Technical Reference Guide v4.5

Detailed technical information for VARs, OEMs and System Integrators

Jaguar SX

ByteSphere

Revision 4.5.0662

For more information about JAGUAR SX and BYTESPHERE software products, please visit our Web site at www.bytesphere.com or contact us at:

ByteSphere LLC
260 Franklin Street, 11th Floor
Boston, MA 02110, USA
617-475-5209
support@oidview.com
www.oidview.com

JAGUAR SX

Contents

Introduction 6
- Jaguar SX 6
- Jaguar SXME 6
- Supported Browsers 7
- Supported Platforms 7
- What is Jaguar? 6
- Why use Jaguar? 7

Licensing 8

Certification 10

Support 10

System Overview 11
- Jaguar SX External Components 11
- Jaguar SXME Internal Engine Components 11
- Supported Monitoring Technologies and Protocols . . 12

Intellectual Property 13
- 3rd Party IP 14
- ByteSphere IP 13

System Requirements 15
- Licensing Requirements 15
- Software Requirements 15

Installation 16
- Generic Install - OSX 16
- Generic install via Fully automated applet 16
- Generic Install - Windows Platforms 16
- Headless Install - Unix Platforms (OSX, Linux, Solaris) 16

Jaguar Setup and Configuration 17
- Checking the Version and System State 18
- Database Rollups 19
- Database Storage and Data Output 18
- Database Web Interface 19
- Installing the license file 17
- Jaguar Web Interface 20
- JVM Configuration 17
- Polling Configuration 21
- Polling Frequency 20
- Polling Performance 21
- Starting and Stopping the Monitoring Service . . . 17

Custom Poller schedules 23

Suggested Setup Configurations 24
- Integrated Data Collector (Service Providers) 25
- MSP or Distributed Enterprise 24
- OEM Data Collector (3rd Party Vendors) 25
- Stand-alone NMS 24

System Properties 26
- engineconfig.properties file 26

API 50
- API Operations List 52
- API Response Codes 51
- Common API examples 67

Monitor Configuration 69
- AccessConfig 70
- AgentConfig 70
- ConfigUpdate File 69

MonitorConfig 70
MonitorConfig sample 71
Monitor section attributes 71
Monitor Types 72
Monitor Exceptions 73
Monitor Objects 73
Monitor Relations. 73
Monitor Type system files. 72
Monitor Definition Files 74
Built-in Functions. 78
Custom Functions. 80
Declare section. 75
Expressions Section 77
Init section 74
Query Section 75
Reserved Keywords 78
Discover Files 81
Filename format 81
File structure 82
Extending the Engine using Code 87
Extendable Components 87
Get the SDK 87
Database. 88
Administration. 88
Automated Rollups 88
Configuration Tables. 90
Data Tables 89
Performance 88
Supported Databases 88
Exception Engine 93
Exception Parameters 93
Triggering an Exception 93
Notifier 95
Duplicate Events 95
Notifier Events 95
Notifier Users 95
Report Engine 96
PDF Reports 96
Report Arguments via API 97
Report Results 100
Report Types. 96
Running and Retrieving Reports 97
Glossary 102

Introduction

What is Jaguar?

Jaguar software is the fruit of more than 6 years of intensive development, and is –we believe – the most advanced system of its kind for IT Monitoring and Management of a wide range of network sizes and configurations. Jaguar currently comes in two formats, a stand alone engine (SXME) and a fully integrated NMS product including an embedded database and WEB UI (SX).

Jaguar SXME

SXME is an ultra-**S**calable and e**X**tensible **M**onitoring **E**ngine (SXME). With the most advanced, fastest and most scalable automated data collection software framework, SXME can be used by itself as a fully stand-alone system (see SX), or integrated with any 3rd party NMS, EMS, BI, reporting or dashboard system to provide efficient data collection, sorting and delivery to your data and reporting sources. Data sampling can be done with any of a variety of protocols and polling frequency can go down to 100 millisecond intervals. Data is analyzed in REALTIME and actions can be performed based on pre-defined criteria. Information and data can be displayed, automatically exported or shared with 3d party applications for MTBF reporting, capacity planning, and other functions limited only by your needs and imagination. *SXME's customers are primarily Hardware and Software companies that want to OEM or integrate with a monitoring or reporting solution.*

Jaguar SX

SX is a fully-featured Network Monitoring, Alerting, and Reporting Solution, that sits on top of the SXME platform. Event Management is performed by normalizing different types of events and displaying them in a "normalized" alert view. Hosts can be viewed in terms of groups, subnets, or monitored capabilities. Historical and Realtime reports can be run on any groups of items, monitors or hosts over any period of time. Dashboards can be created with a couple clicks, and different types of information including charts, grids, and maps can easily be added or removed. *SX customers are resellers and end-users.*

Why use Jaguar?

For the end-user, Jaguar acts as an affordable monitoring and reporting system supporting both Fault and Performance Management. It can rival the effectiveness and even surpasses the performance (with respect to the polling of performance data), of the largest, most expensive management frameworks.

Integrators, MSPs and VARs can use Jaguar as an add-on to an existing NMS or as one or more remote pollers and/or data-collectors in the end-user or customers's environment.

For OEM use, Jaguar can save engineering and QA teams months and sometimes even years of development and test time... as integration is fast and extension is fairly painless. Using this guide, an engineering team can readily customize and extend monitoring and reporting metrics to meet specific requirements using either meta-data or code (our technical staff can help, too, if necessary).

Supported Platforms

Jaguar is a high-performance, low memory footprint, Java based monitoring solution that runs on just about any platform that supports a JVM (e.g. Windows, Linux variants, MacOSX (intel platforms), Solaris Sparc 64bit), even embedded systems. A minimum of the Java 6u1 JRE is required in order to run. In most circumstances, 64-bit machines should always run the 64-bit version of the JRE. This may require an extra download and installation of Java. See http://java.com/en/download/manual.jsp

Supported Browsers

The WEB-UI interface is based on a HTML5 / Javascript engine using a lot of AJAX and runs very well in most modern browsers (e.g. Chrome 22+, Firefox 16+, Safari 6+, Opera 12+). Functionality in Microsoft Internet Explorer is limited to IE9 and above (we recommend 10+), and will not work correctly in compatibility mode. Internet Explorer IE8 or below are not supported.

Licensing

Jaguar is licensed on volume of elements monitored "monitors", in increments of "hosts". The licensing model is essentially a derivative of "device-based" licensing, except we take into account the size of the device as well. We automatically figure this out for you, and can easily tell you with a simple command how many licenses are being used and how many licenses each host uses as well (for more information please see the API chapter in this document, and refer to the command **system.license.usage**). This greatly simplifies the licensing model for the end-user and resellers alike.

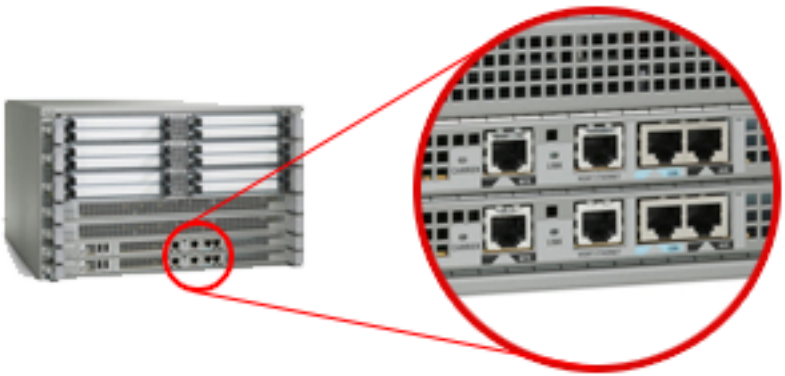
What is a Host?

A **host** is any agent/machine/device with an IP address and less than 100 monitored elements. The idea is that average sized devices will each will be counted as a single host. If a particular device is very large (with respect to the number of elements being monitored on it), then it will subsequently be counted as more than 1 host. The latter case is very rare, but can happen. For example, the majority of devices are discovered with an average of 30 monitored elements (please see the definition of "Monitor", below). If a host supports more than 100 monitored elements (3x the average), then each subsequent 100 monitored elements that the device supports counts as another "host". E.g.: A host supporting 1000 monitored elements will count as 10 licensed "hosts".

1 host 2 hosts 5 hosts 10 hosts

What is a Monitored Element?

A **monitor** is an element that was discovered on a host and represents a group of statistics that will be polled for data each poll period. A monitor can support as many as 128 polled statistics and as few as 1 (the average is 20). Examples of monitored elements are ethernet ports, CPUs, hard-disks, fans, sensors, power-supplies, etc.



Each ethernet port is a monitored element.

In the example of the Ethernet Port as a monitored element, that particular base monitor type ('**interface**') supports more than 30 statistics, including but not limited to, Bytes In, Bytes Out, Utilization (In/Out/Total), Bits, Discards, Errors, Packets, Unicast Packets, Multicast Packets, Broadcast Packets, Latency, Availability, OperStatus, AdminStatus, etc. Derivatives of that base type (such as Cisco Lans or Wireless Lans), could add another 20-30 statistics easily.

The more things Jaguar monitors, the more licenses are needed. The Jaguar Discover process reaches out to each devices and creates monitored elements for everything that the device supports. To decrease the number of monitors that count towards a host license, simply "Disable" that monitor using the UI or API (see the User Guide or the section in this document on API). Conversely, to purchase additional host licenses for more monitoring capability and data collection, please contact our sales department by calling 617-475-5209 or using the form: http://www.oidview.com/contact.html

Certification

Jaguar supports a large number of general device models and hundreds of vendors 'out of the box'. New monitoring support and device models can be supplied by our certification team or implemented by your team. To find out more about how to extend the system using just META-DATA and XML files, please see the sections on **Monitor Types**, **Monitor Defintions**, and **Discover Files**.

In order for our team to add support for a new device or technology, we will need a certification request submitted, which will include a snapshot of the device output (e.g. a mibwalk, traplog, or syslog, etc.), as well as any relevant technical documentation (i.e. ASN.1 MIBs, reference guides, etc.). Your request will be acknowledged within 24 hours, but please allow up to 30 days for our team to complete your request and get you a patch for your newly supported device.

To submit a certification request, please see our page online: http://www.oidview.com/certification_request.html

Support

Our support staff is here to help. Our team can provide answers to most questions by using web-based methods (email and Instant Messenger). If you need additional priority support services (like phone, on-site, after-hours, or development services), please contact us and we will give you a quote by calling 617-475-5209 or using the form: http://www.oidview.com/contact.html

System Overview

Jaguar SX External Components

There are 3 major external software components that comprise the entire Jaguar SX solution platform.

Console UI	Provides native system console application interface
Jaguar SXME	Provides monitoring functionality, runs as a background service
WEB UI	Provides web-based interface

All three of these components can run on any platform and come with separate installers.

Jaguar SXME Internal Engine Components

JaguarSXME is comprised of several "engines" that make up the overall monitoring and reporting system. Not all engines have to be running at the same time, and most can be disabled/enabled by using the properties configuration file, discussed later in this document. The following core engines are run by default on startup:

AnalysisEngine	The module that analyzes and correlates collected data in realtime
ApiEngine	The API module that provides direct communication
DatabaseEngine	The database module that provides local storage access
DiscoverEngine	The module that interrogates the network for inventory to poll
DispatchEngine	The module that dispatches scheduled jobs, including polling
EventEngine	The module that processes event-based information
MonitorEngine	The core engine that launches all other engines
NotifierEngine	The module that processes severity based events and notifies users
OutputEngine	The module that processes and outputs data
PollerEngine	The module that polls statistical and configuration information
ReportEngine	The module that generates report data
SyslogEngine	The module (based on EventEngine) that processes Syslog Events
TrapEngine	The module (based on EventEngine) that processes SNMP Traps
WebEngine	The module provides a WEB UI (based on Jetty)

Supported Monitoring Technologies and Protocols

The core monitoring technologies include:

Events	SNMP Traps, Syslog Events
Ping	ICMP, HTTP, TCP Ports
Query	SQL, WMI (wip), XML
Statistics	SNMP, SNMPv2, SNMPv3, IPMI (wip), AMF

The engine can be extended by using META-data or custom code to extend and enhance the supported monitoring capabilities, for just about any situation. Please see the certification and extending code sections.

Intellectual Property

ByteSphere IP

The core Jaguar MonitorEngine technology has been awarded 1 patent from the USPTO and there is another patent pending. All of our core IP is written by U.S. based engineers and is not outsourced. ByteSphere also has two additional patents awarded that are related to MIB databases and repositories, and MIB detection and acquisition.

There are many pieces of technology that stand out in our Monitoring Engine, making it stand out from the rest. Following are just some of the highlights...

Adaptive Polling

Our patented Adaptive-Polling technology allows the engine to automatically adjust for MIB or data changes in the agent, without getting bad polls or losing data (like most other systems).

Realtime Polling

Our poller can poll at nearly any polling frequency, all the way down to every 100ms if desired. Our realtime, on demand poller allows for timely 3rd party data acquisition or realtime updating charts.

Realtime Data Analysis

Our REALTIME data analysis technology in the exceptions engine not only allows for deep introspection and correlation of data based on polled or calculated values, but does so in memory at poll time, without having to query a database or schedule an hourly or daily job.

Non-Blocking Ultra-Scalable Architecture

Our patent-pending architecture enables the system to not only poll data at incredibly fast speeds, use a (comparatively) small footprint, but can scale to incredibly large numbers of polled devices on a single server and eliminates nearly every source of I/O bottleneck under the most strenuous situations.

Extensible and Customizable

Our highly extensible device model allows users (using XML files only), to define any set of statistics or relate anything in a variety of ways, with multiple levels of hierarchy. (e.g. data aggregation is handled automatically in realtime by the Analysis Engine - no special code has to be written).

3rd Party IP

The Jaguar MonitorEngine uses a number of commonly available, open-source frameworks in the form of "Jars", libraries that are separate from the core code, but referenced when necessary. Most of these libraries use the LGPL, MIT, or Apache license. For an up to date, full list of open-source and 3rd party libraries used, please see the file disitributed with the core engine install `(e.g.: /monitorengine/lib/thirdparty_license_information.txt)`

System Requirements

Software Requirements

Jaguar will run on nearly any machine running a Windows, OSX, Linux, or Solaris operating system, with a minimum JRE 1.6 installed. No other software is needed to run.

