HEYMon 3.7
Users Guide

Release Revision

Table of Contents

Overview

The current generation of distributed enterprise software applications can contain numerous business-critical systems and processes which should be monitored to ensure proper operations, 24-hours a day.
By default most of these types of systems and applications do not offer a complete monitoring tool. Therefore, issues with any business-critical processes or transactions can go unnoticed for periods of time and this can have a negative effect on many facets of business operations and accounting.
The HEYMon application allows full, real-time monitoring of any system, feature, or process in just about any computing environment.

Description

HEYMon is an application which can be used to monitor business-critical processes and/or collect statistical data within distributed enterprise applications.
Monitoring rules are configured through the use of an XML-based properties file, and alerts are sent out via email.
The HEYMon application can monitor or generate statistics from the following types of processes:

- Database processes: Any information or statistic that can be derived from SQL queries can be monitored. Is the database running? Has an error been logged into a database table?
- Server processes: Is the app server running? Is the application responding to page view requests?
- Server metrics: Server values like RAM, disk space, processes, and windows services can be monitored or collected as statistics.
- Error log: Has an error been logged to one of the log files for the Guidewire application? Are there any statistical values in the log files which are of interest to the business?
- Internal process: Has an internal process encountered errors? Are there database records which indicate a process encountered errors?

HEYMon can be configured to monitor just about any application or Windows system environment.
Additionally, HEYMon allows users to extend its functionality by utilizing simple Java API to create job type classes to perform discreet units of work.

Primary Functions

HEYMon is designed to allow configuration and customization through the use of an XML-based properties file. HEYMon utilizes the properties file for the configuration of individual monitor jobs. At a high-level, the HEYMon application knows how to do the following:

- Send email: Emails are sent to administrators when there is an alert condition they are interested in.
- Perform SQL queries: HEYMon can query a database to check for records or deliver data sets via email.
- HTTP client: HEYMon can perform HTTP POST and GET operations as a client to any HTTP end-point.
- Read text files: HEYMon can read text files for any purpose, including the parsing of log files or the collection of data from properties files.
- Read server information such as RAM, Disk Space, Processes, and Windows Services.
- Read Windows Event logs
- Monitor drive mounts, files, or directories for changes.
- Write log files: Since HEYMon requires very little administration, all progress and issues from the monitor application are logged to a log file.

Understanding HEYMon jobs

HEYMon executes jobs which check for conditions that indicate the need for an alert. When an alert condition is determined, email is sent to the related administrator(s) with all the details.
Jobs are configured though the use of an XML-based properties file. Each job has a set of common properties which can be described as follows:

- Name: The name of the configured job.
- Description: A description of what the job does and its purpose as an alert or informational message.
- Connection: An abstract connection to something which is needed by the job. For database-related jobs, this will usually be a JDBC database connection string. For file-based jobs the connection string represents the path to the input file. A connection property has the context which is set by the Job Type.
  See the Job Type documentation for the definition of each type of connection.
- Job scan interval: Each job performs checks for alert conditions on a user-configured interval. The interval is expressed in seconds, but can be any value which fits into an integer.
- Destination: The notification destination for the job.
  HEYMon only supports email as a notification destination at this time.
- Queries: One or more abstract query patterns to be used by the job. For database-related jobs, this will actually be a SQL query. For HTTP-based jobs this will be a URL.
  See the Job Type documentation for the definition of each type of query.

- Notify Subject: The string to use when providing a notification for the job. The string in this property is used as the Subject of an email alert.
- Notify Body: The string to use when providing a notification for the job. The string in this property is used as the Body of an email alert. Some of the Job Types do not require the notify body property as they output the result of a query or a log file entry.
  See the Job Type documentation for details on when the notify body property is required.
- Job Type: The job type contains the steps to take to determine if there is an alert condition. The job type also determines the property value that is required by a job destination or query.

