 3) List Alerts     | Displays the definitions of all alerts in the local namespace; for example:  
      Alert: LockTable90 USER  
      Action: email  
      Class: %Monitor.System.LockTable  
      Property: UsedSpace, TotalSpace  
      Expression: \( \%1/\%2 > .9 \)  
      Notify Once: True  
      Enabled: Yes |
| 4) Delete Alert    | Lets you delete an existing alert. Enter the name of the alert to edit, or enter ? for a list of existing alerts and then enter a number or name.  
      **Note:** Each alert must be entered individually; that is, you cannot specify a series or range of alerts to delete. |
| 5) Enable/Disable Alert | Enabling an alert activates it. Disabling an alert deactivates it.  
      **Note:** It is not necessary to reconfigure alert options when you disable and then reenable an alert. |
The following table describes the valid responses to Alert prompts:

**Table 4–5: Responses to Alert Prompts**

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert Name?</td>
<td>Enter an alphanumeric name. To display a numbered list of alert names already defined in the local namespace, enter ? at the Alert Name? prompt.</td>
</tr>
<tr>
<td>Application?</td>
<td>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.</td>
</tr>
<tr>
<td>Action (0=default, 1=email, 2=method)?</td>
<td>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.</td>
</tr>
<tr>
<td>Raise this alert during sampling? or Define a trigger for this alert?</td>
<td>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.</td>
</tr>
<tr>
<td>Class?</td>
<td>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. <strong>Note:</strong> You can create an alert for an inactive class. An alert is not removed when the class it is configured for is removed.</td>
</tr>
</tbody>
</table>
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.

### 4.3.3 Application Monitor Metrics

The system monitor classes included with Application Monitor call the following sample classes:

**Table 4–6: Application Monitor System Classes**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Monitor.System.Sample.AuditCount</td>
<td>Audit metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.AuditEvents</td>
<td></td>
</tr>
<tr>
<td>%Monitor.System.Sample.Clients</td>
<td>Client metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.CSPGateway</td>
<td>Web Gateway metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.Diskspace</td>
<td>Disk space metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.Freespace</td>
<td>Free space metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.Globals</td>
<td>Global metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.HistoryPerf</td>
<td>History database metrics (see the “History Monitor” chapter of this guide)</td>
</tr>
<tr>
<td>%Monitor.System.Sample.HistorySys</td>
<td></td>
</tr>
<tr>
<td>%Monitor.System.Sample.HistoryUser</td>
<td></td>
</tr>
<tr>
<td>%Monitor.System.Sample.License</td>
<td>License metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.LockTable</td>
<td>Lock table metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.Routines</td>
<td>Routine metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.Servers</td>
<td>Server metrics</td>
</tr>
<tr>
<td>%Monitor.System.Sample.SystemMetrics</td>
<td>System activity metrics</td>
</tr>
</tbody>
</table>

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

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

4.3.3.1 Generating Metrics

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

The %Monitor.SampleAgent.%New() 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:

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

4.3.3.2 Viewing Metrics Data

All metrics classes are CSP-enabled; the CSP code is generated automatically when the sample class is generated. Therefore, the simplest way to view metrics is using a web browser; for example, based on the example in Generating Metrics and assuming a superserver port of 52773, the CSP URL is http://localhost:52773/csp/user/MyMetrics.Sample.Freespace.cls, which displays output 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:

```plaintext
%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, for the %Monitor.System.Freespace class, use

4.3.4 Writing User-Defined Application Monitor Classes

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

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

1. 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 in the InterSystems Class Reference for full details on this class, as well as the code you must write.

2. 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 or Atelier 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 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 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:

```
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.

```
/// 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.

```
/// 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.

```
/// 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:

```clike
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:

```plaintext
/// 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:

```plaintext
/// 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.
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).

5.1 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:

```
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:

<table>
<thead>
<tr>
<th>Response</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Displays the report again with updated cumulative statistics since the last initialization.</td>
</tr>
<tr>
<td>q</td>
<td>Quits the ^GLOSTAT routine.</td>
</tr>
<tr>
<td># (a positive integer indicating number of seconds)</td>
<td>Initializes statistics, counts statistics for the indicated number of seconds, and reports statistics as an average per second (Example B).</td>
</tr>
</tbody>
</table>
5.2 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>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global references (all)</td>
<td>Logical count of accesses to globals, including Sets, Kills, $Data, $Order, $Increment, $Query, and global references in expressions.</td>
</tr>
<tr>
<td>Global update references</td>
<td>Logical count of global references that are Sets, Kills, or $Increments.</td>
</tr>
<tr>
<td>Private global references</td>
<td>The count of all process private global accesses.</td>
</tr>
<tr>
<td>Private update references</td>
<td>The count of process private global references that are SETs or KILLs, etc.</td>
</tr>
<tr>
<td>Routine calls</td>
<td>Number of calls to a routine.</td>
</tr>
<tr>
<td>Routine buffer loads and saves</td>
<td>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).)</td>
</tr>
<tr>
<td>Routine commands</td>
<td>Number of routine commands executed since system startup.</td>
</tr>
<tr>
<td>Routine not cached</td>
<td>Number of routines not cached in memory. This information help you determine whether or not the routine buffer cache is adequately sized.</td>
</tr>
<tr>
<td>Logical block requests</td>
<td>Number of database blocks read by the globals database code. (In a well-tuned environment, many of these reads are satisfied without disk access.)</td>
</tr>
<tr>
<td>Block reads</td>
<td>Number of physical database blocks read from disk for both global and routine references.</td>
</tr>
<tr>
<td>Block writes</td>
<td>Number of physical database blocks written to disk for both global and routine references.</td>
</tr>
<tr>
<td>WIJ writes</td>
<td>Number of writes to the write image journal file.</td>
</tr>
<tr>
<td>Cache Efficiency</td>
<td>Number of all global references divided by the number of physical block reads and writes. Not a percentage.</td>
</tr>
<tr>
<td>Journal Entries</td>
<td>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.</td>
</tr>
<tr>
<td>Journal Block Writes</td>
<td>Number of 64-KB journal blocks written to the journal file.</td>
</tr>
</tbody>
</table>
5.3 Examples of ^GLOSTAT Output

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

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

### 5.3.1 Example A

The following is sample output of the initial running of the ^GLOSTAT routine. The 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:
Private update references:                   76,739
Routine calls:                               650,085
Routine buffer loads & saves:
Routine commands:                            17,747,411
Routine not cached:                          710
Logical block requests:                      289,166
Block reads:                                 2,179
Block writes:                                680
WIJ writes:                                  903
Cache Efficiency:                            186
Journal Entries:                             1,356
Journal Block Writes:                        6

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

### 5.3.2 Example B

The following example shows ^GLOSTAT statistics per second for a 30-second timed interval. The 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:
Private update references:                 0.9
Routine calls:                             8.8
Routine buffer loads & saves:
Routine commands:                          222.2
Routine not cached:
Logical block requests:
Block reads:                                0
Block writes:
WIJ writes:
Cache Efficiency:                          no i/o
Journal Entries:
Journal Block Writes:

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

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

```sql
%SYS> do ^GLOSTAT

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

Continue (c), Timed Stats (# sec > 0), Quit (q)?
```
Monitoring System Performance Using ^PERFMON

^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 in the “Using System Monitor” chapter of this guide and the “Examining Routine Performance Using ^%SYS.MONLBL” chapter of this guide.

6.1 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
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.

### 6.1.1 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>:

### 6.1.2 Start

Turns on collection of the statistics.

*Format:*
status = $$Start^\text{PERFMON}(\text{process, routine, global, database, network})$$

Parameters:

- **process** — number of process slots to reserve (default = $$\text{pcount}$$ (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:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Somebody else is using Monitor</td>
</tr>
<tr>
<td>-2</td>
<td>Monitor is already running</td>
</tr>
<tr>
<td>-3</td>
<td>Memory allocation failed</td>
</tr>
<tr>
<td>-4</td>
<td>Could not enable statistics collection</td>
</tr>
</tbody>
</table>

### 6.1.3 Stop

Stops collection of statistics.

**Format:**

```
status = $$\text{Stop^PERFMON}()$$
```

**Status Codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Somebody else is using Monitor</td>
</tr>
<tr>
<td>-2</td>
<td>Monitor is not running</td>
</tr>
</tbody>
</table>

### 6.1.4 Pause

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

**Format:**