Hardware Requirements

Jaguar can run on a virtual machine or a physical one. Memory requirements vary, as it will perform OK in a JVM with a very small amount of RAM (i.e. 32M), but will take as much as it needs as the monitoring requirements grow. On the average, it uses between 200-300M for monitoring 100 devices. If monitoring up to 100 hosts, it can run either on a shared machine or a VM. The general rule is to have 512-1024M extra for the engine to use, and at least 2 CPU cores. Following are some minimum memory and CPU requirement guidelines for different configurations:

CONFIG SIZE	REQUIREMENT GUIDELINES
10,000 monitors (100 hosts)	1G RAM, 1 CPU, 2 Cores, 50G disk (2G database)
100,000 monitors (1,000 hosts)	8G RAM, 2 CPUs 2 Cores each, 500G disk (200G database)
1,000,000 monitors (10,000 hosts)	16G RAM, 2 CPUs 4 Cores each, 8T SAN Array (2T database)

Licensing Requirements

Jaguar will run in restricted mode for 1 host and up to 100 monitors. When a license file is placed into the installed directory, it will automatically read it and license itself. Multiple licenses can be placed into the directory, as long they all start with the text **license.jaguar.key** (e.g. *license.jaguar.key.005056091D0A.monitor*). This helps when there are multi-homed NICs or if a machine dynamically attaches via VPN.

Installation

Once running, the installer is self-explanatory, but it may be a challenge either getting it to run or figuring out which one to use. Here are some tips to help get things moving.

Generic install via Fully automated applet

There is a web-based installer (signed) applet online that should cover most installation scenarios, operating systems and environments. Unless encountering a problem or directed by support, this installation method should be used. Please get the most current applet installer link from support.

Generic Install - Windows Platforms

There are 32-bit and 64-bit installers for Windows platforms. Pick the installer that matches your operating system, and simply double-click on the installer. Make sure you are running it as Administrator. Follow the prompts once loaded.

Generic Install - OSX

Download and launch the DMG as administrator. Follow the prompts once loaded.

Headless Install - Unix Platforms (OSX, Linux, Solaris)

At times it will be necessary to install the engine on a unix machine from a command line. After you download the installer, follow the steps:

1. Change the SH script to executable:

```
chmod 755 monitorengine_unix_45.sh
```

2. Execute the script as root or using sudo, using the -c parameter:

```
sudo ./monitorengine_unix_45.sh -c
```

Jaguar Setup and Configuration

This chapter discusses the more common configuration aspects of the engine after the initial installation. The properties mentioned here are all editable in the **engineconfig.properties** file, and explained in the System Properties section of this document.

JVM Configuration

The Java Virtual Machine must be configured correctly in order to be able to monitor large numbers of devices. The installer puts a file named **monitorengine_svc.vmoptions** into the monitorengine directory, with some basic defaults. This file must be edited in order to setup the JVM properly according to your needs. You can specify the standard JVM memory and garbage collection arguments, each on it's own line. Example:
-Xmx8196m

Starting and Stopping the Monitoring Service

After the installation, the MonitorEngine Service will have been started automatically. You may need to stop / start the service at some point manually in the future.

Windows

Go to Windows Services (or run services.msc), and find the "OiDViEW MonitorEngine Service". Stop and Start as needed.

Linux and OSX

In the installed directory, find the monitorengine_svc file.
To run, *sudo ./monitorengine_svc -start*
To stop, *sudo ./monitorengine_svc -stop*

Installing the license file

Simply copy the license.key file provided by sales or support into the installed directory. Then, run the command: *./xmlApiClient -o system.license.load*

Checking the Version and System State

From a command prompt, run the following commands:
xmlApiClient -o system.alive
xmlApiClient -o system.version

Database Storage and Data Output

Will Jaguar be storing data in a database? If so, by default all data (configuration, historical statistics, and event data), will be stored internally using the high-performance embedded database (using H2 database technology). Otherwise, collected data can be stored in an external database like MySQL, SQL-SERVER, or Oracle. The database can be local or remote to the polling engine. Finally, one can choose to NOT store data in a database, but to simply output it as a CSV, XML, or SQL file. These files are output to the `/monitorengine/output` directory.

Properties settings for the databases are as follows:

DATABASE	PROPERTY SETTING
No Database	db_type=none
H2 Database (default)	db_type=h2
MySQL Database	db_type=mysql
SQLServer Database	db_type=sqlserver

The other database settings (excluding type), must be changed according to your particular situation (for example, SQLServer, the db_name is usually the instance name). Following are the default database settings on install:

```
db_host=localhost
db_name=bytesphere
db_pass=bytesphere!
db_user=bytesphere
```

For all these database settings, the output-engine must be configured to use SQL Engine:
output_engine_fqcn=com.bytesphere.outputengine.SqlEngine

To output all statistics and configuration data to flat files, use the following settings:
```
output_engine_fqcn=com.bytesphere.outputengine.CsvEngine
output_engine_fqcn=com.bytesphere.outputengine.XmlEngine
output_engine_fqcn=com.bytesphere.outputengine.SqlEngine
```

Database Web Interface

If using the default H2 database, there is an embedded WEB UI for general SQL queries and table administation. To access this database administration interface, use a web browser and point it to https://localhost:8082. The login screen will ask for a JDBC url and a username and password. The defaults are:

jdbc url: jdbc:h2:file:data/bytesphere
username: bytesphere
password: bytesphere!

Database Rollups

Jaguar polls data and if using a database, will be storing LOTS of it. The system is configured to automatically roll up data based on hourly and daily tables. There is 1 table for each hour and poll period. These tables store RAW data. Then, there is 1 table created for each day and poll period, and the RAW data is averaged and placed into this table (1 datapoint for each hour). Then, there is a monthly table, where all daily data is again rolled up and averaged and placed into the monthly table. This allows us to show you quick snapshots of data over large periods of time for large numbers of datapoints. There are a number of settings for rollups that control this behavior (defaults below):

data_daily_days_to_keep=60	Number of months of rolled up data to store
data_hourly_days_to_keep=30	Number of days of rolled up data to store
data_raw_days_to_keep=7	Number of days of RAW data to keep
data_rollup_enabled=true	(should be true unless not using DB)

Jaguar Web Interface

Jaguar ships with a high-performance embedded WEB UI for reports and general system and user administration. The embedded URL for access is http://localhost:8080. You will be asked to create an admin login the first time it is run. The default username and password used is admin/admin.

There is a configuration file for the web interface that can be found in the includes directory (includes/config.php). This file can be modified to change the database settings, timeouts, email settings, and a number of other parameters. If you make changes to this file, then you must back it up and copy it back into place when updating the web UI.

The core underlying WEB technology is a Jetty Servlet container which will automatically expand a WAR file from the /webapps directory. It will also update itself automatically with the latest bytesphere WAR files, as they are released, as long as the engine has access to the internet. You can create a customized UI by simply editing the PHP files that come inside the WAR. You can also create your own WAR files, and deploy them to remote engines by copying them to the webapps/web.war file. Once present, they will automatically be expanded and run by the system.

You can also choose to use a different servlet container (i.e. JBOSS, Tomcat, etc.), on a different machine, on which to deploy the WAR. There may be some customization for your specific instance, if you need assistance please contact us @ **617-475-5209** or support@oidview.com.

Polling Frequency

There are three different polling frequencies in the system, each are set in milliseconds:

FREQUENCY	DESCRIPTION
Default Polling Frequency	5 minutes is the default (300000 ms)
Realtime Polling Frequency	Anything under 30 seconds is considered “realtime”
Scheduled Polling Frequency	Groups of monitors can be scheduled to poll at certain times

Each monitor can have one of these three polling settings, either individually or all at the same time. By default Jaguar polls for historical data every 5 minutes. If you need to increase that, then you can change the default polling frequency (for all monitors in the system), by changing the global **poller_period** property, which is set in milliseconds. Realtime polling frequency is changed per "monitor", and can be set via the API or UIs. Scheduled frequencies can be setup through the UI or through the **engineconfig.properties** file. More information can be found in the properties section of this guide.

Polling Configuration

The polling configuration is the list of things the engine is going to poll each default poll period. This is supported in both a text file (XML) format and in the database. If the database is configured, the configuration will be stored there as well as in memory. The default XML filename is monitor.cfg.xml and is located in the root installed directory `(i.e. /opt/monitorengine/monitor.cfg.xml)`. This file is read on startup, and can be created/updated by the system when using the API command **system.config.read**. For more information on this, please see the ConfigUpdate section of this document.

Polling Performance

Jaguar can be configured to poll a huge number of monitors and devices, on a single server, depending on the network topology and specifications of the management machine. Our performance testing was conducted in our own private performance lab, using in-house SNMP simulators and just the SNMP poller-engine blade, polling hosts and ethernet ports over a simple ethernet switch (1 hop). Machines running simulators were DELL SC1435 blade servers, running Centos 5, each with a single Dual-Core 2Ghz CPU, and 8G RAM. Poller machine was a DELL 2900 PowerEdge server, running Centos 5.0 with 2 Dual-Core 2Ghz CPUs, and 16G RAM.

Based on our internal testing, following (see Table PERF-1) are some recommended settings based on network size, polling configuration, and machine requirements. Actual settings may need to change based on network size, topology, devices and agents being polled (speed and capability of agents), and the specifications of the management machine designated as the poller running Jaguar.

Table PERF-1 - Recommended property settings for performance polling.

HOSTS	MONITORS	RAM	CPU	PROPERTY SETTINGS
100	5K	1G	1x2Ghz	Default
1K	50K	4G	2x2Ghz	max_agent_pdus_downshift_enabled=false max_snmp_contexts_per_pool=50 max_snmp_context_threads=20 max_agent_simultaneous_request_pdus=40
10K	100K	8G	4x2Ghz	max_agent_pdus_downshift_enabled=false max_snmp_contexts_per_pool=500 max_snmp_context_threads=10 max_agent_simultaneous_request_pdus=10
100K	100K	16G	4x2Ghz	max_agent_pdus_downshift_enabled=false max_snmp_contexts_per_pool=2500 max_snmp_context_threads=2 max_agent_simultaneous_request_pdus=2
50	1M	16G	4x2Ghz	max_agent_pdus_downshift_enabled=false max_snmp_contexts_per_pool=10 max_snmp_context_threads=30 max_agent_simultaneous_request_pdus=60 snmp_pdu_retry_inter- vals=100,150,250,500,1000,2000

Custom Poller schedules

Pollers run on a DEFAULT schedule and can be activated in REALTIME. Pollers also can be scheduled to startup for a specific group of monitors at a specific time in the day (CUSTOM). By default, all monitor items when created are assigned to group 0, and are automatically assigned to the normal poll (30sec+) or realtime poll (1sec+) schedule by their assigned poll frequency. To see the details on how to configure them, please see the Properties section of the document. Here are some examples:

The following poller schedule is called "subsecond_am"
This will launch a 2 minute 250ms interval poller process at just before 9am every day
This will only process monitors that have been defined with a schedule group of 1

```
custom_scheduler_1_name=subsecond_am
custom_scheduler_1_group=1
custom_scheduler_1_params=,,8,59,0,0,250,480
```

The following poller schedule is called "subsecond_pm"
This will launch a 2 minute 250ms interval poller process at just before 5pm every day
This will only process monitors that have been defined with a schedule group of 1

```
custom_scheduler_2_name=subsecond_pm
custom_scheduler_2_group=1
custom_scheduler_2_params=,,16,59,0,0,250,480
```

The following poller schedule is called "hourly_ping"
This will launch a single ping every hour. This will only process monitors that have been defined with a schedule group of 3

```
custom_scheduler_3_name=hourly_ping
custom_scheduler_3_group=3
custom_scheduler_3_params=,,,,,,3600000,0
```

Suggested Setup Configurations

Stand-alone NMS

To setup Jaguar as a stand-alone Network Management System, pick a computer that is central to your network infrastructure (or at least has complete access to your network devices). It will be installed on this computer. By default, the WEB-UI and database will both be embedded. We recommend, for simplicity, keeping it this way.

MSP or Distributed Enterprise

Jaguar can be very useful for MSPs or very large distributed enterprises, as it can act as a central or remote poller, send data to a 3rd party NMS, or just watch KPIs and alert via email or trap when thresholds are crossed.

Single Local Poller

In this situation, it is very much like a stand-alone NMS. The MSP admin will have to access it behind the firewall (either using LogMeIn, GoToAssist, VPN, or some other remoting technology). If Jaguar is storing data, it will be stored internally using the high-performance embedded database, and no settings or changes really need to be made. The MSP can remote into the customer network, and then use the embedded WEB UI to log in and perform administration. The notification email settings can be set so the MSP is aware of issues occurring on the network.

Multiple Remote Pollers

Multiple engines can be installed geographically dispersed, in an unlimited number of locations. Each engine needs to be assigned a different engine_id (set in the engineconfig.properties file). If each remote poller is for another customer, each engine also needs to have a different organization_id (set in the engineconfig.properties file). The database can then be configured for each engine (also in the engineconfig.properties file) to send collected data to a remote, central database (which needs to be accessible from all the remote pollers). This database could be hosted online or be accessed through an always-on VPN which is installed on the remote poller behind the customer's firewall. The

administrative WEB UI may be accessed on a single engine or installed in a central location (on the internet), using a Tomcat, JBOSS, or other type of web servlet container instance. It must be configured as well to look at the central database. To do this, the config.php file in the web.war distribution needs to be modified. More on this will be explained in the Web UI Configuration section.

Integrated Data Collector (Service Providers)

For large service providers using a number of reporting and data collection frameworks, Jaguar can be an ideal addition to the arsenal and help you to quickly get additional monitoring and reporting online. It can bubble up a variety information and data to large frameworks like CA eHealth and Spectrum, IBM Tivoli, BMC, EMC Smarts, Telcordia Service Director, etc. The list goes on. Please contact us @ **617-475-5209** or support@oidview.com for assistance with integration points and APIs.

OEM Data Collector (3rd Party Vendors)

For technology vendors looking to expand their monitoring capabilities, scalability, or add an integrated NMS and/or EMS into their product portfolio, Jaguar's white-label OEM Engine solution can easily fill the void and become part of your branded solution. Use all or just some of the plug and play components or engines. Please contact us @ **617-475-5209** or support@oidview.com for assistance with integration points and APIs.