The programming model for HEYMon is directly aligned with the content of the job properties.
All jobs have a set of common properties, and each job has a specific Job Type that contains how the job should be run.
If there was a need to add a new job type, a programmer could create a new job type class by extending a simple API. Once the job type class was created, the job could be setup and configured through the XML-based properties file.
For the specific details of all the HEYMon Job Types, see the following section titled ‘Understanding HEYMon Job Types’

Installing HEYMon

The HEYMon distribution is a compressed file (ZIP or TAR.GZ) that contains a set of files and folders. There is no installer or installation process included with HEYMon. HEYMon can be setup and installed anywhere that you want to put it.
To install HEYMon, create a directory to hold the application and then extract the files and folders from the compressed file.

HEYMon directory on Windows:

On Windows it is suggested to NOT setup HEYMon in a directory below the 'Program Files' folder. For security reasons it is recommended to setup HEYMon in its own directory in Windows as follows:

c:\HEYMon

HEYMon should be setup on a physical hard drive (like c:/), and should not be setup on a shared drive, a network drive, or a mounted drive such as a flash drive.
HEYMon should be setup in a path that does not contain any spaces.

HEYMon directory on Mac OS/X, Linux, or UNIX:

For the non-Windows operating systems, HEYMon can be setup anywhere required. HEYMon should be setup on a physical drive mount and should not be setup on a shared drive, remote mount, or local device such as a flash drive.
HEYMon should be setup in a path that does not contain any spaces.

Configuring HEYMon properties files

HEYMon has two XML-based properties files that allow full configuration of the HEYMon engine. One of the files configures the HEYMon engine, and the other file is used to configure the HEYMon monitoring jobs.