```
status = $$\text{Pause^PERFMON}()$$
```

**Status Codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
</tbody>
</table>
### 6.1.5 Resume

Resumes collection of statistics that you previously paused.

**Format:**

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

**Status Codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Somebody else is using Monitor</td>
</tr>
<tr>
<td>-2</td>
<td>Monitor is not running</td>
</tr>
<tr>
<td>-3</td>
<td>Monitor is already paused</td>
</tr>
</tbody>
</table>

### 6.1.6 Sample Counters

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

**Format:**

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

**Parameters:**

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

**Status Codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-2</td>
<td>Monitor is not running</td>
</tr>
<tr>
<td>-8</td>
<td>Sample job already running</td>
</tr>
</tbody>
</table>

### 6.1.7 Clear

Clears all metric counters.
Format:

\texttt{status = \$\texttt{\$Clear}\textasciitilde\texttt{PERFMON}()}

\textbf{Status Codes:}

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Somebody else is using Monitor</td>
</tr>
<tr>
<td>-2</td>
<td>Monitor is not running</td>
</tr>
</tbody>
</table>

\section*{6.1.8 Report}

The report function gathers and outputs a report of metrics.

\textit{Format:}

\texttt{status = \$\texttt{\$Report}\textasciitilde\texttt{PERFMON}(\texttt{report,sort,format,output,\{list\},\{data\})}}

\textit{Parameters:}

\begin{itemize}
  \item \texttt{report} — type of report to output; valid values are:
    \begin{itemize}
      \item \texttt{G} – for global activity
      \item \texttt{R} – for routine activity
      \item \texttt{N} – for network activity
      \item \texttt{C} – for a custom report where you select the metrics to report
    \end{itemize}
  \item \texttt{sort} — grouping and sort order of report; valid values are:
    \begin{itemize}
      \item \texttt{P} – to organize the report by Process
      \item \texttt{R} – to organize the report by Routine
      \item \texttt{G} – to organize the report by Global
      \item \texttt{D} – to organize the report by Database
      \item \texttt{I} – to organize the report by Incoming node
      \item \texttt{O} – to organize the report by Outgoing node
    \end{itemize}
  \item \texttt{format} — output format; valid values are:
    \begin{itemize}
      \item \texttt{P} – for a printable/viewable report (\texttt{.txt} file, no pagination)
      \item \texttt{D} – for comma delimited data (\texttt{.csv} file) which can be read into a spreadsheet
      \item \texttt{X} – for Microsoft Excel XML markup suitable for import into Excel (\texttt{.xml} file)
      \item \texttt{H} – for an HTML page (\texttt{.html} file)
    \end{itemize}
  \item \texttt{output} — enter a file name, \textbf{Return} to accept the default file name displayed, or 0 (zero) for output to the screen
  \item \texttt{list} — (for Custom report only) comma-separated list of metric numbers which specify what columns to include in the report; the following table lists the metrics available.
\end{itemize}
## Table 6-1: Metrics for Custom ^PERFMON Report

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

<table>
<thead>
<tr>
<th>Metric Number</th>
<th>Column Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>NCacheHit</td>
<td>network cache hits</td>
</tr>
<tr>
<td>33</td>
<td>NCacheMiss</td>
<td>network cache misses</td>
</tr>
<tr>
<td>34</td>
<td>NetLock</td>
<td>network locks</td>
</tr>
<tr>
<td>35</td>
<td>RtnLine</td>
<td>M commands</td>
</tr>
<tr>
<td>36</td>
<td>RtnLoad</td>
<td>routine loads</td>
</tr>
<tr>
<td>37</td>
<td>RtnFetch</td>
<td>routine fetches</td>
</tr>
<tr>
<td>38</td>
<td>LockCom</td>
<td>lock commands</td>
</tr>
<tr>
<td>39</td>
<td>LockSucc</td>
<td>successful lock commands</td>
</tr>
<tr>
<td>40</td>
<td>LockFail</td>
<td>failed lock commands</td>
</tr>
<tr>
<td>41</td>
<td>TermRead</td>
<td>terminal reads</td>
</tr>
<tr>
<td>42</td>
<td>TermWrite</td>
<td>terminal writes</td>
</tr>
<tr>
<td>43</td>
<td>TermChRd</td>
<td>terminal read chars</td>
</tr>
<tr>
<td>44</td>
<td>TermChWrt</td>
<td>terminal write chars</td>
</tr>
<tr>
<td>45</td>
<td>SeqRead</td>
<td>sequential reads</td>
</tr>
<tr>
<td>46</td>
<td>SeqWrt</td>
<td>sequential writes</td>
</tr>
<tr>
<td>47</td>
<td>IJCMsgRd</td>
<td>local IJC messages read</td>
</tr>
<tr>
<td>48</td>
<td>IJCMsgWt</td>
<td>local IJC messages written</td>
</tr>
<tr>
<td>49</td>
<td>IJCNNetMsg</td>
<td>network IJC messages written</td>
</tr>
<tr>
<td>50</td>
<td>Retransmit</td>
<td>network retransmits</td>
</tr>
<tr>
<td>51</td>
<td>BuffSent</td>
<td>network buffers sent</td>
</tr>
</tbody>
</table>

**Note:** This counter is no longer available in the current network protocol. However, network cache hits can be calculated using the formula NetGloRef - NetGloSet - NetGloKill - NCacheMiss.

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:**
<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Monitor is not running</td>
</tr>
<tr>
<td>-2</td>
<td>Missing input parameter</td>
</tr>
<tr>
<td>-3</td>
<td>Invalid report category</td>
</tr>
<tr>
<td>-4</td>
<td>Invalid report organization</td>
</tr>
<tr>
<td>-5</td>
<td>Invalid report format</td>
</tr>
<tr>
<td>-6</td>
<td>Invalid list for custom report</td>
</tr>
<tr>
<td>-7</td>
<td>Invalid data format</td>
</tr>
</tbody>
</table>

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

### 6.1.9 Collect

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

**Format:**

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

**Parameters:**

- `time` — number of seconds for data collection (default 30)
- `format` — output format; valid values are:
  - `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

**Status Codes:**

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Successful</td>
</tr>
<tr>
<td>-1</td>
<td>Somebody else is using Monitor</td>
</tr>
<tr>
<td>-3</td>
<td>Monitor is already running</td>
</tr>
</tbody>
</table>

### 6.2 Report Examples

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

7.1 Using ^PROFILE

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

%SYS>do ^PROFILE

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

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

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

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

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Flag the specified line(s) for detailed profile-level data collection.</td>
</tr>
<tr>
<td>Note:</td>
<td>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.</td>
</tr>
<tr>
<td>B</td>
<td>Display the previous page of the list.</td>
</tr>
<tr>
<td>E</td>
<td>Export the displayed collection of metrics.</td>
</tr>
<tr>
<td>N</td>
<td>Display the next page of the list.</td>
</tr>
<tr>
<td>O</td>
<td>Re-sort the page based on different metrics (the selected metric is displayed in the first column).</td>
</tr>
<tr>
<td>Q</td>
<td>Exit from the ^PROFILE utility.</td>
</tr>
<tr>
<td>Note:</td>
<td>If you flagged routines that you want to analyze, this option lets you choose between collecting subroutine- and line-level metrics or exiting.</td>
</tr>
<tr>
<td>R</td>
<td>Refresh the list with the most recent metrics.</td>
</tr>
<tr>
<td>X</td>
<td>Clear all flags of selected routines (including those selected on other pages) and refresh the collection of metrics.</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>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.</td>
</tr>
<tr>
<td>B</td>
<td>Display the previous page of the list.</td>
</tr>
<tr>
<td>L</td>
<td>Switch to the line level display of the subroutine.</td>
</tr>
<tr>
<td>N</td>
<td>Display the next page of the list.</td>
</tr>
<tr>
<td>Q</td>
<td>Exit the list, return to the previous level.</td>
</tr>
<tr>
<td>R</td>
<td>Refresh the list with the most recent metrics.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>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.</td>
</tr>
<tr>
<td>B</td>
<td>Display the previous page of the list.</td>
</tr>
<tr>
<td>C</td>
<td>Switch code display between source code and intermediate (INT/MVI) code.</td>
</tr>
<tr>
<td>M</td>
<td>Change the page margin and length.</td>
</tr>
<tr>
<td>N</td>
<td>Display the next page of the list.</td>
</tr>
<tr>
<td>O</td>
<td>Re-sort the page based on different metrics.</td>
</tr>
<tr>
<td>Q</td>
<td>Exit the list, returning to the previous level.</td>
</tr>
</tbody>
</table>
7.2 ^PROFILE Example

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

1. Enter the following command:
   
   ```
   do ^PROFILE
   ```

2. The following message appears.

   *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%       36.77%      42.55%
       %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%       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%       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 (IRISSYS)
   10.   0.28%       0.93%       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:

   ```
   Monitoring Routine Performance Using ^PROFILE
   ```
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.

<table>
<thead>
<tr>
<th>RtnLine</th>
<th>Routine Name (Database)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>96.72%  %Library.Persistent.1.INT (IRISLIB)</td>
</tr>
<tr>
<td>2.</td>
<td>3.28%   Config.CPF.1.INT (IRISSYS)</td>
</tr>
</tbody>
</table>

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.
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.

8.1 Invoking the Line-by-line Monitoring Routine

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

%SYS>Do ^%SYS.MONLBL

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

The line-by-line monitor also allocates shared memory to track these statistics for each line of each routine selected. This is taken from the general shared memory already allocated 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 the Estimate Memory Requirements section for details.
- Press the Enter key to exit the routine.
8.1.1 Start Monitoring

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

1. **Routine Names** – Enter a list of routine names to monitor. You can only select routines accessible from your current namespace. Do not use the leading ^ when entering the routine name; the names are case-sensitive. You may use asterisk (*) wild cards to select multiple routines. Press Enter twice after entering the last routine name to end the list.

2. **Select Metrics to monitor** – Enter the number of your choice of what type of metrics. The default is 1 for minimal metrics.

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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric#: 34 - RtnLine</td>
<td>Number of times a routine line is executed</td>
</tr>
<tr>
<td>Metric#: 51 - Time</td>
<td>Clock time spent in executing that line</td>
</tr>
<tr>
<td>Metric#: 52 - TotalTime</td>
<td>Total clock time for that line including time spent in subroutines called by that line</td>
</tr>
</tbody>
</table>

   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 the Line-by-line Monitor Programming Interface section for a method of retrieving the list.

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

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

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

Enter the number of your choice: <1> 3

^%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

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

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

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

Monitor started.
Press RETURN to continue ...

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

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

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

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: 5

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

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

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

Press RETURN to continue ...

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

8.2 Line-by-line Monitoring Options

If you invoke ^%SYS.MONLBL while the monitor is running you have the following menu options:
Line-by-Line Monitor

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

Enter the number of your choice:

The first three options are fairly self-explanatory:

• Stop Monitor — Stops all ^%SYS.MONLBL monitoring; deallocates the counter memory and deletes collected data.

• Pause Monitor — Pauses the collection and maintains any collected data. This may be useful when viewing collected data to ensure that counts are not changing as the report is displayed. This option only appears if the monitor is running.

• Resume Monitor — Resumes collection after a pause. This option only appears if you paused the monitor.

• Clear Counters — Clears any collected data, but continues monitoring and collecting new data.

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

8.2.1 Report Line-by-line Statistics

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

You have four types of reports to choose from:

• Detail — Generates a report of the selected metrics for each line in the selected routines. The report accumulates and displays totals for each of the performance columns.

• Summary — Generates a report of summary information for each selected routine including coverage and time. The report orders the routines by coverage percentage.

• Delimited (CSV) Output — Generates the same report information as the detail report, but presents it as comma-delimited output facilitating its import into a spreadsheet.

• Procedure Level — Generates a report of selected metrics at a subroutine level within the selected routines. 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) JRNUITL2

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 for the output, or enter nothing and press Enter to display the report on your terminal. If you enter a name, the file is created in the manager’s directory. For example:

FileName: monlbl_JRN_dtl.txt

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

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

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

8.3 Sample Line-by-line Monitor Reports

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

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

8.3.1 Line-by-line Detail Report

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

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

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

The following routines have been executed during the run, and have detail statistics available for them.
1) JRNDUMP
2) JRNOPTS
3) JRNSTART
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 /*
3 0 0 0 n (l,usecluster)
85 3 0.000016 0.000016 d showlistclu(.1) q
86 3 0.000008 0.000008 s diroff=($(3+12+1)+10+1)
87 3 0.000072 0.000072 s i="" f s i=$o(l(i)) q:i="" d
88 11 0.001542 0.001542 . w /cup(i+3,1),?3,$S($F(l(i),";"),?)...
89 11 0.028125 0.028220 . w ?(3+12+1),l(i,"info"),?diroff...
90 11 0.000378 0.000895 . w $$GJrnPrefix(l(i))
91 3 0.000027 0.000027 q
92 0 0 0 listjrn(f,list,n) ;list at most...

Total 582 17.258963
Total Lines = 579
Total Lines Hit = 100
Coverage Percentage = 17.27%

This is a partial display of one selected routine.

8.3.2 Line-by-line Summary Report

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

Line-by-Line Monitor

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

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

Press RETURN to continue ...

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

Routine   Lines  LinesHit Percent  RtnLine  Time
JRNOPTS   109    60  55.05%    155   14.172230
JRNSWTCH  249    58  23.29%    69    0.926131
This is the complete sample report.

### 8.3.3 Line-by-line Delimited Output Report

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

Line-by-Line Monitor

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

Enter the number of your choice: 6

The following routines have been executed during the run, and have detail statistics available for them.

1) JRNDUMP
2) JRNSTART
3) JRNOPTS
4) JRNSWTCH
5) JRNUTIL
6) JRNUTIL2

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

Press RETURN to continue ...

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

8.3.4 Line-by-line Procedure Level Report

The following is an example of reporting the detail of the minimal metrics of selected journal utilities by subroutine function.

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

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 ...

<table>
<thead>
<tr>
<th>Tag</th>
<th>RtnLine</th>
<th>Time</th>
<th>TotalTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0.000154</td>
<td>0.000154</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0.000050</td>
<td>0.000050</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>0.000240</td>
<td>0.000232</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>0.136909</td>
<td>0.330301</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>0.188435</td>
<td>0.188435</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This is a portion of the report for one selected routine.
8.4 Line-by-line Monitor Programming Interface

Programmers can also interface with the 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 the %Monitor.System.LineByLine class entry in the online InterSystems Class Reference for more details.
Monitoring Block Collisions Using ^BLKCOL

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

9.1 Using ^BLKCOL

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 in the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide, also samples block collisions, but identifies only the blocks involved. The output of irisstat -D is included in the reports generated by the ^SystemsPerformance utility, as described in the Monitoring Performance Using ^SystemsPerformance chapter of this guide.

When running ^BLKCOL, you can specify the following:

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

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(9668," 18") - ^acctest(9668," 26") (Data)
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@$J,node)=S$getData(S$E(Str,1,$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
This chapter describes the ^SystemPerformance utility, 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 report it produces to the InterSystems Worldwide Response Center (WRC) to help diagnose system problems. ^SystemPerformance is similar to Diagnostic Reports (see “Using the Diagnostic Report” in this guide), 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 (see Running the ^SystemPerformance Utility in this chapter) or schedule runs using the Task Manager in the Management Portal (see Scheduling the ^SystemPerformance Utility with Task Manager in this chapter). In addition, you can add, modify, and delete profiles using the API that is included with the utility.

### 10.1 Running the ^SystemPerformance Utility

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; alternatively, you can specify the output directory as described in the Change Output Directory section in this chapter.

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:


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

   select profile number to run:

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

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

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

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

### 10.1.1 Abort ^SystemPerformance

If you want to stop a running profile, you can abort the collection of data and optionally delete all .log files for the profile with the $Stop^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:

```bash
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:

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

For more information about this command, see $Stop^SystemPerformance("runid") in the [Run ^SystemPerformance Programmatically](#) subsection.

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

### 10.1.2 Run ^SystemPerformance Programmatically

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

**Note:** You can run multiple profiles concurrently.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$run^SystemPerformance(&quot;profile&quot;)</td>
<td>Starts the specified profile. If successful, returns the runid; if unsuccessful, returns 0.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$$\text{litterun}^\text{SystemPerformance}(&quot;profile&quot;)$</td>
<td>Same as $$\text{run}^\text{SystemPerformance}(&quot;profile&quot;)$, except that it does not include operating-system data.</td>
</tr>
<tr>
<td>$$\text{Collect}^\text{SystemPerformance}(&quot;runid&quot;)$</td>
<td>Produces a readable HTML performance report file for the specified runid. If successful, returns 1 and the report filename; if unsuccessful, returns 0 followed by a carat and the reason for the failure.</td>
</tr>
<tr>
<td>$$\text{Preview}^\text{SystemPerformance}(&quot;runid&quot;)$</td>
<td>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.</td>
</tr>
<tr>
<td>$$\text{Stop}^\text{SystemPerformance}(&quot;runid&quot;,[0])$</td>
<td>Stops (aborts) ^SystemPerformance from collecting data for a specified runid and by default deletes the associated .log files produced by the utility. To stop data collection without deleting the .log files and produce an HTML performance report from those log files, include the 0 parameter following the runid. If unsuccessful, the function returns 0 followed by a carat and the reason for the failure; if successful, it returns: 1:2:3:4_1:2:3:4. The “successful” status is made up of two parts separated by an underscore: OS-specific and InterSystems IRIS-specific; within each part, colon-separated values specify: 1. Number of jobs successfully stopped 2. Number of jobs that failed to stop 3. Number of files successfully deleted 4. Number of files not deleted.</td>
</tr>
<tr>
<td>$$\text{waittime}^\text{SystemPerformance}(&quot;runid&quot;)$</td>
<td>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.</td>
</tr>
</tbody>
</table>

In the following example the runid, which is created by the ^SystemPerformance utility, 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.
Monitoring Performance Using ^SystemPerformance

10.2 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, _Pn is appended to the file name, where P identifies it as a preliminary report and n is the number of the preliminary report.

10.3 Scheduling the ^SystemPerformance Utility with Task Manager

This section provides examples using the Task Manager (System Operation > Task Manager) in the Management Portal to schedule ^SystemPerformance to run. For general instructions for how to schedule a task, see the procedure described in Schedule Task Manager in the “Managing InterSystems IRIS” chapter of System Administration Guide.

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:
Example 2: Daily 2-Minute Run

In this example, a task is created to schedule the ^SystemPerformance utility to run a profile named 2mins (which collects performance data for two 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 2MinRun.
   - Description — enter Start 2-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("2mins")
     ```
   - Output file — leave blank; the task has no output (see Change Output Directory for information on customizing the ^SystemPerformance output directory).

2. Click Next. Then enter the following information in the specified fields:
   - How often ... — choose Daily from the drop-down list.
   - Start Date — enter the start date in the text box.
     Click Run once at this time: and enter 12:00:00 in the text box.

3. Click Finish.

10.4 Customizing the ^SystemPerformance Utility

This section describes the tasks you can accomplish with the API:
- Change Output Directory
- Get Version Information
- Manipulate Profiles
### 10.4.1 Change Output Directory

The default output directory for both the log files and the resulting HTML report file is the `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 table.

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>do setlogdir^SystemPerformance(&quot;directory&quot;)</code></td>
<td>Sets the pathname of output directory to <code>directory</code>; if the does not exist, it is created. <strong>Note:</strong> If you do not specify an absolute pathname (for example, <code>C:\Reports</code>), the directory is assumed to be relative to the <code>install-dir\mgr</code> directory.</td>
</tr>
<tr>
<td><code>set x = $$getlogdir^SystemPerformance()</code></td>
<td>Sets variable <code>x</code> equal to the output directory pathname.</td>
</tr>
<tr>
<td><code>do clrlogdir^SystemPerformance()</code></td>
<td>Resets the output directory pathname to the default directory (<code>install-dir\mgr</code>).</td>
</tr>
</tbody>
</table>

### 10.4.2 Get Version Information

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

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

### 10.4.3 Manipulate Profiles

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

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

#### 10.4.3.1 Create New Profiles

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

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

You must specify:
The function returns a 1 if successful and a 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
```

**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:

<table>
<thead>
<tr>
<th><strong>duration</strong></th>
<th>How long the profile should run. The valid formats are &quot;hh:mm&quot;, &quot;hh:&quot;, or mm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interval (optional)</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

The function returns a 1 if successful. If unsuccessful, it returns a 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:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12hours</td>
<td>A 12 hour run sampling every 300 seconds</td>
</tr>
<tr>
<td>90mins</td>
<td>A 90 minute run sampling every 10 seconds</td>
</tr>
</tbody>
</table>

### 10.4.3.2 Edit Profiles

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

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

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profilename</td>
<td>The name of the existing profile you want to edit.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the profile that is displayed in the <code>^SystemPerformance</code> menu.</td>
</tr>
<tr>
<td>interval</td>
<td>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.</td>
</tr>
<tr>
<td>count</td>
<td>The number of times to run the profile.</td>
</tr>
</tbody>
</table>

The function returns a 1 if successful and a 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:

```bash
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:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2minrun</td>
<td>A 5-minute run sampling every 30 seconds</td>
</tr>
</tbody>
</table>

### 10.4.3.3 Copy Profiles

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

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

You must specify:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceprofilename</td>
<td>The name of an existing profile</td>
</tr>
<tr>
<td>targetprofilename</td>
<td>The name of the profile you want to create. This must be double-quoted.</td>
</tr>
</tbody>
</table>
where `sourceprofilename` is the name of an existing profile, and `targetprofilename` must be unique and cannot contain spaces or white-space characters.