System Properties

engineconfig.properties file

The ***engineconfig.properties*** file provides the main engine configuration and helps to control how the MonitorEngine behaves. This is a simple text file using name=value pairs, and is read fully by the MonitorEngine on startup. To list all the properties at runtime, please see the API section of this document, and refer to the comand **system.properties.list**.

DO NOT EDIT THIS FILE WHEN THE ENGINE IS RUNNING. Editing the file at runtime will not cause the engine to re-read the properties. In addition, properties that are set via the API at runtime will be written out to the file, potentially overwriting any changes made manually while the system is running. To make changes during runtime, please see the API section of this document, and refer to the command **system.properties.set.**

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
allow_frequency_update	Setting this allows monitor polling frequency to be updated by the discover/merge process. By default this is 'false', as one normally does not want discover to override a user's polling frequency setting. But, there may be cases where the global polling frequency has been changed, and it would be easiest to just have everything that is discovered use the new polling frequency. In this case, setting this to 'true' will simply set all the discovered (and updated) monitors to the new polling frequency. `allow_frequency_update=false`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
custom_scheduler_{X}_group	The group of monitors to poll for a custom poller schedule. This is typically the group_id availabe in the database. Monitor Items that are assigned to that group will be put on the custom poll schedule. Those same items will also be polled by the default schedule unless the monitor item default poll frequency is set to -1. Valid custom group schedule #s are greater than 1. `custom_scheduler_1_group=1`
custom_scheduler_{X}_name	The name for a custom poller schedule. `custom_scheduler_1_name=subsecond_am`
custom_scheduler_{X}_params	Parameters specify when the scheduler will start and how often it will fire. PARAMS = DATE_FIELD,DATE_VALUE,TIME_HOUR,TIME_MINUTE,TIME_SECOND,TIME_MILLISECOND,INTERVAL,REPEAT DATE_FIELD is a field like HOUR_OF_DAY, DAY_OF_WEEK, DAY_OF_MONTH, MONTH_OF_YEAR, etc. DATE_VALUE would be the value of the date field TIME_HOUR, TIME_MINUTE, TIME_SECOND, TIME_MILLISECOND are just that. Leave these null unless you are specifically setting those fields (i.e. 0,0,0,0 means start at midnight). INTERVAL is the amount of time in milliseconds between scheduled executions REPEAT is the number of times to repeat this scheduled execution - 0 is forever `custom_scheduler_1_params=,,8,59,0,0,250,480`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
custom_scheduler_count	Turns On / Off custom poller schedules. Setting this to zero (even if they are defined) shuts off custom scheduling completely. `custom_scheduler_count=0`
data_daily_days_to_keep	Data rollup parameter specifying how many days of DAILY averaged data to keep in the database (1 day averaged data from HOURLY). `data_raw_days_to_keep=180`
data_hourly_days_to_keep	Data rollup parameter specifying how many days of HOURLY polled data to keep in the database (60 minute averaged data from RAW). `data_hourly_days_to_keep=30`
data_keep_raw_bulkstats_files	Delete all bulkstats files created (i.e. MySQL bulk stats). Defaults to 'true'. If 'false', then all bulkstats files will be left on the disk and not cleaned up (good for debugging or if an external process is downloading those files for insertion to another system). `data_keep_raw_bulkstats_files=false`
data_raw_days_to_keep	Data rollup parameter specifying how many days of RAW polled data to keep in the database (usually 5 minute data, depending on default poller period configuration). `data_raw_days_to_keep=7`
data_raw_hours_to_keep	Data rollup parameter specifying how many hours of RAW polled data to keep in the database (usually 5 minute data, depending on default poller period configuration). Only use raw hours if you are NOT using raw days. `data_raw_hours_to_keep=0`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
data_rollup_enabled	Data rollup parameter specifying whether or not the database rollups are enabled (true by default). `data_rollup_enabled=true`
data_rollup_include_min_max	Include MIN/MAX in rollups `data_rollup_include_min_max=true`
db_auto_tuning_enabled	Allow automatic tuning of db based on config size `db_auto_tuning_enabled=false`
db_debug_logging_enabled	Database sql connection and statement logging `db_debug_logging_enabled=false`
db_host	Database parameter specifying the IP address of the host running the database. `db_host=localhost`
db_large_config_threshold	The number of monitors at which point to cut over to bulk stats loading, auto tuning, etc. `db_large_config_threshold=25000`
db_name	Database parameter specifying name of the database. `db_name=bytesphere`
db_pass	Database parameter specifying the login password `db_pass=bytesphere!`

db_type	Database initialization parameter specifying the type of database. Valid values are:
	`db_type=none`
	`db_type=h2`
	`db_type=mysql`
	`db_type=sqlserver`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
db_user	Database parameter specifying the login name `db_user=bytesphere`
discover_allow_address_updates	If IP addresses change in the network and device is already being monitored using a different IP address, setting this to 'true' will update the system to use the new address. Set this to 'false' to disable update behavior. Default is 'false' due to the common scenario of multiple IPs on a single device. The first IP address a device is discovered with will continue to be the **management address**, unless the device is no longer available at the original management address. Then, updates will happen automatically regardless of the setting here. `discover_allow_address_updates=false`
discover_host_lookup_file	The file to use for hostname lookup PRIOR to using DNS lookups (if enabled). Leave blank to disable this extra lookup mechanism. `discover_host_lookup_file=`
discover_polling_disabled	Configures discovery so that polling for that particular monitor will be disabled when it is discovered. By default, all monitors are automatically enabled for polling when initially discovered and entered into the system. `discover_polling_disabled=false`
discover_scheduler_main_enabled	Specifies whether scheduled discovery is enabled. Also see discover_scheduler_main_params. `discover_scheduler_main_enabled=false`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES

discover_scheduler_main_params	If auto-discover is enabled, this value specifies the scheduler format. The format follows the same format as the Poller Schedules.
	
	PARAMS = DATE_FIELD,DATE_VALUE,TIME_HOUR,TIME_MINUTE,TIME_SECOND,TIME_MILLISECOND,INTERVAL,REPEAT
	
	Interval values are in milliseconds. The example below shows auto-discovery being launched once a day at 7:30PM:
	
	`discover_scheduler_main_params=`
	`,,19,30,0,0,86400000,0`

discover_seed_type	The discover seed type specifies how scheduled auto-discovery will work. disabled - do not do auto-discovery auto - try to discover as much as possible without a seed range - discover things in a specified range (e.g. 192.168.1-3.1-254) seedfile - discover things specified in a separate seed file config - only (re)discover things in the configuration `discover_seed_type=range`

discover_seed_value	The discover seed value (only valid with ‘range’ and ‘seedfile’. specifies the parameter for discovery
	
	Example for a range value:
	
	`discover_seed_value=192.168.1.1-254`
	
	Example for a seedfile value:
	
	`discover_seed_value=/opt/discoverseeds.txt`

discover_type_location	The location of discover type definition files `discover_type_location=discover`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
dynamic_poller_tuning_enabled	Automatically tries to tune the poller based on configuration size and network response times. Better to do this manually, it can be a real resource hog. `dynamic_poller_tuning_enabled=false`
engine_id	The unique ID of the engine, default is 0. Each engine must have it's own ID set to a unique number if they are writing to the same database. This is a user assigned number (the admin must do it at install time), it is not automatically assigned. `engine_id=0`
engine_primary_ip	The IP Address of the primary, active monitor engine `engine_primary_ip=192.168.1.15`

engine_status	The monitoring status for the engine (used with failover). **active** means it will monitor and act as primary. **passive** means it will be in backup mode, polling the primary.
	`engine_status=active`
	`engine_status=passive`

event_autoclear_time	The time that an event will take to automatically clear. If the event has been in the system after this period of time, it will clear by itself. Default is 1 day. `event_autoclear_time=86400000`
event_autodelete_time	The time that an event will take to automatically delete. If the event has been in the system after this period of time, it will delete itself. Default is 1 week. `event_autoclear_time=604800000`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
event_dbstorage_disabled	Allows for the disabling of event storage in the database. Events (traps, syslogs, etc.) will be processed but not stored in the database. `event_dbstorage_disabled=false`
event_maint_frequency	Sets the maintenance frequency for the Event Engine, in minutes. The default is 60. `event_maint_frequency=60`
event_max_storage_count	Sets the max event count that will be stored (in memory or the db), by the event engine. If the count gets higher than this, maintenance will be triggered and excess events (oldest first), will be removed. `event_max_storage_count=100000`
exception_collapse_agent_threshold_count	The number of identical exceptions for different monitors on the same agent that will trigger collapse of the events into 1 single event. `exception_collapse_agent_threshold_count=20`
h2.bindAddress	Sets the bind address for the H2 embedded database. Leaving it empty will force the system to search for the best physical bind address (not a VPN or virtual address) `h2.bindAddress=192.168.1.15`
hostname_resolution_enabled	Enables / Disables DNS lookup for Hostname resolution `hostname_resolution_enabled=true`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
max_agent_pdus_downshift_enabled	If agent timeouts occur, this controls the behavior to decrease the number of maximum pdus being sent out at the same time. For example, if the max is set to 60, and a timeout occurs, if this is set to 'true' then the max will next be set to 30. If timeouts continue to occur, it will again decrease the max value of simultaneous requests, until either it gets to below 10 or timeouts do not occur anymore. `max_agent_pdus_downshift_enabled=false`
max_agent_simultaneous_request_pdus	The maximum number of requests that will be sent out per agent (irregardless of protocol), at any one time. `max_agent_simultaneous_request_pdus=60`
max_engines_per_pool	The maximum number of engines to create per engine pool. The controls all engine pools. Engines are created dynamically as they are needed and retired to the pool after use. `max_engines_per_pool=25`
max_engines_per_pool	The maximum number of engines to create per engine pool. The controls all engine pools. `max_engines_per_pool=25`
max_results_queue_size	The maximum size of the results collector queue. This can be helpful if for some reason the connection to the database is temporarily disconnected, the results will continue to be stored in the collector queue until the connection is re-established. `max_results_queue_size=60`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
max_snmp_context_threads	The maximum number of threads per SNMP context. This controls the number of simultaneous requests that can be made against a single host at the same time. `max_snmp_context_threads=40`
max_snmp_context_threads	The maximum number of threads per SNMP context. This controls the number of simultaneous requests that can be made against a single host at the same time. `max_snmp_context_threads=40`
max_snmp_contexts_per_pool	The maximum number of SNMP contexts to create per context pool. This controls the number of simultaneous hosts that can be queried via SNMP at the same time. `max_snmp_contexts_per_pool=5`
monitor_cfg_write_file_enabled	Controls whether or not the polling configuration will be written out to an XML file after configuration changes. `monitor_cfg_write_file_enabled=true`
monitor_debug_level	This parameter sets the amount of debugging to be output to the log. Level 1 is the least. Level 10 is the most. Default is '5'. `monitor_debug_level=5`
monitor_debug_mode	Default is 'false'. Setting to 'true' will only turn on debugging for monitors that are marked as 'debug' - this way one can follow a single or set of monitors through the system without having to see all the rest of the debug. `monitor_debug_mode=false`
monitor_keys_include_access_id	Include Access ID in the monitor keys (internal) `monitor_keys_include_access_id=false`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
monitor_max_timeouts_{ACCESS}	The maximum number of timeouts allowed for a particular access type before disabling access. {ACCESS} can be any protocol the system supports (e.g. SNMP, IPPORT, IPMI, WMI, etc.). When a timeout occurs, a warning alarm is generated and the poller will wait for the retry interval suggested (see monitor_retry_interval_{ACCESS}) before it tries again. If the engine still cannot get a response from the agent, it doubles the retry interval, and tries again, and so on, until the max_timeouts value is reached. If X successive timeouts occur and the device is still reachable (i.e. it can be pinged), but not queried with this access type, another more severe alarm is generated and then access is disabled for a specified sleep period (see monitor_retry_sleep_{ACCESS}). `monitor_max_timeouts_SNMP=10`
monitor_retry_interval_{ACCESS}	The base retry interval in milliseconds used when an access type times out. The retry interval cannot exceed 120000 (2 minutes). `monitor_retry_interval_SNMP=5000`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
monitor_retry_sleep_{ACCESS}	The amount of time in milliseconds that polling for a timed out agent will be suspended. Max value for "monitor_retry_sleep" that can be specified is 32000000. Min value is -1, which indicates that the agent should not be polled anymore for this access type (until the system is restarted). A value of 0 means that the timeout count just goes back to 0 and the entire process starts over. `monitor_retry_sleep_SNMP=7200000`
monitor_type_location	The location of the monitor type definition files `monitor_type_location=monitor`
monitorcfg_filename	The filename of the monitor cfg file `monitorcfg_filename=monitorcfg.xml`
monitorcfg_reader	The default monitor config reader to use. The monitor config reader is a group of reader classes that read in configuration files. For example, if the reader is **reader_bytesphere_cfg** then it itself must be defined in the properties (see reader_bytesphere_cfg), and then sub-properties based on this property must be defined. For more information see the section on File Reader Associations. `monitorcfg_reader=reader_bytesphere_cfg`
monitorobjs_filename	The filename of the monitor objects file `monitorobjs_filename=system/monitor-objects.xml`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
monitortypes_filename	The filename of the monitor types file `monitortypes_filename=system/monitor-types.xml`
net_snmp_procfix_enabled	If set to 'true' and polling a NET-SNMP agent, if it supports and has a configured PROC FIX setting, it will issue a SET command to the agent to set prErrFix to 1 and commence the fix. For more information on this NET-SNMP feature see: http://www.net-snmp.org/docs/man/snmpd.conf.html#lbAR `net_snmp_procfix_enabled=true`
notifier_audio_url	Tells the Notifier Engine to play an audio file whenever there is an alert. The supported file formats are: "wav", "au" and "aiff". Can be local or a URL. `notifier_audio_url=./my_alert.wav`
notifier_audio_severity_trigger	Triggers the Notifier Audio Playback (if so configured), whenever an alert over a certain severity is generated. `notifier_audio_severity_trigger=critical`
notifier_disabled	Allows for the Notifier Engine to be disabled `notifier_disabled=false`
notifier_email	The default email address to send alerts. To disable email alerts, leave blank `notifier_email=myaddress@mycompany.com`
notifier_event_expire_period	This is the amount of time that the Notifier will wait before expiring an event completely... time is in milliseconds. Specify 0 to never expire events. `notifier_event_expire_period=3600000`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
notifier_event_timeout_period	The default period for which the same event will be ignored by the notifier (milliseconds) `notifier_event_timeout_period=300000`
organization_id	The default organization id for this engine. All monitors and hosts by default will be saved to the organization set here, unless otherwise specified. Default is 0. `organization_id=0`
output_config_with_stats	Determines whether or not polled configuration data should be output with statistics. `output_config_with_stats=false`
output_directory	The directory that output files will be written to `output_directory=./output`
output_engine_force_sqlfile	Forces or suppresses SQL file output. If going directly to DB, then the sql file by default will not be generated because the engine is communicating directly with the DB over a port (i.e. sql files will not be written separately to disk). Set this to 'true' in order to generate it. If running in debug mode, sql files will be generated anyway. If suppression of sql files is desired, even in debug mode, set to 'false'. `output_engine_force_sqlfile=true`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
output_engine_fqcn	The java fully-qualified class name (FQCN) of the output engine to be used by the monitor engine. To output XML: com.bytesphere.outputengine.XmlEngine To output CSV: com.bytesphere.outputengine.CsvEngine To output SQL (and/or go to an SQL database): com.bytesphere.outputengine.SqlEngine `output_engine_fqcn=com.bytesphere.outputengine.SqlEngine`
output_engine_fqcn_high_volume	The java fully-qualified class name (FQCN) of the HIGH VOLUME output engine to be used. `output_engine_fqcn=com.bytesphere.outputengine.MySQLEngine`
output_header_column_names_storage	Controls output for column names in the header of the output files. 'false' will use the object_id for column names (e.g. bytes_in). 'true' (the default) will use the storage_id for column names (e.g. c1) `output_header_column_names_storage=true`
ping_monitor_default_type	The default PING type. Can be 'none', 'auto', 'isreachable', 'icmp', 'tcp_connect', 'tcp_echo', 'udp', 'udp_echo'. The default is 'auto', which usually picks 'icmp' based on the situation. `ping_monitor_default_type=auto`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
ping_monitor_enabled	Enables/disables PING monitoring (‘true’/’false’). A PING does not have to be ICMP. `ping_monitor_enabled=false`
ping_monitor_icmp_enabled	Enables/disables ICMP monitoring (‘true’/’false’). ICMP monitoring means using ICMP as type of monitoring for availability. `ping_monitor_icmp_enabled=true`
ping_monitor_icmp_packet_count	The number of ICMP packets to be sent to determine RTT (rount trip time). It takes the average of the number of packets time. `ping_monitor_icmp_packet_count=3`
ping_monitor_icmp_timeout	The number of milliseconds before an ICMP packet is considered to be timed out. `ping_monitor_icmp_timeout=200`
ping_monitor_mode	Tells the engine whether to PING all the time (‘constant’ - the default), or just at the beginning of each poll (‘poll’). Set to ‘off’ to disable completely. If set to ‘constant’, it will be used to determine host availability and the event engine will be notified when a host goes offline. If set to ‘poll’, then it will only poll those agents that respond to the initial PING. `ping_monitor_mode=constant`
ping_monitor_no_response_count	The number of unsuccessful pings before calling a host unreachable. `ping_monitor_no_response_count=3`
ping_monitor_tcp_connect_ports	The list of ports, comma separated, to use for TCP connect PING monitoring `ping_monitor_tcp_connect_ports=23,80`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
ping_monitor_tcp_connect_time-out	The number of milliseconds for a TCP connection attempt to be consifered a failure during the polling process. `ping_monitor_tcp_connect_timeout=2000`
ping_monitor_tcp_connect_time-out_discover	The number of milliseconds for a TCP connection attempt to be considered a failure during the service discovery process. `ping_monitor_tcp_connect_timeout_discover=1000`
ping_monitor_wait_time	The amount of time in milliseconds to wait between PING jobs. `ping_monitor_wait_time=5000`
poller_bind_addresses	If the poller should use multiple NIC cards or needs to bind to a specific card or IP address, enter those values here. If more than one bind address is specified, the system will bind to them in a round robin fashion. Default, not set. `poller_bind_addresses=10.1.1.96,10.1.1.56`
poller_blade_count	The number of custom polling blades defined in the configuration. Then, this number is used in the rest of the poller_blade_** property settings. `poller_blade_count=2`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES

poller_blade_X_class	The java fully-qualified class name (FQCN) of the poller blade being defined. The pollerengine's blade classloader looks for this FQCN and loads it dynamically at runtime.
	`poller_blade_1_class=com.bytesphere.`
	`pollerblades.PING`
	`poller_blade_2_class=com.bytesphere.`
	`pollerblades.SNMP`

poller_blade_X_type	The access type for this particular poller blade. The X represents the number of the blade definition. The value for the variable is the access-type itself.
	`poller_blade_1_type=PING`
	`poller_blade_2_type=SNMP`

poller_counters_zero_negative_deltas	This prevents aberrant counter behavior from skewing data if certain deltas come back negative. When set to true, those values are simply inserted as zero. `poller_counters_zero_negative_deltas=false`
poller_dispatch_max_monitors	Max number of monitors to dispatch per poller instance `poller_dispatch_max_monitors=100000`
poller_div0_null_values	The way to handle results with division by zero. Setting to false sets the result to 0 (false) Setting to true sets the result to null `poller_div0_null_values=false`
poller_output_normalized_values	Should the expression engine pre-normalize values based on the defined storage_type in the monitor-types.xml file for this object? `poller_output_normalized_values=false`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
poller_period	The default poll period between polls in milliseconds. all monitoritems will be polled at this frequency unless otherwise specified. `poller_period=300000`
poller_period_rae_launch_cutoff	The max length of the poll period before Results Analysis Engine (RAE) is launched. By default it waits up to 90% of the defined poller period. `poller_period_rae_launch_cutoff=270000`
poller_period_realtime_cutoff	Any monitored item with a frequency of LESS than this will automatically be added to the real-time poller queue. `poller_period_realtime_cutoff=30000`
poller_period_realtime_default	The realtime default polling period between polls in milliseconds. `poller_period_realtime_default=10000`
poller_period_realtime_relation_max	The max number of relations allowed to set to realtime polling when setting realtime frequencies for relations automatically. If the relation count is higher than this do not allow realtime for a device (i.e. a router and it's relations). `poller_period_realtime_relations_max=1000`
poller_timeouts_autoreset_interval	Implement autoreset of all agent timeouts in milliseconds. Setting this resets all agents that may have timed out and had polling turned off every {interval} milliseconds. Polling will be attempted again after this interval, globally. `poller_timeouts_autoreset_interval=360000`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
poller_zero_null_values	Substitutes 0 for null values. Default value should be true. This allows for evaluations of complex expressions if certain values are missing and could not be collected `poller_zero_null_values=true`
port_monitoring_enabled	Enables / Disables service port discovery and monitoring. Service monitoring will use TCP connections to talk to well known services to ensure that they are still running and responding to requests. This function does not work well on Windows operating systems, but works fine on Linux. It is disabled by default. `port_monitoring_enabled=false`
reader_bytesphere_cfg	The reader that scans the monitor.cfg.xml file on startup (or on an API call), that populates the polling configuration. To extend the reading capability, put in the custom classname here and make sure the jar is in the classpath. `reader_bytesphere_cfg=com.bytesphere.reader.ByteSphereCfgReader`
reader_bytesphere_cfg_monitor	The reader that scans all the monitor xml files in the monitor directory, on startup (or on an API call), that populates the monitor polling definitions in the engine's memory and db. To extend the reading capability, put in the custom classname here and make sure the jar is in the classpath. `reader_bytesphere_cfg=com.bytesphere.reader.MonitorFileReader`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
reader_bytesphere_cfg_objects	The reader that scans the monitor-objects.xml file on startup (or on an API call), that populates the monitor object definitions in the engine's memory and db. To extend the reading capability, put in the custom classname here and make sure the jar is in the classpath. `reader_bytesphere_cfg=com.bytesphere.reader.ObjectFileReader`
reader_bytesphere_cfg_type	The reader that scans the monitor-types.xml file on startup (or on an API call), that populates the monitor type definitions in the engine's memory and db. To extend the reading capability, put in the custom classname here and make sure the jar is in the classpath. `reader_bytesphere_cfg=com.bytesphere.reader.MonitorTypeFileReader`
smtp_password	The SMTP password to use for logging into the SMTP server `smtp_password=password`
smtp_server	The IP address (or hostname) of the SMTP server to send emails through `smtp_server=192.168.100.45`
smtp_username	The SMTP username setting for the SMTP server `smtp_username=username`
snmp_community_read	The SNMPv1/v2c read community string to use for discovery and/or polling `snmp_community_read=public`
snmp_community_write	The read and write SNMPv1/v2c write community string to use for discovery and/or polling `snmp_community_write=private`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
snmp_discover_allowed_vendors	Allows the SNMP discovery process to filter discovery only to certain allowed, pre-defined vendors by specifying allowed enterprise numbers. This filtering involves querying the system table and looking at sysOid to determine which vendor is being discovered. Default is set to nothing - so all vendors will be discovered. If `snmp_discover_allowed_vendors=9,141`
snmp_discover_ports	The UDP ports to use for SNMP discovery. To specify more than one port, use comma delimeter (e.g.: 161, 5000, 6000). Default is 161. `snmp_discover_ports=161`
snmp_error_suppress_no_such_name	Suppresses SNMP NO_SUCH_NAME errors. `snmp_error_suppress_no_such_name=false`
snmp_force_getbulk	If set to 'true', SNMP discovery and polling will always use GETBULK instead of GETNEXT requests for agents that do not support GETNEXT. Default is 'false'. `snmp_force_getbulk=false`
snmp_max_vars_per_pdu	Defines the maximum number of variables to be put in a PDU. This value has a huge effect on polling performance. If set too high, it can break some SNMP agent implementations or cause a TOO BIG or TOO MANY error, which can affect performance. If set too low, it can also increase the number of requests that have to be sent and processed and also can lower performance. The default is '32'. `snmp_max_vars_per_pdu=32`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
snmp_pdu_retry_intervals	Defines the SNMP PDU retry intervals in milliseconds. e.g. 250,500,1000,2000 defines 4 retries at different timeout intervals. If the PDU times out after 250ms, then it will try again and wait 500ms. If it still times out after 500ms, it will try again at 1000ms, and so forth. `snmp_pdu_retry_intervals=250,500,1000,2000`
syslog_disabled	Enables / Disables the Syslog Engine. If disabled, the engine cannot receive syslog notifications. `syslog_disabled=false`
thread_prio_api	Sets the Thread Priority of the API process. Valid values are 1-10 with 10 being the maximum. `thread_prio_api=5`
thread_prio_discover	Sets the Thread Priority of the discover process. Valid values are 1-10 with 10 being the maximum. `thread_prio_discover=5`
thread_prio_rollup	Sets the Thread Priority of the rollup process. Valid values are 1-10 with 10 being the maximum. `thread_prio_rollup=5`
timezone	The timezone which the engine resides. Default is 0 (GMT). `timezone=0`
trap_host	The host to send SNMP Traps `trap_host=192.168.100.4`
trap_host_port	The port to send SNMP Traps to `trap_host_port=162`
trap_receive_port	The port to receive SNMP Traps on `trap_receive_port=162`

PROPERTY NAME	PROPERTY DESCRIPTION and EXAMPLES
trapengine_disabled	Enables / Disables the Trap Manager Engine. If disabled then the engine cannot receive traps. `trapengine_disabled=false`
webserver_disabled	Enables / Disables the embedded WEB UI `webserver_disabled=false`
webserver_port	The TCP port the webserver (WEB HTTP service) listens on `webserver_port=8080`
webservice_port	The TCP port the webservice (API service) listens on `webservice_port=1970`

API

MonitorEngine has a built in webservice interface that can be used for administration. Almost everything can be done via the API. The UIs also use the same webservice interface to communicate with the system. The API runs on port 1970 by default. To change this port, please see the PROPERITES section of this document and refer to the '**webservice_port**' property.

To see an example of the WSDL of the running service, use the following command on the local system: http://localhost:1970/XmlApi/XmlApiService?wsdl

There is also a command line tool supplied for all operating systems during the install process, called **xmlApiClient**. The command line structure takes seven parameters:

1. **operation** (required) – the specific operation to be performed ("-o")
2. **object** – the id or name of the object on which to perform the operation ("-i")
3. **value** – the parameter or value to be set ("-v")
4. **address** - the address of the service (defaults to localhost) ("-a")
5. **username** - the username to authenticate with (defaults to 'bytesphere') ("-u")
6. **password** - the password to authenticate with (defaults to 'bytesphere') ("-p")
7. **wsdl** - the wsdl file to use (defaults to the local wsdl file) ("-w")

Not all parameters are required for all commands, but the -o (operation) parameter is required. For example, on the local machine, the client command is called like so:

```
xmlApiClient -o {operation} -i {object(s)} -v {value}
```

For example, to check whether or not a remote system is alive, use the "-a" parameter:

xmlApiClient –o system.alive –a http://{remoteIP}:1970/XmlApi/XmlApiService

API Response Codes

Most of the time, the API responds with an error code and then the result. For example, the response to "system.alive" should be "0|OK". The first number is the API response code, and the rest of the response is either the response itself or a descriptive string of the code or error. At other times but only in special circumstances (i.e. report.results.get), the response will be a very large string with no error code.

CODE	DESCRIPTION
0	OK
1	Operation does not exist
2	Exception Occurred
3	Monitor does not exist
4	Agent does not exist
5	Monitor could not be deleted
6	Could not get required dispatcher
7	Could not set frequency
8	Monitor ID not specified
9	Agent ID not specified
10	Operation failed
11	No such monitor type exists
12	No such access type exists
13	Operation not allowed
14	Disabled
15	Job ID not specified
16	No jobs exist

API Operations List

Following is a full list of all the operations available via the API.