App.properties

The HEYMon engine is configured through the file `app.properties`, which looks similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<appProperties>
   <properties>
        <property name="mail_server">172.16.0.74</property>
        <property name="logLevel">INFO</property>
        <property name="enableDBLogging">true</property>
        <property name="mailInterval">500</property>
        <property name="maxDbConnections">3</property>
        <property name="agentPort">7169</property>
        <property name="msExchangeMode">true</property>
        <property name="db_logger_connect_string">
            jdbc:jtds:sqlserver://SERVER:1433/DBName
        </property>
   </properties>
   <jdbcDrivers>
      <driverClass name="jdbc:jtds:sqlserver">
          net.sourceforge.jtds.jdbc.Driver
      </driverClass>
   </jdbcDrivers>
</appProperties>
```

The property values in `app.properties` are defined as follows:

- `mail_server`: The name or IP address of the SMTP mail server. HEYMon uses this mail server to send email alerts. SMTP mail servers and SMTP on Exchange is supported by HEYMon.
- `logLevel`: The Logging level used to log details to the heymon.log file. Allowed values are INFO, WARNING, or SEVERE. INFO is suggested when setting up HEYMon the first time to allow the most amount of logging. Change the value to SEVERE to allow the least amount of logging.
- `enableDBLogging`: Value should be set to ‘true’ to allow HEYMon to log alerts to a database.
  This feature only works in the Professional and Enterprise versions of HEYMon. The free version of HEYMon does not support DB Logging.
- `mailInterval`: HEYMon utilizes a configurable interval between sending email messages specified in milliseconds. This is done so that a sudden amount of HEYMon alerts do not overload or spam the email server and get cut-off. The interval defaults to 500 milliseconds, or a half a second. It is not recommended to set this value below 100 ms, and in most cases the default of 500 milliseconds is adequate.

- `msExchangeMode`: Determines if the SMTP server used by HEYMon is Microsoft Exchange. This value should be set to `true` if the server is any version of Microsoft Exchange.
- `agentPort`: HEYMon uses a port to communicate with remote HEYMon Agents. This value should be set to a port number which is not already in use.
  HEYMon comes with the agentPort value set to 7169.
  This value must be the same as the value in the HEYMon Agent files `HEYMonAgent.sh` and `HEYMonAgent.cmd`.
- `maxDbConnections`: HEYMon utilizes a database pool for all of its database connections. This value represents the number of starting connections in the pool for each unique database connection string. This value should be set based on the number of jobs that utilize a database connection. For example, if there were 7 jobs that utilized database connections, then this value should be set to 7. It does not matter if the connections were to the same database or to different databases, this value should be set the same as the number of jobs that use a database. See the following section on HEYMon Job Types to understand more about which job types utilize a database connection.
- `db_logger_connect_string`: This value should be set to the JDBC connection string for the HEYMon database.
  This feature only works in the Professional and Enterprise versions of HEYMon. The free version of HEYMon does not support DB Logging.
- `jdbcDrivers`: This section of app.properties is used to configure any of the database drivers utilized by HEYMon. See the following section on configuring database connections for more information on how to configure this property.

Configuring database connections

HEYMon is designed to work with any database that has a Type 4 JDBC driver.

There are four steps required to integrate a database driver with HEYMon. The first three steps only need to be performed once for each JDBC driver:

1. Get the latest Type 4 JDBC driver files. Most vendors provide one .jar file, but some provide the driver as a set of .jar files. Whatever files are provided by the vendor, all the files will be required by HEYMon.
   Add the files to the `HEYMon/lib` directory. In the location that you installed HEYMon there is a lib directory. Copy all the JDBC driver files into the lib directory.
2. Update the HEYMon startup script. The startup script HEYMon.cmd or HEYMon.sh requires you to add the jar files to the list of files in the classpath. For example, take a look at part of the classpath setting in HEYMon.cmd
   ```
   java -cp .\HEYMon3_6.jar;.\lib\serializer.jar;
   ```
   If you were adding the jTDS driver for SQL Server, which is named `jtds-1.2.5.jar`, you would add the following item to the classpath setting:
   ```
   java -cp .\HEYMon3_6.jar;.\lib\serializer.jar;.\lib\jtds-1.2.5.jar;
   ```
   The JDBC driver file(s) can be placed anywhere in the list after the `-cp` option.