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

```
set rc=$$copyprofile^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 in this section of the guide.

### 10.4.3.4 Delete Profiles

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

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

You must specify:

<table>
<thead>
<tr>
<th><code>profilename</code></th>
<th>The name of the profile you want to delete. This must be double-quoted.</th>
</tr>
</thead>
</table>

The function returns a 1 if successful. If unsuccessful, it returns a 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.

## 10.5 Performance Reports Created by `^SystemPerformance Utility`

The `^SystemPerformance` utility generates platform-specific reports as described in this chapter. 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) 2018.1 (Build 508) Fri Jan 26 2018 17:51:22 EDT

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 Red Hat Linux/SuSE Linux Platforms

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

**Table 10–1: InterSystems IRIS Performance Data Report for Microsoft Windows Platforms**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SS</td>
<td>Four samples taken over the course of the run using the ALL^%SS command.</td>
</tr>
<tr>
<td>Configuration *</td>
<td>InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.</td>
</tr>
<tr>
<td>cpf file *</td>
<td>A copy of the currently active configuration file.</td>
</tr>
</tbody>
</table>
| 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 the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| irisstat -D | Eight samples taken at even intervals over the course of the run using the command `iris stat cache --f1 -D10,100`. Following is a brief description of each argument:  
  - `-fl`: basic flags.  
  - `-D10,100`: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds.  
  For more information about the `irisstat` utility, see the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide. For information about monitoring block collisions using the `^BLKCOL` utility, see the “Monitoring Block Collisions Using `^BLKCOL`” chapter of this guide. |
| license * | 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 the Monitoring Performance Using `^mgstat` section of the Monitoring Guide. |
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:

```plaintext
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):

```plaintext
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:

```plaintext
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:

```plaintext
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 default is set.

To reset the default SystemPerformance counter definition, use:

```plaintext
do clrctrdefault^SystemPerformance()
```

To associate a specific SystemPerformance counter definition with an existing profile, use:

```plaintext
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.
### Table 10–2: InterSystems IRIS Performance Data Report for Apple macOS Platforms

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SS</td>
<td>Four samples taken over the course of the run using the <code>ALL%^SS</code> command.</td>
</tr>
<tr>
<td>Configuration</td>
<td>InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.</td>
</tr>
<tr>
<td>cpf file</td>
<td>A copy of the currently active configuration file.</td>
</tr>
<tr>
<td><code>irisstat -c</code></td>
<td>Four samples taken at even intervals over the course of the run using the command <code>iris stat cache -p-1 -c-1 -e1 -m8 -n2 -N127</code>. Following is a brief description of each argument:</td>
</tr>
<tr>
<td></td>
<td>• <code>-p-1</code>: samples the process table to include process and global state information.</td>
</tr>
<tr>
<td></td>
<td>• <code>-c-1</code>: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics.</td>
</tr>
<tr>
<td></td>
<td>• <code>-e1</code>: the SYSLOG error table.</td>
</tr>
<tr>
<td></td>
<td>• <code>-m8</code>: the file table, which includes all IRIS.DAT files and their attributes.</td>
</tr>
<tr>
<td></td>
<td>• <code>-n2</code>: the network structures table, including local-to-remote database mappings.</td>
</tr>
<tr>
<td></td>
<td>• <code>-N127</code>: ECP statistics for both client and server connections.</td>
</tr>
<tr>
<td></td>
<td>For more information about the <code>irisstat</code> utility, see the &quot;Monitoring InterSystems IRIS Using the irisstat Utility&quot; appendix of this guide.</td>
</tr>
</tbody>
</table>
Eight samples taken at even intervals over the course of the run using the command `iris stat cache --f1 -D10,100`. Following is a brief description of each argument:

- `-f1`: 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 the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide. For information about monitoring block collisions using the `^BLKCOL` utility, see the “Monitoring Block Collisions Using ^BLKCOL” chapter of this guide.

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

Table 10–3: InterSystems IRIS Performance Data Report for IBM AIX® Platforms

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SS</td>
<td>Four samples taken over the course of the run using the <code>ALL%^SS</code> command.</td>
</tr>
<tr>
<td>AIX info *</td>
<td>Output from the <code>oslevel</code>, <code>uname -a</code>, <code>prtconf</code>, and <code>lspv</code> commands.</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Configuration</td>
<td>InterSystems IRIS instance name and hostname from the server, the full InterSystems IRIS version string, the licensed customer name, and the license order number.</td>
</tr>
<tr>
<td>cpf file</td>
<td>A copy of the currently active configuration file.</td>
</tr>
<tr>
<td>cpu type</td>
<td>Information on processors installed and whether or not SMT is enabled; output from <code>lsattr -El proc0</code>.</td>
</tr>
<tr>
<td><strong>irisstat -c</strong></td>
<td>Four samples taken at even intervals over the course of the run using the command <code>iris stat cache -p1 -c1 -e1 -m8 -n2 -N127</code>. Following is a brief description of each argument:</td>
</tr>
<tr>
<td></td>
<td>• <code>-p1</code>: samples the process table to include process and global state information.</td>
</tr>
<tr>
<td></td>
<td>• <code>-c1</code>: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics.</td>
</tr>
<tr>
<td></td>
<td>• <code>-e1</code>: the SYSLOG error table.</td>
</tr>
<tr>
<td></td>
<td>• <code>-m8</code>: the file table, which includes all IRIS.DAT files and their attributes.</td>
</tr>
<tr>
<td></td>
<td>• <code>-n2</code>: the network structures table, including local-to-remote database mappings.</td>
</tr>
<tr>
<td></td>
<td>• <code>-N127</code>: ECP statistics for both client and server connections.</td>
</tr>
<tr>
<td>For more information about the <code>irisstat</code> utility, see the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide.</td>
<td></td>
</tr>
<tr>
<td><strong>irisstat -D</strong></td>
<td>Eight samples taken at even intervals over the course of the run using the command <code>iris stat cache --f1 -D10,100</code>. Following is a brief description of each argument:</td>
</tr>
<tr>
<td></td>
<td>• <code>-f1</code>: basic flags.</td>
</tr>
<tr>
<td></td>
<td>• <code>-D10,100</code>: sampling of block collisions every 100 milliseconds over a total sample period of 10 seconds.</td>
</tr>
<tr>
<td>For more information about the <code>irisstat</code> utility, see the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide. For information about monitoring block collisions using the <code>^BLKCOL</code> utility, see the “Monitoring Block Collisions Using <code>^BLKCOL</code>” chapter of this guide.</td>
<td></td>
</tr>
<tr>
<td>df -k</td>
<td>Information about mounted file systems, including mount points, logical volumes, and free space; output from <code>df -k</code> command.</td>
</tr>
<tr>
<td>filesystems</td>
<td>Current <code>/etc/filesystems</code> file.</td>
</tr>
<tr>
<td>ioo -a</td>
<td>Current values of I/O tunable parameters; output from <code>ioo -a</code> command. Included only if the user initiating the <code>^SystemPerformance</code> profile run has root access.</td>
</tr>
</tbody>
</table>
### 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 the Monitoring Performance Using ^mgstat section of the Monitoring Guide.

**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 10–4: InterSystems IRIS Performance Data Report for Red Hat Linux/SuSE Linux Platforms**

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

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 `iris stat cache -p-1 -c-1 -e1 -m8 -n2 -N127`. Following is a brief description of each argument:

- **-p-1**: samples the process table to include process and global state information.
- **-c-1**: samples the Counters section of shared memory to display journal, lock, disk, and resource usage statistics.
- **-e1**: the SYSLOG error table.
- **-m8**: the file table, which includes all 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 the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide.

### irisstat -D

Eight samples taken at even intervals over the course of the run using the command `iris stat cache --f1 -D10,100`. Following is a brief description of each argument:

- **-f1**: 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 the “Monitoring InterSystems IRIS Using the irisstat Utility” appendix of this guide. For information about monitoring block collisions using the `^BLKCOL` utility, see the “Monitoring Block Collisions Using `^BLKCOL`” chapter of this guide.

### df -k

Information about mounted file systems, including mount points, logical volumes, and free space; output from `df -k` command.

### free -m

Memory usage statistics in MB (-m).

### iostat

CPU and disk throughput.

### license

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 the Monitoring Performance Using `^mgstat` section of the Monitoring Guide.
### Section Description

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>Information about the ^SystemPerformance profile that created this log.</td>
</tr>
<tr>
<td><strong>ps:</strong></td>
<td>Four samples taken at even intervals over the course of the run using the command <code>ps -efly</code>.</td>
</tr>
<tr>
<td>sar -d</td>
<td>Disk (block) device throughput and latency statistics.</td>
</tr>
<tr>
<td>sar -u</td>
<td>CPU usage statistics include <code>iowait</code> percentage.</td>
</tr>
<tr>
<td>vmstat -n</td>
<td>CPU, queuing, paging statistics. Only one header is printed (<code>-n</code>).</td>
</tr>
</tbody>
</table>
This chapter 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/.

You must call `^mgstat` from the `%SYS` namespace, and can use the following positional arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sample time</strong></td>
<td>This argument specifies the frequency (in seconds) for sampling counters. If not specified, the default is 2 seconds.</td>
</tr>
<tr>
<td><strong>number of samples</strong></td>
<td>This argument specifies the maximum number of samples to be obtained. If not specified, the default is 10 iterations.</td>
</tr>
<tr>
<td><strong>filename</strong></td>
<td>This argument specifies the filename for the .mgst file that <code>^mgstat</code> generates, relative to <code>install-dir\mgr</code>. If not specified, the default filename is <code>ServerName_InstanceName_Date_Time.mgst</code>.</td>
</tr>
<tr>
<td><strong>page length</strong></td>
<td>If you run <code>^mgstat</code> 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), <code>^mgstat</code> increases it to 5.</td>
</tr>
</tbody>
</table>