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
access.community.get	Retrieves the SNMP community string for a particular access type. Retrieval requires the access_id, please see the database schema section for more information. `-o access.community.get -i {access_id}`
access.community.set	Sets the SNMP community string for a particular access type. Requires the access_id, please see the database schema section for more information. `-o access.community.set -i {access_id} -v {community_string}`
agent.config.reset	Resets all timeout values for the entire agent configuration. `-o agent.config.reset`
agent.delete	Deletes a specific agent from the agent configuration. `-o agent.delete -i {agent_id}`
agent.enabled.set	Enables / Disables a specific agent in the agent configuration. Disabling an agent turns off polling and it will not be turned back on until manually re-enabled, even on discovery (in effect, this is the same as “retiring” an agent. `-o agent.enabled.set -i {agent_id} -v false`
agent.frequency.set	Sets the polling frequency for ALL monitors belonging to a specific agent in the agent configuration. `-o agent.frequency.set -i {agent_id} -v 60000`
agent.organization.get	Gets the organization ID for a specific agent in the agent configuration. `-o agent.organization.get -i {agent_id}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
agent.organization.set	Sets the organization ID for a specific agent in the agent configuration. `-o agent.organization.get -i {agent_id} -v {organization_id}`
eventmgr.{facility}.ack	Ack a particular event from the EVENT_TYPE {facility} specified. Valid facilities are currently 'trap' or 'syslog'. For a full list , please see the MonitorConsts. EVENT_TYPE Enum Constant, in the CONSTANTS section of this document. The 'event_id' to be specified must be the id of the actual event generated by {facility}. For example, if it's 'trap', then the trap_id must be specified. If it's 'syslog', then the syslog_ id must be specified. The rest of the ack, clear, and unack operations for eventmgr all follow suit. `-o eventmgr.trap.ack -i {event_id}`
eventmgr.{facility}.ack_n	Ack one or more events from the specified facility. `-o eventmgr.trap.ack_n -i {event_id1,event_id2}`
eventmgr.{facility}.ackall	Ack all events from the specified facility. `-o eventmgr.trap.ackall`
eventmgr.{facility}.clear	Clear an event from the specified facility. `-o eventmgr.syslog.clear -i {event_id}`
eventmgr.{facility}.clear_n	Clear one or more events from the specified facility. `-o eventmgr.syslog.clear -i {event_id1,event_id2}`
eventmgr.{facility}.clearall	Clear all events from the specified facility. `-o eventmgr.syslog.clearall`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
eventmgr.{facility}.unack	UnAck an event from the specified facility. `-o eventmgr.trap.unack -i {event_id}`
eventmgr.{facility}.unack_n	UnAck one or more events from the specified facility. `-o eventmgr.trap.unack_n -i {event_id1,event_id2}`
eventmgr.{facility}.unackall	UnAck all events from the specified facility. `-o eventmgr.trap.unackall`
mibwalker.walk	Performs an SNMP MIBWalk on a target agent. All the parameters are specified in the -v switch. -req <request> send request to agent. (0=get_bulk,1=get_next) -v 1|2c|3 SNMP version -ip <hostaddress> IP Address of Host -a <outputfile> File to Append to -f <outputfile> File to Output -o <targetOid> OIDs to Target -p <port> SNMP Port -m <max_repetitions> Maximum number of repetitions -r <attempts> Number of retries -t <secs> SNMP Timeout -forceGB Force GetBulkRequest, do not walk -hc <mode> STRING output (0 = hex, 1 = ascii, 2 = text) -c <community> SNMP Community (SNMPv1/v2c only) -u <username> Username (required) (SNMPv3 only) -E <engineid> Context Engine ID (SNMPv3 only) -n <name> Context Name (SNMPv3 only) -ap <authproto> Authentication protocol <md5|sha> (SNMPv3 only) -A <password> Authentication password (SNMPv3 only) -X <password> Privacy password (SNMPv3 only) `-o mibwalker.walk -v "ip={address},v 2c, o=1.3.6.1.2.1.1.2.2.1.2"`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
monitor.config.reset	Resets all the monitor configuration polling states. Only use when directed by technical support. `-o monitor.config.reset`
monitor.delete	Deletes a particular monitor from the configuration (both in memory and the DB). Either the monitor ID or the NAME can be used. `-o monitor.delete -i {monitor_id | monitor_name}`
monitor.enabled.get	Gets the enabled state of a particular monitor. Either the monitor ID or the monitor NAME can be used. `-o monitor.enabled.get -i {monitor_id}`
monitor.enabled.set	Sets the enabled state of a particular monitor. Either the monitor ID or the NAME can be used. `-o monitor.enabled.get -i {monitor_id} -v true`
monitor.frequency.get	Gets the frequency of a particular monitor. Either the monitor ID or the NAME can be used. `-o monitor.frequency.get -i {monitor_id}`
monitor.frequency.set	Sets the polling frequency of a particular monitor. Either the monitor ID or the NAME can be used. Specify milliseconds. `-o monitor.frequency.set -i {monitor_id} -v 60000`
monitor.organization.get	Gets the organization ID of a particular monitor. Either the monitor ID or the NAME can be used. `-o monitor.organization.get -i {monitor_id}`
monitor.organization.set	Sets the organization of a particular monitor. Either the monitor ID or the NAME can be used. `-o monitor.organization.set -i {monitor_id} -v 2`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
monitor.realtime.get	Gets the realtime frequency of a particular monitor. Either the monitor ID or the NAME can be used. `-o monitor.realtime.get -i {monitor_id}`
monitor.realtime.set	Sets the realtime frequency of a particular monitor. Either the monitor ID or the NAME can be used. Specify milliseconds. `-o monitor.realtime.set -i {monitor_id} -v 15000`
monitor.states.list (D)	Gets the summary of states for all monitors `-o monitor.states.list`
monitor.stats.get	Gets the last polled statistic for a particular monitor. `-o monitor.stats.get -i {monitor_id} -v {stat_id}`
monitor.type.get	Returns the monitor type for a particular monitor `-o monitor.type.get -i {monitor_id}`
monitor.type.set	Sets the monitor type for a particular monitor `-o monitor.type.get -i {monitor_id} -v {type}`
notifier.events.ack	Acknowledges an event. Event_id is the database id of the event. `-o notifier.events.ack -i {event_id}`
notifier.events.ack_n	Acknowledges one or more events. Use comma to separate event ids. `-o notifier.events.ack_n -i {event_id1,event_id2}`
notifier.events.ackall	Acknowledges all events. `-o notifier.events.ackall`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
notifier.events.add	Adds an event. Use when wanting to trigger an externally generated Notifier Event or Exception that is not triggered by the engine. Takes the following parameters: **object**: target object (could be monitor name, IP address, etc.) **event_type**: type of event (see schema **NotifierEventType**) **severity**: severity of event (see schema **NotificationSeverityType**) **message**: text message describing the event **exception_key**: if triggered by an exception, the key of the exception `-o notifier.events.add -v "object={object},event_type=THRESHOLD_OUTOFRANGE,severity=CRITICAL,message='Threshold exceeded for Errors'"`
notifier.events.clear	Clears an event in the notifier (in memory and database). `-o notifier.events.clear -i {event_id}`
notifier.events.clear_n	Clears one or more events in the notifier (in memory and database). `-o notifier.events.clear_n -i {event_id1,event_id2}`
notifier.events.clearall	Clears ALL events in the notifier (in memory and database). `-o notifier.events.clearall`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
notifier.events.lookup	Returns all matching events in the notifier by either monitor id or exception key. This allows the caller to figure out if there are any events directly generated by a particular monitor OR if there are any events that match a specific triggered exception. Valid parameters are 'monitor_id', 'exception_key', or 'mid_ekey' (for both monitor id and exception key. If using the latter method, the string passed in should use the pipe delimeter between the monitor id and exception key (e.g. 10683|bytes_total_100.1.0) `-o notifier.events.lookup -v monitor_id=10683`
notifier.events.unack	UnAck an event in the notifier (in memory and database). `-o notifier.events.unack -i {event_id}`
notifier.events.unack_n	UnAck one or more events in the notifier (in memory and database). `-o notifier.events.unack_n -i {event_id1,event_id2}`
notifier.events.unackall	UnAcks ALL events in the notifier (in memory and database). `-o notifier.events.unackall`
notifier.mailserver.test	Tests the a SMTP mail server with the specified parameters. The test does not actually send an e-mail but it does connect to the server and login. Parameters: -smtp_server {ipaddress | hostname} - the server to test -smtp_username {username} - the username to login with -smtp_password {password} - the password to use for login `-o notifier.mailserver.test -v smtp_server={hostname},smtp_username={username},smtp_password={password}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
notifier.users.add	Adds a user to the notifier users list. Must pass the username and at least 1 parameter. Parameters include 'email_address' (in order to get email notifications), 'trap_address' (in order to send a SNMP TRAP), 'database' to log to database. `-o notifier.users.add -i {username} -v {email_address={email_address},trap_address={trap_address},database=0}`
notifier.users.remove	Removes a user from the notifier users list. Must pass the username. `-o notifier.users.remove -i {username}`
report.results.clear	Clears the results of a report from cache. All report results are stored in cache for a period of 1 poll period longer than the timeframe of the report (running a report over the same time period does not generate a new report - it returns the report that has already been generated). `-o report.results.clear -i {report_id}`
report.results.get	Gets the results of a report `-o report.results.get -i {report_id}`
report.schedule.add	Schedules a report, and returns a report_id. The 'report.results.get' must be called with the returned report_id in order to retrieve the results. Object must be the report name and the parameters passed include a set of arguments describing the report. For more information, please see the REPORT ENGINE section of this document. `-o report.schedule.add -i {report_name} -v {args}`
report.schedule.cancel	Cancels a scheduled or running report. `-o report.schedule.cancel -i {report_id}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.accesstypes.updated	Notifies the system that the accesstypes have been updated and the engine will reload them from the database. `-o system.accesstypes.updated`
system.alive	Tells if system is alive and well, responding with an "0|OK". `-o system.alive`
system.component.details	Gets the detailed state of a specific component of the engine. Only 1 component can be specified at a time (or ALL is OK as well). Please see components above. `-o system.component.state -v DE`
system.component.state	Gets the overall state of a specific component of the engine. Only 1 component can be specified at a time (or ALL is OK as well). ALL = ALL MRM = Monitor Results Manager PE = Poller Engine DE = Discover Engine OE = Output Engine NE = Notifier Engine RAE = Results Analysis Engine SP = Symbol Processor PM = Ping Monitor PB_SNMP = Poller Blade (SNMP) `-o system.component.state -v ALL`
system.config.read	Tells the system to read in a configuration file. Configuration will be merged with the running configuration. If no file is specified, the default will be used 'monitor.cfg.xml'. `-o system.config.read -v {file}`
system.config.write	Tells the system to write the running configuration out to a configuration file. All data will be written to 'monitor.cfg.xml' by default, unless a file is specified. `-o system.config.write -v {file}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.debug.database	Only run when instructed by support. Launches a database debugging session, but locks out the engine from making changes to it while the session is running. This should not be performed on a production machine, as the monitorengine must be shut down in order to reset everything. The database can be queried normally while the engine runs, for more information please see the DATABASE chapter. `-o system.debug.database`
system.discover	Schedules a discovery and adds it to the queue. If nothing is in the queue, it will execute immediately. Arguments to be specified in name=value pairs, comma separated. Arguments: **access_id** - use a specific access_id for the access parameters **auth_password** - the authentication password **auth_proto** - the authentication protocol (NONE, MD5, SHA) **context** - the SNMPv3 context name **community** - the SNMP community string (default) **community_read** - the SNMP community string (read-only) **community_write** - the SNMP community string (read-write) **host** - specifies a single host to process **jobid** - when set to 1, the discover jobid is returned. **priv_proto** - the privacy protocol (NONE, CBC_DES) **priv_password** - the privacy password **port** - port to query on (SNMP uses 161 by default) **profile_id** - execute a pre-defined discover profile **range** - specifies the range of IP addresses to process **type** - type of discover (i.e. **snmp**, icmp, ipmi, etc.) **username** - the username for discovery **version** - version of SNMP to use (SNMP only) `-o system.discover -v {arg1=value1,arg2=value2}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.discoverstats	Returns realtime statistics for the discover process. `-o system.discoverstats`
system.dispatchers.start	Starts the poller dispatchers (i.e. enables polling). `-o system.dispatchers.start`
system.dispatchers.stop	Stops the poller dispatchers (i.e. disables polling). `-o system.dispatchers.stop`
system.exceptions.add	Adds an exception definition to the system. Arguments to be specified in name=value pairs, comma separated. For more information, please see the EXCEPTIONS chapter in this document. Arguments: **action** - notify, watch_normal, watch_realtime **compare_object** - baseline_recent, baseline_running, value_current, value_last, value_max, value_min, value_sum **compare_type** - less_than, greater_than, range, contains, equal, notequal **compare_value** - the value to compare to **exception_text** - the description text of the exception **monitor_id** - the ID of the monitor to specifically create this exception for **monitor_type** - the monitor type for this exception **object_id** - object to watch **reset_count** - number of times polled value must not trip the exception **realtime_poll_period** - frequency of realtime polling action in ms **severity_id** - the severity of the exception **trip_count** - number of times value must trip exception to trigger action `-o system.exceptions.add -v {arg1=value1,arg2=valu e2,arg3=value3,arg4=value4}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.exceptions.delete	Delete the exception definition including all events derived by a matching exception from the system, by either the monitor_id or the exception_key. `-o system.exceptions.delete -v exception_key={exception_key}`
system.exceptions.updated	Tells the system to reload exception definitions for all exceptions based on a specific monitor_type (‘type’) or a specific exception db_id (‘id’). `-o system.exceptions.updated -v type={type}`
system.gc	Forces garbage collection. `-o system.gc`
system.groups.updated	Tells the system to re-read the groups from the database. `-o system.groups.updated`
system.jobs.cancel	Cancels a running or queued job `-o system.jobs.cancel -i {job_id}`
system.jobs.list	Lists the current jobs running and queued. `-o system.jobs.list`
system.license.info	Displays the current license information `-o system.license.info`
system.license.load	Attempts to load the current license(s) `-o system.license.load`
system.license.usage	Displays the current license usage `-o system.license.usage`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.logging.component	Specifies which system component to enable logging for. Unlike the 'state' command, multiple components can be specified here. ALL = ALL MRM = Monitor Results Manager PE = Poller Engine DE = Discover Engine OE = Output Engine NE = Notifier Engine RAE = Results Analysis Engine SP = Symbol Processor PM = Ping Monitor PB_SNMP = Poller Blade (SNMP) `-o system.logging.component -v DE,RAE,SP`
system.logging.database	Enables/Disables detailed debugging information in the database (e.g. SQL queries, reponses, etc.) `-o system.logging.database -v true`
system.logging.level	Sets the logging level (1-10). 1 least, 10 is most. `-o system.logging.level -v 5`
system.logging.priority	Sets the logging priority. INFO is the default. Valid values are ERROR, INFO, WARN, DEBUG. `-o system.logging.priority -v DEBUG`
system.pollerblades.reset	Resets the poller blades. Only use with tech support. `-o system.pollerblades.reset`
system.pollerstats	Returns realtime statistics for the polling process. `-o system.pollerstats`
system.properties.delete	Deletes a system property `-o system.properties.delete -i {property_name}`
system.properties.get	Returns the property value for the property requested. `-o system.properties.get -i {property_name}`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.properties.list	Lists all the system properties and their values `-o system.properties.list`
system.properties.rename	Renames a system property `-o system.properties.rename -i {old_property_name,new_property_name}`
system.properties.set	Set a property value for the property specified. `-o system.properties.set -i {property_name} -v "{property value}"`
system.properties.write	Writes the property values out to the engineconfig.properties property file. `-o system.properties.write`
system.relations.reset	Recalculates all system relations. `-o system.relations.reset`
system.shutdown	Shuts down the system. `-o system.shutdown`
system.status	Displays the current system status `-o system.status`
system.threads.list	Displays all threads in use and their states `-o system.threads.list`
system.threads.summary	Displays a summary of all threads in use and states `-o system.threads.summary`
system.traplisteners.start	Starts the Trap Manager Listeners `-o system.traplisteners.start`
system.traplisteners.stop	Stops the Trap Manager Listeners `-o system.traplisteners.stop`

OPERATION NAME	OPERATION DESCRIPTION and EXAMPLES
system.trapstats	Displays the statistics for Trap Manager Service `-o system.trapstats`
system.types.reset	Resets the Monitor Types and re-reads all currently known types into the system. By default does not read newly added types into the system. `-o system.types.reset`
system.version	Returns the engine version. `-o system.version`
system.version.web	Returns the web-engine version, if running `-o system.version.web`
system.webserver.restart	Restarts the webserver. `-o system.webserver.restart`
trapmgr.filters.deleted	Tells Trap Manager Service (TMS) that a specific filter was deleted. TMS will delete that filter from the db and memory and remove it from the processing trees. `-o trapmgr.filters.deleted -i {filter_id}`
trapmgr.filters.read	Tells Trap Manager Service (TMS) to read a trap-filters file. If no file is specified, it will default to the system file. `-o trapmgr.filters.read -i {filter_file}`
trapmgr.filters.updated	Tells Trap Manager Service (TMS) that a specific filter was updated. TMS will update that filter in the db and memory and consequently update the processing trees. `-o trapmgr.filters.updated -i {filter_id}`

Common API examples

Delete an agent:
```
xmlApiClient –o agent.delete –i <agent-id>
```

Delete a monitor:
```
xmlApiClient –o monitor.delete –i <monitor-id>
```

Get the monitor frequency:
```
xmlApiClient –o monitor.frequency.get –i <monitor-id>
```

Set the monitor frequency to 30 seconds:
```
xmlApiClient –o monitor.frequency.set –i <monitor-id> -v 30000
```

Get the monitor enabled status:
```
xmlApiClient –o monitor.enabled.get –i <monitor-id>
```

Set the monitor enabled status to **false**:
```
xmlApiClient –o monitor.enabled.set –i <monitor-id> -v false
```

Launch discover with a specific range:
```
xmlApiClient –o system.discover -v range=192.168.1.1-254
```

Launch discover with a specific range and different SNMP community string:
```
xmlApiClient –o system.discover -v range=192.168.1.1-254,community=test
```

Launch discover for a specific IP and port:
```
xmlApiClient –o system.discover -v range=192.168.1.80,port=5000
```

Set the system parameter *snmp_community_string* to **private**

```
xmlApiClient –o system.config.set –i snmp_community_string -v private
```

Add a Notifier user to the system and specify email address:
```
xmlApiClient –o notifier.users.add –i <username> -v email_
address=<email>
```

Update notifier user in the system specifying email and trap IP addresses:

```
xmlApiClient –o notifier.users.add –i <username> -v  email_
address=<email>, trap_address=<trap-ip-address>
```

Acknowledge an event in the system:

```
xmlApiClient –o notifier.events.ack –i <event_id>
```

Clear an event in the system:

```
xmlApiClient –o notifier.events.clear –i <event_id>
```

Update exceptions in the system for the snmp-mib2-if monitor-type:

```
xmlApiClient.exe -o system.exceptions.updated -i type -v snmp-mib2-if
```

Update a community string:

```
xmlApiClient –o access.community.set –i <access_id> -v read_community/
write_community
```

Monitor Configuration

MonitorEngine needs to be told which things to monitor, and how to monitor them. This section will explain how to update the system with the set of things to be monitored, as well as how to define custom monitoring types.

ConfigUpdate File

To tell MonitorEngine what to monitor, a configuration update file must be created. The default filename for the configuration update file is *monitor.cfg.xml*.

As this is an XML file, we have included a schema describing the structure. The schema describing the actual format can be found in the 'system' subdirectory from the install location. The default locations are listed below.

```
Linux: "./system/cConfigUpdate.xsd"
Windows: ".\system\cConfigUpdate.xsd"
```

The configuration update file has a basic structure like this:

```
<ConfigUpdate>
        <agentConfig>
                <agent_ip/>
        </agentConfig>
        <accessConfig>
                <access_type/>
        </accessConfig>
        <monitorConfig>
                <monitor/>
                <monitor/>
                <monitor/>
                <monitor/>
        </monitorConfig>
</ConfigUpdate>
```

An example file can be found in the main directory of a MonitorEngine install, it is called *monitor.cfg.xml.example*. There are three main parts, the ***agentConfig***, ***accessConfig***, and ***monitorConfig***. Each section can handle an unlimited number of agents, access types, and monitors, respectively.

AgentConfig

The ***agentConfig*** lists all the IP addresses or hostnames that will be monitored. Agent IDs must be numbered and unique. A section for the agentConfig with 2 agents could look like this:

```
<agentConfig>
        <agent_ip agent_id="1">
                <ipaddress>192.168.1.1</ipaddress>
                <ping>true</ping>
        </agent_ip>
        <agent_ip agent_id="2">
                <ipaddress>192.168.1.5</ipaddress>
                <ping>true</ping>
        </agent_ip>
</agentConfig>
```

AccessConfig

The ***accessConfig*** contains a list of protocol/port based access types, and for each of those types, the details used to define access. A section with one SNMP access type for the accessConfig would look like this:

```
<accessConfig>
        <access_snmp port="161" access_id="1">
                <snmp_version>2</snmp_version>
                <community_read>public</community_read>
                <community_write>private</community_write>
        </access_snmp>
</accessConfig>
```

MonitorConfig

The monitorConfig contains a list of monitored elements.

Monitor section attributes

The monitor section of the monitor config has several attributes:

agent_id – points to the agent (required)
access_id – points to the access type (required)
monitor_type – points to the monitor type (required)
frequency – the frequency in milliseconds to polls this monitor (optional)
schedule_group – the scheduled polling group that this monitor belongs to (optional)
debug – show debugging for this monitor when monitor debug is on (optional)

MonitorConfig sample

A monitorConfig which monitors the first 3 interfaces from each router agent using SNMP may look like this:

```
<monitorConfig>
        <monitor agent_id=”1” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>1</index>
        </monitor>
        <monitor agent_id=”1” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>2</index>
        </monitor>
        <monitor agent_id=”1” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>3</index>
        </monitor>
        <monitor agent_id=”2” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>1</index>
        </monitor>
        <monitor agent_id=”2” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>2</index>
        </monitor>
        <monitor agent_id=”2” access_id=”1” monitor_type=”snmp-mib2-if”>
                <index id=”1”>3</index>
        </monitor>
</monitorConfig>
```

Monitor Types

MonitorEngine out-of-the-box comes with a set of pre-defined monitor files. They are a set of XML files that define the way the MonitorEngine needs to collect data.

Monitor Type system files

Monitor Types properties and relations are defined by a set of 3 XML files in the system subdirectory named ***monitor-types.xml***, ***monitor-descriptions.xml***, ***monitor-objects.xml***.

Please see the cMonitorType.xsd schema file in the system directory for detailed information about file structure, and what values are permissable and supported.

The ***monitor-objects.xml*** file contains one or more descriptive entries for each object referenced by the monitor-types.xml file (one for each supported language).

The ***monitor-description.xml*** file contains one or more descriptive entries for each monitor type defined in the monitor-types.xml (again, one for each language).

The ***monitor-types.xml*** file defines the objects for each monitor type, as well as any relations to other monitor types. The basic format of this file is this:

```
<MonitorTypes>
        <monitor_type>
                <objects>
                        <object/>
                </objects>
                <relations>
                        <relation/>
                </relations>
                <exceptions>
                        <exception/>
                </exceptions>
        </monitor_type>
</MonitorTypes>
```

There can be unlimited monitor type sections. Monitor Types are hierarchical, and base types must be defined before descendents.

Monitor Objects

These are the actual objects defined in the monitor type definition. It defines the name of an object, how it will be stored, how data will be treated, etc. An example of an object definition for a monitor type could be:

```
<object id="bytes_in" storage_id="c1" storage_type="counter"/>
```

The storage_id refers to the external database column that data will be stored in. Storage_type refers to the storage methodology used (i.e. **gauge** means data will be multiplied over the poll interval, **counter** means that the delta will be used over the poll interval, and **raw** means that there will be no transformation).

Monitor Relations

Relations are a nifty way to relate one monitor type to another, and then the system can do relational calculations on data (e.g. aggregations and sums, averages, min, max, etc.). One could relate interface or trunk monitor types to the router monitor type, and then look at total interface utilization or average utilization over the router, over time. Some relation examples could be:

```
<relation id="interface" type="agent"/>
<relation id="memory" type="agent"/>
<relation id="cpu" type="agent"/>
```

Monitor Exceptions

Exceptions define triggers for the Notifier based on thresholds, polled or calculated values, etc. If there are exceptions to be defined for a particular type, do it here:

```
<exception text="Printer Toner Low" param="prtPercentageConsumed" compare_
value="90" compare_type="greater_than" compare_object="value_current" action="notify"
severity="warning" trip_count="1" reset_count="10" realtime_poll_period="0"/>
```

Monitor Definition Files

The monitor file definitions tell the engine how to monitor something by specifying the protocol to use, the objects to query for, and the expressions to use for calculations. In addition exceptions can be defined. Every monitor file is based on a monitor-type.

All the monitor definitions for the base installation are XML files that live in the **monitor** directory. This directory can be changed by changing the system property 'monitor_file_location', please see the PROPERTIES section of the document.

All of these XML definition files are described by the master schema. Please see the cMonitorType.xsd schema file in the system directory for detailed information about file structure, and what values are permissable and supported.

The basic structure of the monitor definition file can be found described below. Each MonitorType definition file can have up to 5 sections, ***Init***, ***Declares***, ***Queries***, ***Expressions***, and ***Exceptions***. The only required sections are the first three.

Init section

Init describes which type in the system this monitor type represents, the ASN.1 MIB from which to get OID definitions, and the default access type used for getting data. Following is an example init section.

```
<init>
        <monitor_type>snmp-mib2-if</monitor_type>
        <default_mib>IF-MIB</default_mib>
        <default_access_type>snmp</default_access_type>
</init>
```

Declare section

Declare section defines OIDS, constants, and index values for use during polling. An example of a declare section could be:

```
<declares>
        <declare id="ifSpeed" type="oid">1.3.6.1.2.1.2.2.1.5</declare>
        <declare id="ifHighSpeed" type="oid">1.3.6.1.2.1.31.1.1.1.15</declare>
        <declare id="ifOperStatus" type="oid">1.3.6.1.2.1.2.2.1.8</declare>
        <declare id="ifLastChange" type="oid">1.3.6.1.2.1.2.2.1.9</declare>
        <declare id="ifInDiscards" type="oid">1.3.6.1.2.1.2.2.1.13</declare>
        <declare id="ifInErrors" type="oid">1.3.6.1.2.1.2.2.1.14</declare>
        <declare id="ifInUnknownProtos" type="oid">1.3.6.1.2.1.2.2.1.15</declare>
        <declare id="ifOutDiscards" type="oid">1.3.6.1.2.1.2.2.1.19</declare>
        <declare id="ifOutErrors" type="oid">1.3.6.1.2.1.2.2.1.20</declare>
        <declare id="ifHCInOctets" type="oid">1.3.6.1.2.1.31.1.1.1.6</declare>
        <declare id="ifInOctets" type="oid">1.3.6.1.2.1.2.2.1.10</declare>
        <declare id="$ifIndex" type="index">1</declare>
</declares>
```

Query Section

Queries define the actual data collection details for the monitor type. There is a maximum of 128 queries per monitor type. There is a default, and one or more optional alternate query methods. The query class must be specified, as well as the query alias, which will be used for reference by subsequent queries and/or expression calculations. Following is a simple query for the object operstatus. It has been classified as a "status" variable by using the "class" attribute. It uses a SNMP GET method to query ifOperstatus at ifIndex:

```
<query>
        <attributes alias="operstatus" class="status"/>
        <default method="snmp_get">ifOperStatus.$ifIndex</default>
</query>
```

Following is a complex query for the object speed. It has been classified as a utilization variable. The default query is for the ifSpeed object, but if the ifHighSpeed object is present, this will be chosen instead. Units are specified by the "units" attribute.

```
<query>
        <attributes alias="speed" class="utilization"/>
        <default method="snmp_get" units="bps">ifSpeed.$ifIndex</default>
        <alternate method="snmp_get" units="mbps">ifHighSpeed.$ifIndex </alternate>
</query>
```