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, from 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:

```bash
%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 `^SystemsPerformance` utility (see Change Output Directory in the “Monitoring Performance Using ^SystemsPerformance” chapter in this guide), `^mgstat` uses that output directory.

**Note:** The mgst file is also generated when you run the `^SystemsPerformance` utility and included in the HTML performance report (see the “Monitoring Performance Using ^SystemsPerformance” chapter in this guide).

To ensure minimal impact on system performance, the `^mgstat` utility extracts various counter information from shared memory. If the utility is running and an apparent performance issue occurs, data is available to help you investigate the problem; for assistance with your analysis, contact the InterSystems Worldwide Response Center (WRC), which can provide tasks that automate both the running of `^mgstat` and the purging of files.

Most of the reported data is averaged in per-second values, except as noted in the table below. The generated output file is in a readable, comma-separated value (CSV) format, which is more easily interpreted with a spreadsheet tool such as Microsoft Excel. The first line of the file is a header line which includes the filename and the utility version, as well information about buffer allocation and the version of the product being monitored. The number of columns of data depends on the version of the product: the first two columns are the date and time; the remaining columns are:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glorefs</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>RemGrefs</td>
<td>Remote global references (database accesses). Indicates the number of global references that are generated on behalf of distributed cache cluster application servers.</td>
<td></td>
</tr>
<tr>
<td>GRatio</td>
<td>Ratio of global references to remote global references.</td>
<td></td>
</tr>
<tr>
<td>PhyRds</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Rdratio</td>
<td>Ratio of logical block reads to physical block reads, but zero if physical block reads is zero.</td>
<td></td>
</tr>
<tr>
<td>Gloups</td>
<td>Global updates (sets or kills).</td>
<td></td>
</tr>
<tr>
<td>RemGups</td>
<td>Remote global updates.</td>
<td></td>
</tr>
<tr>
<td>Rourefs</td>
<td>Routine references (includes <code>tag^routine</code>).</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>RemRrefs</td>
<td>Remote routine references.</td>
<td></td>
</tr>
<tr>
<td>RouLaS</td>
<td>Routine loads and saves (fetch from or save to disk).</td>
<td>A high number of routine loads/saves may indicate a performance problem; you can improve the performance by increasing the number of routine buffers.</td>
</tr>
<tr>
<td>RemRLaS</td>
<td>Remote routine loads and saves.</td>
<td></td>
</tr>
<tr>
<td>PhyWrs</td>
<td>Physical writes to disk.</td>
<td></td>
</tr>
<tr>
<td>WDQsz</td>
<td>write daemon Queue size (in blocks).</td>
<td>Not per second.</td>
</tr>
<tr>
<td>WDtmpq</td>
<td>Updated blocks in IRISTEMP.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>WDphase</td>
<td>Phase of the write daemon.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>Wijwri</td>
<td>Number of 256-KB blocks written to the WIJ.</td>
<td>This is non-zero when the WD is writing data to the WIJ.</td>
</tr>
<tr>
<td>RouCMs</td>
<td>Number of Routine Cache Misses.</td>
<td></td>
</tr>
<tr>
<td>Jrnwrts</td>
<td>Number of blocks written to journals.</td>
<td></td>
</tr>
<tr>
<td>GblSz†</td>
<td>Number of seizes on the global resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>pGblNsiz†</td>
<td>Percentage of NSeizes on the global resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>pGblAsz†</td>
<td>Percentage of ASeizes on the global resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>RouSz†</td>
<td>Number of seizes on the routine resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>pRouAsz†</td>
<td>Percentage of ASeizes on the routine resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>ObjSz†</td>
<td>Number of seizes on the object resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
</tbody>
</table>
### 11.1 Considering Seizes, ASeizes, and NSeizes

A **Seize** occurs whenever a job needs exclusive access on a given resource to guarantee that an update occurs without interference from other processes. If the Seize is not immediately satisfied, the update is postponed until it is satisfied.

On a single-CPU system, the process immediately hibernates (because it cannot do anything until the process holding the resource relinquishes it, which does not occur until after its own update completes).

On a multiple-CPU system, the process enters a holding loop in the “hope” that it will gain the resource in a reasonable time, thus avoiding the expense of hibernating. If the process gets access to the resource during the hold loop, the loop immediately exits and the process continues with its update; upon completing the update, the process relinquishes the resource for other processes that may be waiting for it; this is an **Aseize**. If, at the end of the hold loop, the resource is still

---

**Table:**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pObjAsz†</td>
<td>Percentage of ASeizes on the object resource; see “Considering Seizes, ASeizes, and NSeizes” in this chapter.</td>
<td></td>
</tr>
<tr>
<td>ActECP</td>
<td>Number of active ECP connections.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>Addblk</td>
<td>Number of blocks added to ECP Client’s cache.</td>
<td></td>
</tr>
<tr>
<td>PrgBufL</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>PrgSrvR</td>
<td>Number of blocks purged from ECP Client’s cache by ECP server.</td>
<td></td>
</tr>
<tr>
<td>BytSnt</td>
<td>Number of bytes sent as an ECP Client.</td>
<td></td>
</tr>
<tr>
<td>BytRcd</td>
<td>Number of bytes received as an ECP Client.</td>
<td></td>
</tr>
<tr>
<td>WDPass</td>
<td>WD cycle since startup.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>IJUCnt</td>
<td>Number of jobs that WD is waiting for to continue this cycle.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>IJULock</td>
<td>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.</td>
<td>Not per second.</td>
</tr>
<tr>
<td>PPGrefs</td>
<td>The count of all process private global accesses.</td>
<td></td>
</tr>
<tr>
<td>PPGupds</td>
<td>The count of all process private global updates.</td>
<td></td>
</tr>
</tbody>
</table>

* 0 is displayed unless this is an ECP configuration.

† Seize statistics are not included in the report unless the CollectResourceStats parameter in the CPF is enabled.
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.
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.

### 12.1 Base Metrics

All of the collected metrics are defined in four %SerialObject classes in SYS.History. These same classes are used as the basis for the Interval, Hourly, and Daily databases, so all of the properties are defined as %Numeric types to allow for decimal values in the summaries.

The performance related metrics are defined in:

- SYS.History.Performance — The properties in this class are general performance metrics like global references and routine calls.
  
  **Note:** These properties are all “counter” types and the interval data is collected as deltas, which represent the change in the counter over the last interval. When this data is summarized into hourly and daily values, the data is normalized to per-second rates

- SYS.History.WriteDaemon — The properties in this class describe the performance of write daemon cycles. The system automatically keeps track of the last 20 write daemon cycles, and the History Monitor stores the data for the cycles that occurred in each interval. Typically, there are multiple cycles within each interval.
The system usage metrics are defined in:

- SYS.History.SystemUsage — The properties in this class track how busy the system is but do not tend to change as quickly or dramatically as the performance data, 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.

### 12.2 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 in the “Using System Monitor” chapter of this guide for information about these procedures.

The detailed interval collection of data is defined in two persistent classes:

- SYS.History.PerfData — Includes the performance and write daemon classes as embedded objects.
- SYS.History.SysData — Includes the system usage and database classes.

The corresponding %Monitor classes must be activated in Application Monitor in order to collect data and build the history data:


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.

### 12.3 Summaries

The %Monitor.System.HistoryPerf and %Monitor.System.HistorySys classes, as executed by Application Monitor, also create the hourly and daily summaries at the end of each day. The summaries are defined as the persistent classes SYS.History.Hourly and SYS.History.Daily; they include all four of the base classes as embedded objects.

For each metric property, the system may calculate the average, maximum (high-water mark), standard deviation, minimum, median, or total for each hour and for the whole day. The summary functions are selectable (or may be disabled) for each base class (SYS.History.Performance, SYS.History.WriteDaemon, SYS.History.SystemUsage, or SYS.History.Database) and for each summary period class, using the SetSummary() method of each of the base classes. By default, the History Monitor calculates average, maximum and standard deviation for each class for both hourly and daily summaries.

**Note:** The counter properties of the SYS.History.Performance class are normalized to per second rates for these calculations (except Total).
Purging Data

After creating the summaries, Application Monitor automatically purges the interval and hourly databases. The default is seven (7) days for interval data and 60 days for hourly data, but these may be changed using the `SetPurge()` method in `SYS.History.PerfData` and `SYS.History.Hourly` classes. The `SYS.History.Daily` data is not automatically purged, but can be done manually using the `SYS.History.Daily:Purge()` method.

12.4 Accessing the Data

Since the database is defined as persistent classes, the data is available using standard SQL or persistent object access. Using the SQL browser in the Management Portal is a quick and easy way to see the various SQL schemas/tables that are created, including the individual property values.

There are several basic queries implemented in each of the persistent classes in `SYS.History` (`SYS.History.PerfData`, `SYS.History.SysData`, `SYS.History.Hourly`, and `SYS.History.Daily`) that can be used to access the individual tables for a date range; for more information about the queries, see the class reference documentation.

There are also several `Export()` methods provided for each persistent class so that the individual tables can be exported to files in CSV format, suitable for use with a spreadsheet such as Microsoft Excel. In particular, the `SYS.History.PerfData:Export()` method creates a file that is very similar in format to that created by the `^mgstat` utility (for more information, see the “Monitoring Performance Using `^mgstat`” chapter of this guide).

12.5 Adding User-Defined Metrics

You can add user-defined metrics to the History Monitor (`SYS.History` package):

1. Create a class, or multiple classes, that inherit from `SYS.History.Adaptor` and add `%Numeric` properties to define the metrics.
   
   **Note:** User-written classes must be in the `%SYS` namespace, and should begin with “Z” or “z” to prevent naming conflicts with system classes and problems during upgrades.

2. Code the `Sample()` method to instantiate the class and provide periodic values for each property. This method is called when the interval data is collected.

3. When you compile your class, it is added as an embedded object to an interval persistent class in `SYS.History`. You can choose where and when it is collected using the `INTERVAL` parameter provided in `SYS.History.Adaptor` class. This selects which interval class it is added to and which `%Monitor` class does the collection, as shown in the following table:

<table>
<thead>
<tr>
<th>INTERVAL Selected</th>
<th>Interval Class Used</th>
<th>%Monitor Class Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>“User” (default)</td>
<td>SYS.History.User</td>
<td>%Monitor.System.HistoryUser</td>
</tr>
</tbody>
</table>

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” in the “Defining Persistent Classes” chapter of Defining and Using Classes.
Monitoring InterSystems IRIS Using SNMP

This appendix 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](https://example.com)) 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:

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, iacsnmp.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 indices.
- 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 indices.
- The notifications (traps) are defined as individual entries at the same hierarchical level as the “table”. For more information, see InterSystems IRIS SNMP Traps in this appendix.
- 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 in this appendix.

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 in the *InterSystems Class Reference* for details of the `CreateMIB()` method parameters.

The method creates a branch of the private enterprise MIB tree for a specific application defined in the `%Monitor` database. In addition to creating the actual MIB file for the application, the method also creates an internal outline of the MIB tree. The 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* in this appendix.

### 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)**

<table>
<thead>
<tr>
<th>Trap Name (Number)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisStart (1)</td>
<td>The InterSystems IRIS instance has been started.</td>
</tr>
<tr>
<td>irisStop (2)</td>
<td>The InterSystems IRIS instance is in the process of shutting down.</td>
</tr>
<tr>
<td>irisDBExpand (3)</td>
<td>An InterSystems IRIS database has expanded successfully.</td>
</tr>
<tr>
<td>irisDBOutOfSpace (4)</td>
<td>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.</td>
</tr>
<tr>
<td>irisDBStatusChange (5)</td>
<td>The read/write status of an InterSystems IRIS database has been changed.</td>
</tr>
<tr>
<td>irisDBWriteFail (6)</td>
<td>A write to an InterSystems IRIS database has failed. It includes the InterSystems IRIS error code for the failed write.</td>
</tr>
<tr>
<td>irisWDStop (7)</td>
<td>The write daemon for an InterSystems IRIS instance has stalled.</td>
</tr>
<tr>
<td>irisWDPanic (8)</td>
<td>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.</td>
</tr>
<tr>
<td>irisLockTableFull (9)</td>
<td>The lock table for an InterSystems IRIS instance is full, which causes subsequent Locks to fail.</td>
</tr>
<tr>
<td>irisProcessFail (10)</td>
<td>A process has exited InterSystems IRIS abnormally (due to an access violation). For detailed information, see the messages.log file.</td>
</tr>
<tr>
<td>irisECPTroubleDSrv (11)</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

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.

A "severe " error has been logged in the messages.log file. This trap includes the error message defined in irisSysErrorMsg.

A request for a license has exceeded the number of licenses currently available or allowed.

An entry posted in the Interoperability Event Log.

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>
<thead>
<tr>
<th>Auxiliary Object Name (Number)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisDBWriteError (1)</td>
<td>The InterSystems IRIS-specific error code for a failed database write. Possible values are: &lt;DATABASE&gt;, &lt;DISKHARD&gt;, &lt;BLOCKNUMBER&gt;, &lt;FILEFULL&gt; or &lt;DATABASE MAP LABEL&gt;.</td>
</tr>
<tr>
<td>irisApp (2)</td>
<td>A short text string (maximum of 20 characters) that identifies the application that generated (or was the source of) an irisAppAlert trap.</td>
</tr>
<tr>
<td>irisAppSeverity (3)</td>
<td>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).</td>
</tr>
<tr>
<td>irisAppText (4)</td>
<td>A text string description (maximum of 1024 characters) of the problem, error, or event that caused the irisAppAlert trap.</td>
</tr>
</tbody>
</table>

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 in the “Using System Monitor” chapter of this guide) that you can query via SNMP. The Application Monitor includes only properties with %Monitor data types in the SNMP data.

Example Sample Class

The following is the sample class for this example:

```java
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
in the “Using System Monitor” chapter of this guide, and do the following:

• Select option 2, Manage Monitor Classes.

• Select option 3, Register Monitor System Classes.

When you compile the sample class, it creates the SNMP.Sample.Example class to store the sample data.

Important: Do not delete generated sample classes explicitly; if you select both the Application Monitor and generated
sample classes for deletion, the sample class routines remain although the monitor class routines are deleted,
which causes an error. To ensure that all sample class routines are properly removed, delete only the
Application Monitor class that generated it; when you delete the monitor class both the monitor class and
generated sample class, as well as related routines for both classes, are deleted. For example, to delete a
sample class (for example, SNMP.Sample.Example), use the Management Portal or Atelier 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 in the “Using the System Monitor” chapter of this guide.

Example of Creating a User MIB

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

The input parameters for the method are similar to the following:

CreateMIB("MyApp","USER",99990,1,"mycorp","myapp","mc","MC-MYAPP","Unknown",1)

Important: Do not use 99990 as the Enterprise ID for production; each organization should register with the IANA for their own ID.

Example:

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.

```plaintext
--
-- 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 "" ::= { mcMyApp 1 }

mycorp OBJECT IDENTIFIER ::= { enterprises 16563 }
```

Monitoring Guide 131
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
Monitoring InterSystems IRIS Using Web Services

This appendix 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 the “Monitoring InterSystems IRIS Using SNMP” appendix of this guide), 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 as follows:
In more advanced cases, the web client subscribes an event sink service, possibly running on another InterSystems IRIS instance. For example:

Your event sink web service can perform any processing needed by the business.

Note that Studio provides the SOAP Wizard, which can generate web services and web clients from a WSDL. For details on using this wizard, see *Creating Web Services and Web Clients*. (Many third-party technologies also provide similar wizards.)

**B.2 Support Details**

InterSystems supports the following parts of the WS-Management specification:

- `wxf:Get`
- `wsen:Enumerate`
- `wsen:Pull`
- `wsen:Release`
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:


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 example:


Similarly, the WSDL for this web service is available at the following URL:


B.4 Web Methods of the Monitoring Web Service

The `SYS.WSMon.Service` class provides the following web methods:

**EnumBuffer()**

```plaintext
method EnumBuffer() as %XML.DataSet

Returns an instance of %XML.DataSet that enumerates the statistics for all buffer sizes. For this instance, the dataset uses the Sample() class query of the `SYS.Stats.Buffer` class.

For information on working with %XML.DataSet, see the chapter “Using Datasets in SOAP Messages” in *Creating Web Services and Web Clients* or see the class reference for %XML.DataSet.

Also see the class reference for `SYS.Stats.Buffer`.
```

**EnumDatabase()**

```plaintext
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()**

```plaintext
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. For SYS.WSMon.EventSink, you might specify location as the following:

```
http://server:port/csp/sys/SYS.WSMon.EventSink.cls
```

Where server is the server on which 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()**

```plaintext
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()**

```plaintext
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()**

```plaintext
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:

```plaintext
```

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 the section “Web Methods of the Monitoring Web Service” and see the class reference for the return types.

The following shows an example Terminal session:
%SYS>set client=%class(SYS.WSMon.Client).%New()
%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:

```
http://server:port/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:

```python
USER>set $namespace = "%sys"
%SYS>set client=%class(SYS.WSMon.Client).%New()
%SYS>set eventsinklocation="http://localhost:52773/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, use the SOAP Wizard in Studio to generate a web service from the following WSDL:


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 on using this wizard, see Creating Web Services and Web Clients.

Then modify the CacheEventSink() method in the generated web service to include your custom logic.
Monitoring InterSystems IRIS Using REST API

Every InterSystems IRIS® data platform instance contains a REST interface that provides statistics about the instance. The 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.

This appendix describes the metrics the /api/monitor service provides. 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.

**Note:** For an introduction to creating and using REST interfaces, see *First Look: Developing Rest Interfaces with InterSystems Products*.

C.1 /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 the Securing REST Services chapter in *Creating REST Services*.

This API has the following two endpoints:

- **/metrics Endpoint**, which returns all instance metrics, and can be configured to return specific application metrics.
- **/alerts Endpoint**, 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.1.1 /metrics Endpoint

The /metrics endpoint returns a list of metrics, which are described in the Metric Descriptions section. The Create Application Metrics section 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.1.1.1 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.

All the included InterSystems IRIS metrics are listed in the table below. Metric names with a label appear here with line breaks to improve readability.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| iris_cpu_pct {id="ProcessType"}                 | Percent of CPU usage by InterSystems IRIS process type. *Process Type* can be any of the following: ECPWorker, ECPCliR, ECPCliW, ECPSrvR, ECPSrvW, LICENSESRV, WDSLAVE, WRTDMN, JRNDMN, GARCOL, CSPDMN, CSPSRV, ODBCSR, MirrorMaster, MirrorFri, MirrorBack, MirrorPre, MirrorSvrR, MirrorJrnR, MirrorSK, MirrorComm  
(For more information about InterSystems IRIS Processes, see Securing InterSystems Products and Operating System Resources.) |
<p>| iris_cpu_usage                                   | Percent of CPU usage for all programs on the operating system                                  |
| iris_csp_activity {id=&quot;IPaddress:port&quot;}         | Number of web requests served by the Web Gateway Server since it was started                   |
| iris_csp_actual_connections {id=&quot;IPAddress:port&quot;}| Number of current connections to this server by the Web Gateway Server                         |
| iris_csp_gateway_latency {id=&quot;IPaddress:port&quot;}  | 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=&quot;IPaddress:port&quot;}| Number of current connections to this server by the Web Gateway Server that are processing a web request |
| iris_csp_private_connections {id=&quot;IPaddress:port&quot;}| 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=&quot;database&quot;}       | Amount by which to expand <em>database</em>, in megabytes                                              |
| iris_db_free_space {id=&quot;database&quot;}              | Free space available in <em>database</em>, in megabytes (This metric is only updated once per day, and may not reflect recent changes.) |</p>
<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iris_db_latency</code></td>
<td>Amount of time to complete a random read from database, in milliseconds</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_db_max_size_mb</code></td>
<td>Maximum size to which database can grow, in megabytes</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_db_size_mb</code></td>
<td>Size of database, in megabytes</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;,dir=&quot;path&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_directory_space</code></td>
<td>Free space available on the database directory’s storage volume, in megabytes</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;,dir=&quot;path&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_disk_percent_full</code></td>
<td>Percent of space filled on the database directory’s storage volume</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;,dir=&quot;path&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_ecp_conn</code></td>
<td>Total number of active client connections on this ECP application server</td>
</tr>
<tr>
<td><code>iris_ecp_conn_max</code></td>
<td>Maximum active client connections from this ECP application server</td>
</tr>
<tr>
<td><code>iris_ecp_connections</code></td>
<td>Number of servers synchronized when this ECP application server synchronizes with its configured ECP data servers</td>
</tr>
<tr>
<td><code>iris_ecp_latency</code></td>
<td>Latency between the ECP application server and the ECP data server, in milliseconds</td>
</tr>
<tr>
<td><code>iris_ecps_conn</code></td>
<td>Total active client connections to this ECP data server per second</td>
</tr>
<tr>
<td><code>iris_ecps_conn_max</code></td>
<td>Maximum active client connections to this ECP data server</td>
</tr>
<tr>
<td><code>iris_glo_a_seize_per_sec</code></td>
<td>Number of Aseizes on the global resource per second (For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using ^mgstat” section of Monitoring Guide.)</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_glo_n_seize_per_sec</code></td>
<td>Number of Nseizes on the global resource per second (For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using ^mgstat” section of Monitoring Guide.)</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_glo_ref_per_sec</code></td>
<td>Number of references to globals located on local databases per second</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_glo_ref_rem_per_sec</code></td>
<td>Number of references to globals located on remote databases per second</td>
</tr>
<tr>
<td><code>{id=&quot;database&quot;}</code></td>
<td></td>
</tr>
<tr>
<td><code>iris_glo_seize_per_sec</code></td>
<td>Number of seizes on the global resource per second (For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using ^mgstat” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>iris_glo_update_per_sec</td>
<td>Number of updates (SET and KILL commands) to globals located on local databases per second</td>
</tr>
<tr>
<td>iris_glo_update_rem_per_sec</td>
<td>Number of updates (SET and KILL commands) to globals located on remote databases per second</td>
</tr>
<tr>
<td>iris_jrn_block_per_sec</td>
<td>Journal blocks written to disk per second</td>
</tr>
<tr>
<td>iris_jrn_free_space</td>
<td>Free space available on each journal directory’s storage volume, in megabytes. JournalType can be WIJ, primary, or secondary</td>
</tr>
<tr>
<td>iris_jrn_size</td>
<td>Current size of each journal file, in megabytes. JournalType can be WIJ, primary, or secondary</td>
</tr>
<tr>
<td>iris_license_available</td>
<td>Number of licenses not currently in use</td>
</tr>
<tr>
<td>iris_license_consumed</td>
<td>Number of licenses currently in use</td>
</tr>
<tr>
<td>iris_license_percent_used</td>
<td>Percent of licenses currently in use</td>
</tr>
<tr>
<td>iris_log_reads_per_sec</td>
<td>Logical reads per second</td>
</tr>
<tr>
<td>iris_obj_a_seize_per_sec</td>
<td>Number of Aseizes on the object resource per second</td>
</tr>
<tr>
<td></td>
<td>(For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using ^mgstat” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_obj_del_per_sec</td>
<td>Number of objects deleted per second</td>
</tr>
<tr>
<td>iris_obj_hit_per_sec</td>
<td>Number of object references per second, in process memory</td>
</tr>
<tr>
<td>iris_obj_load_per_sec</td>
<td>Number of objects loaded from disk per second, not in shared memory</td>
</tr>
<tr>
<td>iris_obj_miss_per_sec</td>
<td>Number of object references not found in memory per second</td>
</tr>
<tr>
<td>iris_obj_new_per_sec</td>
<td>Number of objects initialized per second</td>
</tr>
<tr>
<td>iris_obj_seize_per_sec</td>
<td>Number of seizes on the object resource per second</td>
</tr>
<tr>
<td></td>
<td>(For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using ^mgstat” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_page_space_percent_used</td>
<td>Percent of maximum allocated page file space used</td>
</tr>
<tr>
<td>iris_phys_mem_percent_used</td>
<td>Percent of physical memory (RAM) currently in use</td>
</tr>
<tr>
<td>iris PHYS_READS_PER_SEC</td>
<td>Physical database blocks read from disk per second</td>
</tr>
<tr>
<td>iris_phys_WRITES_PER_SEC</td>
<td>Physical database blocks written to disk per second</td>
</tr>
<tr>
<td>iris_process_count</td>
<td>Total number of active InterSystems IRIS processes</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>iris_rtn_a_seize_per_sec</td>
<td>Number of Aseizes on the routine resource per second (For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using &quot;mgstat&quot; section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_rtn_call_local_per_sec</td>
<td>Number of local routine calls per second to globals located on remote databases per second</td>
</tr>
<tr>
<td>iris_rtn_call_miss_per_sec</td>
<td>Number of routines calls not found in memory per second</td>
</tr>
<tr>
<td>iris_rtn_call_remote_per_sec</td>
<td>Number of remote routine calls per second</td>
</tr>
<tr>
<td>iris_rtn_load_per_sec</td>
<td>Number of routines locally loaded from or saved to disk per second</td>
</tr>
<tr>
<td>iris_rtn_load_rem_per_sec</td>
<td>Number of routines remotely loaded from or saved to disk per second</td>
</tr>
<tr>
<td>iris_rtn_seize_per_sec</td>
<td>Number of seizes on the routine resource per second (For more information, see Considering Seizes, ASeizes, and NSeizes in the “Monitoring Performance Using &quot;mgstat&quot; section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_sam_get_db_sensors_seconds</td>
<td>Amount of time it took to collect iris_db* sensors, in seconds</td>
</tr>
<tr>
<td>iris_sam_get_jrn_sensors_seconds</td>
<td>Amount of time it took to collect iris_jrn* sensors, in seconds</td>
</tr>
<tr>
<td>iris_smh_available {id=&quot;purpose&quot;}</td>
<td>Shared memory available by purpose, in kilobytes (For more information, including a list of identifiers for purpose, see Generic (Shared) Memory Heap Usage in “Monitoring InterSystems IRIS Using the Management Portal” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_smh_percent_full {id=&quot;purpose&quot;}</td>
<td>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 in “Monitoring InterSystems IRIS Using the Management Portal” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_smh_total</td>
<td>Shared memory allocated for current instance, in kilobytes</td>
</tr>
<tr>
<td>iris_smh_total_percent_full</td>
<td>Percent of allocated shared memory in use for current instance</td>
</tr>
<tr>
<td>iris_smh_used {id=&quot;purpose&quot;}</td>
<td>Percent of shared memory in use by purpose, in kilobytes (For more information, including a list of identifiers for purpose, see Generic (Shared) Memory Heap Usage in “Monitoring InterSystems IRIS Using the Management Portal” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_system_alerts</td>
<td>The number of alerts posted to the messages log since system startup</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>iris_system_alerts_new</td>
<td>Whether new alerts are available on the /api/monitor/alerts endpoint, as a Boolean</td>
</tr>
<tr>
<td>iris_system_state</td>
<td>A number representing the system monitor health state (For more information, see System Monitor Health State in the “Using System Monitor” section of Monitoring Guide.)</td>
</tr>
<tr>
<td>iris_trans_open_count</td>
<td>Number of open transactions on the current instance</td>
</tr>
<tr>
<td>iris_trans_open_secs</td>
<td>Average duration of open transactions on the current instance, in seconds</td>
</tr>
<tr>
<td>iris_trans_open_secs_max</td>
<td>Duration of longest currently open transaction on the current instance, in seconds</td>
</tr>
<tr>
<td>iris_wd_buffer_redirty</td>
<td>Number of database buffers the write daemon wrote during the most recent cycle that were also written in prior cycle</td>
</tr>
<tr>
<td>iris_wd_buffer_write</td>
<td>Number of database buffers the write daemon wrote during its most recent cycle</td>
</tr>
<tr>
<td>iris_wd_cycle_time</td>
<td>Amount of time the most recent write daemon cycle took to complete, in milliseconds</td>
</tr>
<tr>
<td>iris_wd_proc_in_global</td>
<td>Number of processes actively holding global buffers at start of the most recent write daemon cycle</td>
</tr>
<tr>
<td>iris_wd_size_write</td>
<td>Size of database buffers the write daemon wrote during its most recent cycle, in kilobytes</td>
</tr>
<tr>
<td>iris_wd_sleep</td>
<td>Amount of time that the write daemon was inactive before its most recent cycle began, in milliseconds</td>
</tr>
<tr>
<td>iris_wd_temp_queue</td>
<td>Number of in-memory buffers the write daemon used at the start of its most recent cycle</td>
</tr>
<tr>
<td>iris_wd_temp_write</td>
<td>Number of in-memory buffers the write daemon wrote during its most recent cycle</td>
</tr>
<tr>
<td>iris_wdwij_time</td>
<td>Amount of time the write daemon spent writing to the WIJ file during its most recent cycle, in milliseconds</td>
</tr>
<tr>
<td>iris_wdwij_write_time</td>
<td>Amount of time the write daemon spent writing buffers to databases during its most recent cycle, in milliseconds</td>
</tr>
<tr>
<td>iris_wij_writes_per_sec</td>
<td>WIJ physical block writes per second</td>
</tr>
</tbody>
</table>

### C.1.1.2 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 `$\text{OK}$` 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:

   ```
   /// 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 $\text{OK}$
      }
   }
   ```

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
   ```

6. Review the output of the `/metrics` endpoint by pointing your browser to `http://localhost:52773/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.1.2 /alerts Endpoint**

The `/alerts` endpoint fetches the most recent alerts from the alerts.log file and returns them in JSON format, such as:

```json
{"time":"2019-08-15T10:36:38.313Z","severity":2,\n"message":"Failed to allocate 1150MB shared memory using large pages. Switching to small pages."}
```
When `/alerts` is called, it returns the alerts that have been generated since the previous time `/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 the Using Log Monitor chapter of this guide.
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:
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 -p1

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 -f1 -p1 -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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a[0/1]</td>
<td>Displays all information described in this table.</td>
</tr>
</tbody>
</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.  