Following is another complex query, that tell the MonitorEngine to poll the ifHCInOctets object if the value of the speed object (previously defined and polled), is greater than the value of 20Mbps. If not, the default ifInOctets object will be polled.

```
<query>
        <attributes alias="bytes_in" class="utilization"/>
        <default method="snmp_get">ifInOctets.$ifIndex</default>
        <alternate method="snmp_get" param="speed" compare="greater_than"
                value="20000000">ifHCInOctets.$ifIndex
        </alternate>
</query>
```

Finally, a query for non-unicast packets in will result in aggregation of results of two high-capacity variables, ifHCInMulticastPkts and ifHCInBroadcastPkts, if the value of speed is greater than 640Mbs. If not, the default, ifInNUcastPkts, will be queried.

```
<query>
        <attributes alias="packets_nucast_in" class="packets" alt="all"/>
        <default method="snmp_get">ifInNUcastPkts.$ifIndex</default>
        <alternate method="snmp_get" param="speed" compare="greater_than"
                value="640000000">
                ifHCInMulticastPkts.$ifIndex
        </alternate>
        <alternate method="snmp_get" param="speed" compare="greater_than"
                value="640000000">
                ifHCInBroadcastPkts.$ifIndex
        </alternate>
</query>
```

Expressions Section

Expressions describe to the MonitorEngine how the collected data will be transformed. Very often it is just the query object itself. But, it can also be any mathematical expression involving any of the query objects in the current or a related monitor type.

```
<expression object_id="operstatus">operstatus</expression>
```

To get speed_in and speed_out, set them each to the polled speed:

```
<expression object_id="speed_in">speed</expression>
<expression object_id="speed_out">speed</expression>
```

Then, to get total speed, sum speed_in and speed_out:

```
<expression object_id="speed_total">speed_in + speed_out</expression>
```

The following expression defines the value for the object **utilization_line_in,** which is the utilization for the interface at the time of the poll. It multiples bytes_in by 8 bits/byte and then by 100 (for percentage) and then divides by the product of poll time and speed. **_pollTimeSecs** is an internal variable.

```
<expression object_id="utilization_line_in">
       (bytes_in * 8 * 100) / (_pollTimeSecs * speed_in)
</expression>
```

For calculation of the total bytes in for all router interfaces on a router, use the relation_id and transform attributes:

```
<expression object_id="bytes_in" relation_id="interface" transform="sum">bytes_in
</expression>
```

Complex expressions with multiple parts can be specified, including setting of variables inline and multiple if/then/else statements.

```
<expression object_id="scale_test">
       returnValue1=X;returnValue2=Y;if (scale=10) then returnValue1 else returnValue2;
</expression>
```

Reserved Keywords

The following are special, reserved keywords in the Symbol Processor:

_pollTimeSecs - the number of seconds in the poll period
_pollTimeMSecs - the number of milliseconds in the poll period
_availability - the availability of the monitor during the current poll period
_latency - the latency of the monitor over current poll period
_ kwh_hourly_cost - hourly cost per kilowatt hour over current poll period

Built-in Functions

There are a number of real-time built-in processing functions in the base engine, which are accessible during expression evaluation. To call a built-in function, simply use the name of the function and specify the arguments. Function names always start with an underscore ("_"), as do reserved keywords, and they end with the string "Function".

```
<expression object_id="custom_function_call">_thisFunction(arg1,arg2)</expression>
```

RegExpFunction

_regExpFunction - this function allows values to be pulled out of a polled value, using a regular expression, and return it to the expression evaluator. Values can be numbers or strings.

usage: **_regExpFunction**(value,expression,index)

e.g.:

assuming your polled value is "milliAmpsAt42v", and you wanted to get the number of volts specified in the string, one could come up with an expression:

_regExpFunction("milliAmpsAt42v","(.*)(AmpsAt)([0-9]*)",3) = 42

UnitTypeFunction

_unitTypeFunction - this function converts the value from using one unit type to another (e.g. milliseconds to seconds, or millamps to amps, or megabytes to bytes).

usage: **_unitTypeFunction(**value**,**units)

Valid values for units are: YOCTO, ZEPTO, ATTO, FEMTO, PICO, NANO, MICRO, MILLI, CENTI, DECI, NONE, DEKA, HECTO, KILO, MEGA, GIGA, TERA, PETA, EXA, ZETTA, YOTTA

e.g.:

assuming a polled value = 1000, in milliamps, and you want to convert to amps:
_unitTypeFunction(1000**,** MILLI) = 1 amp

ValueMapFunction

_valueMapFunction - this is a function that will map a polled value to another value and return it to the expression evaluator. Values need not be numbers, they can be strings as well.

usage: **_valueMapFunction**(inputValue,nullValue,test1,test2,test3)

e.g.:

assuming a polled value = 300, there is a match (300=5), so the value is 5...
_valueMapFunction(300,66,100=1,200=2,300=5,400=10) = 5

assuming a polled value = 500, there is no match, so result is nullValue which is 66...
_valueMapFunction(500,66,100=1,200=2,300=5,400=10) = 66

Custom Functions

Custom functions can be easily be written in Java and once compiled and put into the classpath, will be loaded and executable at runtime. To learn how to extend the system with your own custom functions for expression evaluation, please inquire about our SDK.

Please contact us by calling 617-475-5209 or using the form:
http://www.oidview.com/contact.html

Discover Files

During the discover process, certain files are loaded and executed depending on the agent being discovered. To extend the discover functionality, these files need to be modified and/or created. All files for the discovery process must be placed in the /discover directory. There are two special files, **services.xml**, which controls TCP service discovery, and **snmp-mib2-rules.xml**, which controls special rules concerning MIB2 behavior during the discover process. The rest of the files follow a specific format and have a very specific structure.

Please see the cMonitorType.xsd schema file in the system directory for detailed information about structure, and what values are permissable and supported.

Filename format

All filenames in the discover directory need to follow a specific nomenclature, as the system programatically executes and attempts to discover items in the network.

In general, the following filename format is used: **{protocol}-{identifier}.xml**

The {protocol} is representative of the discover protocol used. Example, 'snmp', 'ipmi', 'wmi', etc. could be used as the protocol (passed in as 'type' during the schedule discover job request), and will be specified for the file prefixes.

The {identifier} can ether be the enterprise number, the feature-set, or possibly a MIB definition or description.

The majority of them are like this:
snmp-{enterprise-number}.xml
snmp-{enterprise-number}-{feature}.xml

During the discover process, if the discover blade (e.g. SNMP) is looking at a Cisco Switch, it will load up snmp-9.xml (since Cisco's enterprise number is 9), and follow the rules inside that file. Note - a file can fork to other files, that do not have to follow the same filename format or path.

File structure

The discover file structure consists of an **init** section, a **declares** section, and a **rules** section. See schema object **DiscoverRules**. Here is the example file structure:

```
<DiscoverRules>
      <init>
            <prefetches>
                  <prefetch/>
                  <prefetch/>
            </prefetches>
      </init>
      <declares>
            <declare/>
            <declare/>
      </declares>
      <rules>
            <rule>
                  <queries>
                        <query/>
                        <query/>
                  </queries>
                  <logic>
                        <atom/>
                        <atom/>
                  </logic>
            </rule>
      </rules>
</DiscoverRules>
```

The init section contains something called “prefetches”, which collect pre-defined data from the agent, and is executed once per agent discovery.

The declares are executed once per file load. Declares define variables, constants, OIDs, and indices, etc. to be used during the discover process.

The rules are executed each time an agent is discovered, and after the init section is processed. A rule can exist by itself, or can be defined by a number of queries and logic atoms.The behavior is completely customizable and we will discuss in more detail below.

Prefetches

Prefetches help speed up the discover process. Generally, they scan an entire table or a single variable for a specific table, and that data is later used during lookup for elements that will be created as a result of walking an OID or finding instances in that table.

Each prefetch follows the following structure:

```
<prefetch alias="$IFTYPE" mib="IF-MIB">ifType</prefetch>
```

This particular prefetch allows the discover process to know the ifTypes for each index it is processing WITHOUT having to go back to the wire to do an additional SNMP GET request.

The 'alias' is the variable name that will be used later on in the rule during the discover process. The 'mib' is the name of the MIB or identifying group of statistics that this data will be associated with. The actual value (in this case it's 'ifType'), is the representative object that will be retrieved during the prefetch phase.

Declares

Declares specify different object types to be used during the discover process. There are 5 types of declares: system, oid, constant, variable, index.

system - special type of variable, to be automatically populated by the discover process. Some valid values are _SYSNAME (to get the sysname from the system table), and _CLASS (to specify the class type of the object being discovered).

oid - a variable which defines an OBJECT IDENTIFIER (re ASN.1 MIBs). OIDs are treated in a special way during the discover process.
constant - a static constant value that will not change during discovery

variable - a variable that has no initial setting but will be used dynamically

index - a special variable that represents an index into a table

Some declaration examples:

```
<declare id="_SYSNAME" type="system"/>
<declare id="ifNumber" type="oid">1.3.6.1.2.1.2.1</declare>
<declare id="$CEMP_MEM_POOL_INDEX" type="constant">1</declare>
<declare id="$D_IFTYPE" type="variable"/>
<declare id="$cpmCpuTotalIndex" type="index">1</declare>
```

Rules

Rules are the actual engine of the discover process (please see schema '**rule**' and for snmp rules, '**SNMPType**'). All discovery functionality can be achieved by using the rule-based language, no extra perl, tcl, or shell scripting is needed. Each 'rule' is comprised of attributes, and optionally 'query' and 'logic' elements.

The rule attributes are as follows:

id - the identifier (unique) for this rule
class - physical class type for elements to be created (see schema **PhysicalClassType**)
monitor_type - the monitor-type of the element that will be created
index_oid - the OID to walk in order to determine which instances to base creation
pattern_name - the pattern to use for element names
pattern_key - the pattern to use for the key
prefetch_mib - the mib to use for prefetches
virtual - is this to be a virtual element
hidden - only execute when directly referenced
singleton - only execute once per agent
index_scalar - is this a scalar element
indexes_off - do not process indexes
stop - stop if this rule is true

A rule example:

```
<rule id="cable-upstream" class="port_logical" monitor_type="snmp-
cable-upstream" index_oid="ifType" pattern_name="_SYSNAME-port-
$IFDESCR" prefetch_mib="IF-MIB">
```

Rule Queries

The rule query is a query that is executed in the protocol named by the parent file, using the attributes in the query (please see schema **QueryOidType**). There can be an unlimited number of queries, but generally, you want to use as few queries as possible. Each query must return a value of TRUE in order to continue processing a rule, unless otherwise marked as 'optional'. Optional queries are useful when needing to get additional information for naming or helping to determine which path down the decision tree may be followed.

The query attributes are as follows:

alias - the id for this query and also the variable to which query responses are assigned
oid - the object identifier that is queried
query_type - the type of query (i.e. get, getnext, etc. see schema **QueryMethodType**)
match_value - the value to match
execute - the script, logic atom or function to execute
execute_type - type of execute (i.e. atom, rule, shell_cmd, see **DiscoverExecuteType**)
optional - (true/false). If not optional and query fails, discover will fail for this rule

A query example:

```
<query alias="$PHYS_DESCR" oid="entPhysicalName.$TOTALPHYSINDEX"
       query_type="snmp_get" optional="true"/>
```

Rule Logic Atoms

Logic atoms (please see schema object **DiscoverLogicAtom**) are executed either by being called directly from a corresponding or referencing query item in the same rule, or in round-robin fashion simply because they are listed in the logic section of the same rule. All logic atoms in a particular 'rule' group must be executed and all must pass a return value of 'true' except for those that are marked as 'optional' or marked as 'subatoms'. Once all atoms have been processed (or a 'stop' has been hit), the AND results of all the atoms will be considered, and the monitor will be created for that 'rule' if there was an overall 'true' value passed back (i.e. all atoms returned true).

The logic atom attributes are as follows:

alias - the id for this atom as can also be the variable to which the result is stored
input_type - type of input for atom (see schema DiscoverVariableInputType)
input - the actual input (could be a query result, function, etc.)
function_type - the type of function (see schema DiscoverVariableFunctionType)
function - the function itself
regexp_parameter - if regexp is being used, specify pattern here
stop - should discover stop here if this atom executes and returns true
execute_type - type of atom execution (see schema DiscoverExecuteType)
execute - the actual thing/script to execute
sub_atom - (true/false) - specifies whether this atom is a sub-arom (i.e. can it be called directly by an execute command or is it the child of a parent atom)
optional - (true/false) - if set to true, it is not conditionally required to return true

Some logic atom examples:

An atom that takes the split function, uses the index OID as the value to split, and returns the first index into the alias variable $cpmCPUTotalIndex:

```
<atom alias="$cpmCPUTotalIndex" input_type="oid" input="rule" function_
type="split" function="1"/>
```

An atom that takes the result of the walked instance, and sets the variable "$TOTALPHYSINDEX" as the result:

```
<atom alias="$TOTALPHYSINDEX" input_type="result" input="rule"
function_type="set"/>
```

An atom that takes the contents of the variable "$PHYS_DESCR" and sets a new variable "$CPUNAME":

```
<atom alias="$CPUNAME" input_type="result" input="$PHYS_DESCR"
function_type="set"/>
```

Extending the Engine using Code

In addition to all the meta-data configuration that can be implemented with the system, there is also core code and several engine components that can be extended.

Extendable Components

Currently the areas of the system that can be extended include the discover and polling engines (by extending or creating new blades), output engines (for new types of databases and/or data destination), custom formulae in the analysis engine, and various file and configuration readers.

Get the SDK

We have an SDK that is currently available as part of our OEM program. Included in the SDK are documented, detailed and working code samples for extending each part of the system. To find out more, please contact us by calling 617-475-5209 or using the form: http://www.oidview.com/contact.html

Database

Supported Databases

Out of the box, Jaguar SXE supports an embedded version of H2, MySQL, and SQLServer. Some limited support for Oracle was added, but needs further work.

Performance

Performance testing has been conducted using the MySQL Community version database. Bulk data insertion for 300,000 interfaces (over 10,000,000 datapoints), was repeatedly performed in 5 minute intervals in under 30s, using standard IDE 500Gb drives running at 7500rpm.

Administration

To administer the embedded H2 database, please follow the instructions in the setup chapter of this guide under the section 'Database Web Interface'. Administration for the other databases is handled by their own respective tools.

Automated Rollups

Data is stored in the database according to the configuration settings. By default, raw data will be stored for 7 days and rolled up data will be stored hourly and daily. Data is rolled up via an automatic hourly job and the polled data from the raw data tables are averaged into an hourly table. Consequently, once a day, the hourly data is rolled up into a daily datatable.

Database Schema

The full database schema (for all supported databases), can be found on the installation disk/media in the src/database directory, in the form of SQL files. In addition, all update SQL files are present in the same directory.

Data Tables

The schema template for tables performing data storage is in a table called ‘monitor_stats’. All data tables are dervied from this base table. All tables storing data are also indexed in a configuration table named ‘monitor_stats_tables’. This table contains information about each data table (period, time opened, time closed, type of data, etc.). Finally, each data table is named according to the following pattern:

`z{createtime}_{period}_{type}`

where createtime is in milliseconds, period is the poll period for raw data tables (or the rollup period for rollup tables), and type describes the type of data table.

type = 0, raw data or rollup data (avg)
type = 1, realtime data
type = 3, rollup data (min)
type = 4, rollup data (max)

Examples of data table names:

```
Z1352862000000_60000_0 - raw polled data (60s)
Z1352610000000_3600000_4 - hourly rolled up data (max)
Z1351656000000_86400000_0 - daily rolled up data (avg)
```

Configuration Tables

All configuration information can be retrieved from the database using standard SQL. Following is a list of all the table names and descriptions. For more detail, please see the schema for reference.

TABLE NAME	DESCRIPTION
access_cfg	Table of top-level access information
access_cfg_snmp	Table describing access config for SNMP
access_cfg_snmpv3	Table describing access config for SNMPv3
access_profile	Table describing access profiles for discovery
access_profile_group	Table grouping access profiles
agent_cfg	Describes agent configuration
agent_info	Describes detailed information about each agent
alert_severities	Severities for alerts, language specific
alert_types	Describes different alert types
alerts	All notifications (see chapter on Notifier), are stored here
configuration	Polled configuration information is stored here
discover_profiles	Discovery profiles
enterprise	List of enterprise numbers
groups	List of groups configured in the system
groups_stats_summary	State summary by group, updated each default poll period
language	Languages supported
language_web	The translations for the WEB UI
locations	List of Longitude and Latitude Coordinates for Maps
mib	MIBs currently loaded in the system

TABLE NAME	DESCRIPTION
mib_object_values	MIB object detail definitions
mib_objects	MIB objects loaded
monitor_cfg	The entire monitor configuration
monitor_cfg_frequency	The frequency of each monitor
monitor_description	The description of the monitorTypes
monitor_engines	The list of remote pollers (monitor engines) running
monitor_exceptions	Exception definitions (See the chapter 'EXCEPTIONS')
monitor_group	Stores monitor IDs by group
monitor_object	The objects defined in the monitorTypes
monitor_object_unit	The units supported by the objects
monitor_stats	The template table used to create new data tables
monitor_stats_tables	List of data tables presently in the database
monitor_temp	TBD
monitor_type	Details for all Monitor Types registered in the system
monitor_type_relations	All relation definitions between Monitor Types
monitor_type_stats_summary	State summaries for Monitor Types, updated each poll
organization	List of organizations
power_inputs	List of power inputs
power_inputs_hourly	Hourly costs for power inputs
relations	Relates monitors by ID, type and object
reports	List of pre-defined reports (UI)
reports_saved	List of saved report files
scheduled_jobs	All scheduled jobs in the database

TABLE NAME	DESCRIPTION
service_thresholds	Thresholds for SLAs (not currently used)
syslog	Stores captured syslog messages
system_cfg	TBD
system_info	Stores version information about Schema, Monitor Types
system_messages	Logged system information. Purged daily by default.
tiles	Dashboard configurations (UI)
trapbuckets	Trap categories called ‘buckets’ (Trap Manager)
trapconditions	Conditions by Filter (Trap Manager)
trapfilters	Trap Filter definitions (Trap Manager)
traps	Stores captured SNMP Traps (Trap Engine)
user_access_log	User login / logout time and date
user_detail	Extra information on users (UI)
user_group_access	Relates users to group access priviledges (UI)
user_info	Basic information about users like username, email. (UI)
user_notes	User defined notes (UI)
user_profile	User Profiles for quick user creation
user_views	Dashboard details by user (UI)
user_views_layout	Dashboard layout by Dashboard ID (UI)

Exception Engine

The exception engine is a critical component that analyzes polled data in memory, in REALTIME (i.e. it does not query the database nor does it run 'jobs' over historical data). The engine compares the data with pre-defined rules in the system (those rules are definied in the '**monitor_exceptions**' table), and based on the definition criteria can trigger a number of actions based on those rules.

Please see the cMonitorType.xsd schema file in the system directory for detailed information about supported parameter types.

Exception Parameters

ID - the database ID of the exception
MONITOR_TYPE - the monitor type the exception is based on
OBJECT_ID - the object the monitor is based on
COMPARE_VALUE - the value to compare the polled value to
COMPARE_TYPE - the type of comparison to make (see **ParamComparatorType**)
COMPARE_OBJECT - object to compare to (see **ParamObjectComparatorType**)
ACTION - the action to perform - Notify, etc. (see **ExceptionActionType**)
SEVERITY_ID - the severity to assign if Notify (see **NotificationSeverityType**)
TRIP_COUNT - the number of times it must occur to trigger an action
RESET_COUNT - the number of times it must NOT occur to trigger a reset
REALTIME_POLL_PERIOD - if action is set to REALTIME, the poll period in ms
ENABLED - is this exception enabled (allows turning on/off exceptions without delete)
OBJECT_GROUP_ID - allows grouping of multiple exceptions together to trigger
MONITOR_ID - allows specific targeting of the exception for a single Monitor
EXCEPTION_TEXT - the user visible description text of the exception

Triggering an Exception

Exceptions are based on a particular object or set of objects in a monitor type. When these objects have data that crosses a certain threshold or causes an exception definition to be evaluated as 'true', then an 'exception' is triggered. Depending on the action, either

a *Notification* will be created (please see the section on '**Notifier**'), or the particular monitor that triggered the exception can be watched for some additional time at an adjusted polling frequency to see if there are additional trips or if the situation clears itself up.

Notifier

The NotifierEngine is in charge of receiving, managing and alerting on all events created by the system. Events can come through either the Exception Engine, the API, or otherwise be created by the system itself in the case of agent specific reachability events, system errors, or other miscellaneous occurances. If it is desired that the Notifier does not run or process events, it can be disabled (see the system property notifier_disabled). Notifier actions include sending an email, logging to a database, and also playing an Audio File (see the system property notifier_audio_url).

Notifier Users

Each user in the system (defined in the database, if used), has a notifier property. If the email is set, then when a notification is created, if the user is in the notification group, that user will receive an email.

Notifier Events

Events in the notifier have a lifespan (see the system property notifier_event_expire_period). Once created, depending on system configuration, an alert will be sent out initially. Then, there is a repeat period, in which after that period another alert will be sent. After the expire period, if the event has not been reissued by the system, it will be deleted from the system and alerts will not be sent out anymore (this is like an automatic clear). Events that are cleared via the API or internally (i.e. the situation rights itself), will not send out periodic alerts and will be deleted during the maintenance phase.

Duplicate Events

If a duplicate event is created and sent to the Notifier before a timeout period has passed (see the system property notifier_event_timeout_period), then it will become part of the existing event, and the last updated time will be set, essentially incrementing the existing counter and restarting the clock. If the event continues to be sent by the system, the event will essentially never ‘die’, and will repeatedly be alerted on until the user manually clears it or the situation that caused the event is fixed.

Report Engine

Jaguar has an embedded Report Engine that will produce output in XML or JSON format. Reports can be scheduled internally by the engine itself (i.e. from scheduled jobs defined by users), or they can be run ad-hoc via the API. The report engine can be disabled by modifying the system property reportengine_disabled.

Report Types

The following types of reports are supported:

Single Series

A single-series report is a report of a single variable over time.

Multi-Series

A multi-series report is a report of one or more variables over time.

Multi-Subject

A multi-subject report is a report for a single variable, over multiple subjects (i.e. monitors), over time.

Top-N Report

A top-N report is a volume report for a single variable for the top-N subjects over a period of time.

PDF Reports

The engine can also create PDF reports. The PDFs are generated by a third party library called jFreeChart and a supporting library named iText (which is a commercial product). Users of PDF reports must pay a royalty fee (as part of the license cost), so we can in turn pay the iText Corporation.

Running and Retrieving Reports

To run a report, and get the result, a report must be added to the scheduler using the API (see the API command report.schedule.add). Once the report is scheduled, a report_id will be returned.

To get the report results, again the API must be called (please see the API command report.results.get). If the results are ready, they will simply be returned as XML text (or as JSON text if that was specified in the options). If they are not ready, a return code of "RUNNING", "FAILED", or "ERROR" will be returned.

Report Arguments via API

Here is an example API call to schedule and run a report via the API.

```
./xmlApiClient -o report.schedule.add -i report_topn
-v "outputformat=xml&reportname=JaguarReport&reporttyp
e=4&charttype=8&reportcomponent=1&monitor_type=router-
generic&variables=bytes_total&datatype=hourly&periodtype=la
st24hours&pollfrequency=60000"
```

This particular example shows a request for a topN report, in XML format, named 'JaguarReport', which is going to show us the top-10 monitors using the monitor_type 'router-generic' for the variable 'Total Bytes', hourly rolled up data, over the last 24 hours, on data with a poll frequency of 60s.

Report parameter arguments are as follows:

ARGUMENT	DESCRIPTION
outputformat	the output format of the resulting report. options are ‘xml’ or ‘json’
reportname	the name of the report. does not have to be unique but cannot hurt
reporttype	integer representing the type of report 0=NONE 1=SINGLE_SERIES 2=MULTI_SERIES 3=MULTI_SUBJECT 4=TOPN
charttype	integer representing the type of chart 1=CHART_BAR_HORIZ 2=CHART_BAR_HORIZ_STACK 3=CHART_BAR_VERT 4=CHART_BAR_VERT_STACK 5=CHART_LINE 6=CHART_AREA 7=CHART_AREA_STACK 8=CHART_PIE 9=CHART_DOUGHNUT 10=CHART_XY 12=CHART_BAR_HORIZ_TOPN 13=CHART_BAR_VERT_TOPN 14=GRID 15=EXPORT_CSV)

ARGUMENT	DESCRIPTION
reportcomponent	the component to run the report on 1=Monitor 2=Relation 3=Group
monitor_type	The Monitor Type to base the report on. Only valid reports are TopN.
variables	the variable(s) to collect the data from
datatype	specifies type of data to query. Valid values: RAW, RAW_RATE, HOURLY, HOURLY_RATE, DAILY, DAILY_RATE The "_RATE" types simply divide the results by time to get a rate.
periodtype	the period over which to run the report: LIVECHART - live chart which would give the last poll period LASTHOUR - last hour of data LAST6HOURS - last 6 hours of data LAST12HOURS - last 12 hours of data LAST24HOURS - last 24 hours of data LAST7DAYS - last 7 days of data LAST30DAYS - last 30 days of data CUSTOM - use for custom date ranges (use with sd and ed args)
pollfrequency	the frequency tells the engine which tables to query data for and whether or not to use RAW or ROLLED data
sd	start date (in milliseconds) to be used with CUSTOM periodtype
ed	end date (in milliseconds) to be user with CUSTOM periodtype

Report Results

The format of the results is returned in a structure that is fairly easy to parse. A "chart" object is returned, and inside of that, unit definitions, category definitions (for axis labels - usually time), and one or more datasets... inside each dataset is a 'set' object which is itself just a single datapoint.

Report XML

Here's an example of a shortened report result. Each category has a category id (catid), and then in the dataset set objects, they each refer to the catid. That way, the datapoint lines up with the category, which is in this case, a time label.

```
<chart caption='Utilization' yAxisName='Percent - gauge'
     showLegend='1' slantLabels='1' xAxisName='Time'>
   <units>
     <unit label='Percent - gauge' id='PERCENT_GAUGE' />
   </units>
   <categories>
     <category label='Fri Nov 16 15:18:23' catid='1' />
     <category label='Fri Nov 16 15:19:23' catid='2' />
   </categories>
     <dataset id='util_line_in' seriesname='BW Util'>
          <set value='0.005582666666666666' catid='1' />
          <set value='0.005342666666666667' catid='2' />
     </dataset>
</chart>
```

Chart Definition XML Structure

Following is an example of the report return result XML structure. For a multi-series or multi-subject report, there will be several datasets, respectively. For a single-series or TopN report, there will only be 1 dataset.

```
<chart>
        <units>
                <unit/>
        </units>
        <categories>
                <category>
                </category>
        </categories>
        <dataset>
                <set/>
                <set/>
        </dataset>
</chart>
```

Glossary

AGENT
A piece of software (acting as an 'agent'), that's being monitored, is addressable using TCP/IP (or some other addressing scheme), and has X number of monitored properties (and most probably Y number of 'Monitors'). Internally, the monitorengine uses 'Agents'. The term 'Agent' is used interchangably with 'Host' and 'Device'.

DEVICE
A machine that's being monitored. Same thing as a 'Host' or 'Agent'.

ENGINE
A software component with various properties that runs in it's own thread, in perpetuity until the entire system is shut down.

HOST
A device or agent that's being monitored. Used interchangebly with 'Agent' or 'Device'

MONITOR
A group of 1 or more statistics being monitored on an 'Agent'.

POOL
A group of 'engines'. Each pool can be configured in terms of size and component type. Busy engines of a certain type will be managed by the 'Pool' of that specific type. When a new 'Engine' is needed, the 'Pool' will dynamically create one. When an engine is finished it's work, it will be returned to the 'Pool' for later use.

POLLER
A software component that 'polls' or requests data at specific intervals from an 'Agent', in discrete packets organized by 'Monitor'

REALTIME (ANALYSIS)
Data Analysis done in memory, in real-time.

REALTIME (POLLING)
Data acquistion (polling), done in intervals from 100ms to 30 seconds.

ByteSphere LLC
260 Franklin Street, Floor 11
Boston, MA 02110
USA
617-475-5209
support@oidview.com
www.oidview.com

Jaguar

ByteSphere