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c[bits]</code></td>
<td>Displays counters, which are statistics on system performance. You can specify a combination of the following bits:</td>
</tr>
<tr>
<td></td>
<td>• 1 (global)</td>
</tr>
<tr>
<td></td>
<td>• 2 (network)</td>
</tr>
<tr>
<td></td>
<td>• 4 (lock)</td>
</tr>
<tr>
<td></td>
<td>• 8 (optim)</td>
</tr>
<tr>
<td></td>
<td>• 16 (terminal)</td>
</tr>
<tr>
<td></td>
<td>• 32 (syntab)</td>
</tr>
<tr>
<td></td>
<td>• 64 (journal)</td>
</tr>
<tr>
<td></td>
<td>• 128 (disk i/o)</td>
</tr>
<tr>
<td></td>
<td>• 256 (cluster)</td>
</tr>
<tr>
<td></td>
<td>• 262144 (bshash)</td>
</tr>
<tr>
<td></td>
<td>• 2097152 (job cmd)</td>
</tr>
<tr>
<td></td>
<td>• 4194304 (sem)</td>
</tr>
<tr>
<td></td>
<td>• 8388608 (async disk i/o)</td>
</tr>
<tr>
<td></td>
<td>• 16777216 (fsync)</td>
</tr>
<tr>
<td></td>
<td>• 33554432 (obj class)</td>
</tr>
<tr>
<td></td>
<td>• 67108864 (wd)</td>
</tr>
<tr>
<td></td>
<td>• 134217728 (bigstr)</td>
</tr>
<tr>
<td></td>
<td>• 268435456 (swd)</td>
</tr>
<tr>
<td></td>
<td>• 536870912 (sort)</td>
</tr>
<tr>
<td></td>
<td>• 1073741824 (symsave)</td>
</tr>
<tr>
<td></td>
<td>• 2147483648 (freeblkpool)</td>
</tr>
<tr>
<td><code>-d[pid,opt]</code></td>
<td>Creates dump of InterSystems IRIS processes. You can specify the following options:</td>
</tr>
<tr>
<td></td>
<td>• 0 (full); default</td>
</tr>
<tr>
<td></td>
<td>• 1 (partial)</td>
</tr>
<tr>
<td><code>-e[0/1/2]</code></td>
<td>Displays the InterSystems IRIS system error log (see InterSystems IRIS System Error Log in the “Monitoring InterSystems IRIS Using the Management Portal” chapter); <code>-e2</code> displays additional process information (in hex).</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-f[bits]</td>
<td>Displays global module flags. You can specify a combination of the following bits:</td>
</tr>
<tr>
<td></td>
<td>• 1 (basic)</td>
</tr>
<tr>
<td></td>
<td>• 64 (resources)</td>
</tr>
<tr>
<td></td>
<td>• 128 (with detail)</td>
</tr>
<tr>
<td></td>
<td>• 256 (account detail)</td>
</tr>
<tr>
<td></td>
<td>• 512 (incstrtab)</td>
</tr>
<tr>
<td></td>
<td>• 1024 (audit)</td>
</tr>
<tr>
<td>-g[0/1]</td>
<td>Displays ^GLOSTAT information; for information see the “Gathering Global Activity Statistics Using ^GLOSTAT” chapter in this document.</td>
</tr>
<tr>
<td>-h</td>
<td>Displays irisstat usage information.</td>
</tr>
<tr>
<td>-j[0/1/2/3/4/5/6]</td>
<td>Displays the journal system master structure, which lists information about journaling status. –j32 displays mirror server information.</td>
</tr>
<tr>
<td>-k</td>
<td>Displays information about prefetch daemons used by the $PREFETCHON function; see $PREFETCHON in the ObjectScript Reference.</td>
</tr>
<tr>
<td>-l[bits]</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• 1 (all)</td>
</tr>
<tr>
<td></td>
<td>• 2 (cluster)</td>
</tr>
<tr>
<td></td>
<td>• 4 (ECP server)</td>
</tr>
<tr>
<td></td>
<td>• 8 (ECP client)</td>
</tr>
<tr>
<td></td>
<td>• 16 (block contents)</td>
</tr>
<tr>
<td></td>
<td>• 32, but not 1 (most recently used (MRU) order)</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>See also –b.</td>
</tr>
<tr>
<td>-m[0/1/3/4/8/16]</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• 3 (additional details)</td>
</tr>
<tr>
<td></td>
<td>• 4 (volume queues)</td>
</tr>
<tr>
<td></td>
<td>• 8 (disk device id table)</td>
</tr>
<tr>
<td></td>
<td>• 16 (systems remotely mounting this database)</td>
</tr>
<tr>
<td>-n[0/1]</td>
<td>Displays information about network structures and local/remote SFN translations; irisstat –n-1 also displays namespace structures.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-o1</td>
<td>Clears the resource statistics displayed by <code>irisstat -c</code> to reestablish a base situation without rebooting InterSystems IRIS. No output is produced.</td>
</tr>
<tr>
<td>-p[bits]</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• 2 (swcheck)</td>
</tr>
<tr>
<td></td>
<td>• 4 (pstate and %SS)</td>
</tr>
<tr>
<td></td>
<td>• 5 (NT mailbox locks); Windows only</td>
</tr>
<tr>
<td></td>
<td>• 8 (js sum)</td>
</tr>
<tr>
<td></td>
<td>• 16 (js list)</td>
</tr>
<tr>
<td></td>
<td>• 32 (grefcnt info)</td>
</tr>
<tr>
<td></td>
<td>• 64 (gstatebits)</td>
</tr>
<tr>
<td></td>
<td>• 128 (gstate summary)</td>
</tr>
<tr>
<td></td>
<td>• 256 (jrnhib)</td>
</tr>
<tr>
<td></td>
<td>• 512 (transaction summary)</td>
</tr>
<tr>
<td></td>
<td>• 1024 (pidflags)</td>
</tr>
<tr>
<td></td>
<td>• 2048 (pgbdbsav); additionally dumps pgshared table</td>
</tr>
<tr>
<td></td>
<td>• 4096 (freeblk table)</td>
</tr>
<tr>
<td>-q[0/1]</td>
<td>Displays information about hibernation semaphores.</td>
</tr>
<tr>
<td>-s[dir]</td>
<td>Specifies the directory containing the <code>irisstat</code> executable when running the command from other than the <code>mgr</code> or <code>bin</code> directories.</td>
</tr>
<tr>
<td>-t[seconds]</td>
<td>Runs <code>irisstat</code> repeatedly in a loop every <code>seconds</code> seconds until halted. Only the global module flags section is displayed, as when <code>-f1</code> is specified.</td>
</tr>
<tr>
<td>-u[bits]</td>
<td>Displays information about InterSystems IRIS locks stored in the lock table (see Monitoring Locks in the “Monitoring InterSystems IRIS Using the Management Portal” chapter of this guide). You can specify a combination of the following bits:</td>
</tr>
<tr>
<td></td>
<td>• 1 (summary)</td>
</tr>
<tr>
<td></td>
<td>• 2 (waiters)</td>
</tr>
<tr>
<td></td>
<td>• 4 (intermediate)</td>
</tr>
<tr>
<td></td>
<td>• 8 (detail)</td>
</tr>
<tr>
<td></td>
<td>• 16 (watermark)</td>
</tr>
<tr>
<td></td>
<td>• 32 (buddy memory)</td>
</tr>
<tr>
<td></td>
<td>• 64 (resource info)</td>
</tr>
</tbody>
</table>
### Running irisstat with Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-v1</code></td>
<td>Ensures that the InterSystems IRIS executable associated with the shared memory segment <code>irisstat</code> is being run on and the <code>irisstat</code> executable are from the same version; if not, <code>irisstat</code> will not run.</td>
</tr>
<tr>
<td><code>-w[bits]</code></td>
<td>Displays information about BDBs in write daemon queues.</td>
</tr>
<tr>
<td><code>-B[0/1]</code></td>
<td>Displays, in hex, the contents of blocks held in GBFSPECQ.</td>
</tr>
<tr>
<td><code>-C[0/1]</code></td>
<td>Displays configuration information for inter-job communication (IJC) devices.</td>
</tr>
</tbody>
</table>
| `-D[secs],[msecs],[0]` | Displays resource statistics over an interval of `secs` seconds. Sample block collisions ever ‘msec’ milliseconds.  
  **Note:** Resource information same as –c.  
  The `^BLKCOL` utility, described in the “Monitoring Block Collisions Using `^BLKCOL`” chapter of this guide, provides more detailed information about block collisions. |
| `-E[bits]` | Displays status of cluster on platforms that support clustering. You can specify a combination of the following bits:  
  - 1 (vars)  
  - 2 (write daemon locks)  
  - 4 (enqinuse)  
  - 8/16 (allenq) |
| `-G[dbd]` | 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. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N[value]</td>
<td>Displays ECP network information. You can specify a combination of the following values:</td>
</tr>
<tr>
<td></td>
<td>• 1 (client)</td>
</tr>
<tr>
<td></td>
<td>• 2 (server)</td>
</tr>
<tr>
<td></td>
<td>• 4 (client buffers)</td>
</tr>
<tr>
<td></td>
<td>• 8 (server buffers)</td>
</tr>
<tr>
<td></td>
<td>• 16 (client buffers, in detail)</td>
</tr>
<tr>
<td></td>
<td>• 32 (user jobs awaiting answer)</td>
</tr>
<tr>
<td></td>
<td>• 64 (server answer buffers details)</td>
</tr>
<tr>
<td></td>
<td>• 128 (request global)</td>
</tr>
<tr>
<td></td>
<td>• 256 (server send answer buffer details; not -1)</td>
</tr>
<tr>
<td></td>
<td>• 1024 (dump server received request buffers)</td>
</tr>
<tr>
<td></td>
<td>• 2048 (client trans bitmap)</td>
</tr>
<tr>
<td></td>
<td>• 4096 (client GLO Q)</td>
</tr>
<tr>
<td></td>
<td>• 8192 (request global reference dump, in hex)</td>
</tr>
<tr>
<td></td>
<td>• 65536 (ECP blocks downloaded to clients)</td>
</tr>
<tr>
<td></td>
<td>• 131072 (client released request buffer details; not -1)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| \(-R[value]\) | 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=invalidate 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-V[pid]</td>
<td>Displays variables that are part of the process memory structures; of limited value unless you have access to the source code. <strong>Note:</strong> Windows only. Run from the directory that contains the pid.dmp file.</td>
</tr>
<tr>
<td>-W</td>
<td>Performs the same function as the <code>Backup.General.ExternalThaw()</code> classmethod, and may be used to resume the write daemon after <code>Backup.General.ExternalFreeze()</code> has been called in cases when a new InterSystems IRIS session cannot be started. (See External Backup in the “Backup and Restore” chapter of the <em>Data Integrity Guide</em> 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.</td>
</tr>
<tr>
<td>-X[0/1]</td>
<td>Displays the contents of the device translation table. It is organized by device number and shows both the numeric and plaintext class identifiers.</td>
</tr>
</tbody>
</table>

### 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 the “Using the InterSystems Diagnostic Report” chapter in this guide.

**Note:** The Diagnostic Report task cannot be run on a hung system; if your system is hung, see IRISHung Script in this appendix.
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:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Script name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>IRISHung.cmd</td>
</tr>
<tr>
<td>UNIX®/Linux</td>
<td>IRISHung.sh</td>
</tr>
</tbody>
</table>

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 -q1 -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 the “Monitoring Performance Using ^SystemPerformance” chapter in this guide.