3. Add the driver prefix and the driver class name to the `app.properties` xml configuration file.
   Each driver will have some documentation that shows the jdbc connection string and the main driver class name.
   For example, the jdbc connection string for the jTDS SQL Server driver has the prefix:
   ```
   jdbc:jtds:sqlserver:
   ```
   The main driver class for the jTDS SQL Server driver is:
   ```
   net.sourceforge.jtds.jdbc.Driver
   ```
   When you have the two values, you can add them to the app.properties xml configuration file. To do this, add a new driverClass element between the jdbcDrivers element, like this:
   ```
   <jdbcDrivers>
       <driverClass name="jdbc:jtds:sqlserver">
           net.sourceforge.jtds.jdbc.Driver
       </driverClass>
    </jdbcDrivers>
   ```
4. Configure the jdbc connection string for the HEYMon job in job.properties.
   Once the driver(s) are registered one time using the aforementioned steps, you can use the driver by referencing it through the jdbc connection string.
   For example, if you wanted to configure the StatisticTimedQueryJob to perform a query against SQL Server, you can enter the jdbc connection string property name as the job's connection property, like this:
   ```
   <property name="sqlsvrConnection">
       jdbc:jtds:sqlserver://DBSOMESQL:1433/GTDW;user=admin;password=foo
   </property>
   ```

HEYMon Agent

HEYMon utilizes a small agent application to allow secure connections between the HEYMon engine and the systems being monitored. Use of the agent application is optional based on your HEYMon configuration.
HEYMon Agent is a small application which was designed so that it has the least overhead on the systems where it is running. Adding the HEYMon Agent application process to any of your systems will have no noticeable impact on any existing system processes and applications.

Installing the HEYMon Agent

The HEYMon Agent has the same setup requirements as the HEYMon engine. For tips on the best place to setup the HEYMon Agent, refer to the previous section on installing HEYMon.

The HEYMon Agent can be installed on each system where it is needed by simply creating a directory and copying in some directories that came with the distribution:

- `agent`: Copy in the agent directory from the HEYMon distribution.
- `lib`: Copy in the lib directory from the HEYMon distribution.

To run the HEYMon Agent, just execute `HEYMonAgent.sh` or `HEYMonAgent.cmd` depending on your selected computer platform.

When to utilize the HEYMon Agent

The HEYMon Agent may or may not be required based on which job type is being utilized and which operating system the HEYMon engine is running on.

> **TIP**: For the highest level of security, it is recommended to use the HEYMon Agent whenever possible.

The HEYMon job types which support database connections do not require the HEYMon Agent. The HEYMon Agent can be used with database-related job types if needed. One primary use of the HEYMon Agent with database-related jobs is where the database will not accept connections from outside systems or servers. By using the HEYMon Agent with database-related job types, the connections to a given database can actually come from the local server.

The HEYMon job types which support HTTP operations do not require the HEYMon Agent. The HEYMon Agent can be used with HTTP -related job types if needed. One primary use of the HEYMon Agent with HTTP -related jobs is where the web application will not accept connections from outside systems or servers. By using the HEYMon Agent with HTTP -related job types, the connections to a given web application can actually come from the local server.

When the HEYMon engine and all monitored systems are on the same operating system it is not required to use the HEYMon Agent for many of the job types. The following job types will work without HEYMon Agent if properly configured.

- Log file job types: Log file data can usually be accessed from remote systems by the use of Shared Folders in Windows, file system mounts in Linux and MacOS/X, and Samba network mounts on any platform.
- Windows service job types: Permissions can be granted so that a Windows system can check the status of a running Service on another Windows system.

The HEYMon job types that read System Information will only work properly with the HEYMon Agent. To gather details like Available Memory, Available Diskspace, and Windows Event logs from a given system or server, the HEYMon Agent has to be utilized.
This configuration is required to ensure that monitoring of System Information does not compromise the security of the server and allow any external access.

Configuring HEYMon to use agents

All HEYMon job types are designed to work with, and without the HEYMon Agent. To configure a job to be run with HEYMon Agent, a property should be set in the job type configuration of job.properties.
This is an example of how a job type is configured to use the HEYMon Agent for the ProcessWebViewJob job type:

```
<type name="ProcessWebViewJob" useagent=”true”>
      <connectionTimeout>10</connectionTimeout>
</type>
```

If the `useagent` attribute is missing, or if it is present and not set to `true` then this job type will run **without** using HEYMon Agent.

The Agent can also be used to detect system availability. By adding the attribute `notifymissingagent` with a value of `true`, HEYMon will send an alert if the job cannot be executed due to lost communication with HEYMon Agent.

```
<type name="ProcessWebViewJob" useagent=”true” notifymissingagent=”true”>
      <connectionTimeout>10</connectionTimeout>
</type>
```

Note that setting `notifymissingagent` to `true` when `useagent` is `false` will result in no notifications.

HEYMon Agent Security Considerations

The following sections attempt to describe any security-related features or functionality related to HEYMon and HEYMon Agent.
In many cases, the description of security-related details is intentionally sparse so as to not give any cues about implementation.

HEYMon Firewall Considerations

HEYMon utilizes a TCP-IP port to communicate with the systems running the HEYMon Agent. The port is configurable, and is set to port 7169 by default.
The port configured for HEYMon may need to be allowed in either a local firewall or in a network firewall.
The HEYMon engine will make requests to the HEYMon Agent instances. HEYMon Agents listen on the configured port for requests from the HEYMon engine.
The HEYMon engine acts as a client, and is the inception of all Agent request. The HEYMon Agents act as a server and listen for, and service requests. The HEYMon applications cannot operate, and cannot be made to operate in any other roles.

HEYMon Agent Communication Security

The HEYMon engine and the remote systems running the HEYMon Agent communicate over TCP-IP. Port 7169 is configured by default, but the value can be changed to any preferred port number.
Communications between the HEYMon engine and the HEYMon Agents are encrypted. The communication protocol used by HEYMon is designed to prevent man-in-the-middle attacks, or any type of 'capture-change-retransmit' attempt.
The HEYMon Agent is designed to only send back an indication of whether an alert condition exists. The Agent cannot be told to take any actions, cannot be forced to execute processes, and cannot transmit any information other than the indication of an alert condition.

HEYMon Agent Communication Details

The HEYMon Agent is only contacted at the time a job for that specific agent is executed by the HEYMon engine. HEYMon does not send data, or 'ping' the Agents for any reason other than the execution of a scheduled job.
The data interchange between the HEYMon engine and HEYMon Agent is designed to be as compact as possible.
HEYMon is designed to only generate network traffic during the execution of a scheduled job, and generate the minimal amount of traffic required.

Understanding HEYMon Job Types

The concept of HEYMon job types is based on the need to have a generic job definition with the ability to perform a specific type of work in each job. Job Types are really just Java classes which adhere to a specific interface and contain the code required to determine if there is an alert condition.
Job types are also the extensible part of the HEYMon application. The rest of the HEYMon framework is generic and only serves as an execution platform for job types. To add new functionality to HEYMon, a developer just needs to add a new job type class and configure a job to call it.
Job Types in HEYMon contain their own set of properties. Each job type class has a set of property requirements which are configured through the XML-based properties file of HEYMon.

Job execution pattern
The following pseudo code definitions provide the patterns used during the execution lifecycle of a HEYMon job:

- Jobs are configured when the HEYMon application is started.
- A job is started by creating an instance of a job class and executing it as a single worker thread.
- The job worker initializes all required resources and variables and then goes to sleep for the period specified by the scan interval property.
- When the scan interval time has passed, the job awakes and executes the logic in the Job Type class. The logic determines if there is a condition which requires notification or not.
- If the job determines a notification is required, the Job Type class generates the email subject and body and sends the email via SMTP.
- The job then goes back to sleep for the period specified by the scan interval property.

HEYMon Job Types

HEYMon contains a number of Job Types which handle the various logic required to determine if there is an alarm condition.
Each Job Type is outlined in the following section, including any properties or grammars specific to the job type.

ProcessWebViewJob

The ProcessWebViewJob is designed to make HTTP requests to a specific URL and validate whether the related HTTP server is listening and how fast it is responding to requests.
For web-based applications, this job allows a way to check that the related server is up and responding to requests in a timely fashion.

ProcessWebViewJob job type properties

The ProcessWebViewJob job type contains the following properties with the related definition:

- connectionTimeout: Specified in seconds, this value is used as a comparison of the time expected for an HTTP response. If the HTTP server does not respond within the connectionTimeout period, it is considered to be responding too slowly and a notification is created.
- internetVerificationSites: One or more public web sites with the http prefix, separated by commas, that should be checked if the ProcessWebViewJob job determines the web site specified by the connection property is down. The web sites in this list are loaded to determine if they are accessible so that HEYMon can distinguish between a web site down and the total loss of an internet connection.

ProcessWebViewJob job properties

When the ProcessWebViewJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the URL to be used to access/check the HTTP server.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between checking the HTTP server.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: At least one query is required. The query represents a string that should be present in the source of the HTTP response.
  A query can be configured with the [URL] mnemonic. This mnemonic will be replaced with the URL specified by the connection property.
- notifySubject: When an alarm is determined by the ProcessWebViewJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property

string. If the property string does not contain the variable it will still be used as the subject of the alert email.

- notifyBody: When an alarm is determined by the ProcessWebViewJob job type, the body of the email is created from this property. This property can contain two job variables. The first variable will be the name of the server being monitored, and the second variable will be the timeout value specified by the job type.
  To have the server name and timeout amount added to this property add two variables '%s' in the property string. If the property string does not contain any variables it will still be used as the subject of the alert email.

ProcessWebViewJob process

The ProcessWebViewJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Server Availability monitor">
    <description>Monitors the app server to send notifications when it is not up
           and/or responding to HTTP requests.</description>
    <connection>http_bc_url</connection>
    <scanIntervalSeconds>120</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
       <query1>[URL]?inFrame=mainframe</query1>
       <query2>link rel="SHORTCUT ICON" href="res/img/favicon.ico"></query2>
    </queries>
       <notifySubject>Server %s is not responding.</notifySubject>
       <notifyBody>The server %s is not responding within %s seconds.
           The LOGIN page is taking too long to come up, or cannot be reached.
       </notifyBody>
     <type name="ProcessWebViewJob">
         <connectionTimeout>10</connectionTimeout>
     </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to make an HTTP request to the URL specified by the `connection` property. The worker will wait for the amount of time specified by the `connectionTimeout` property.
If the server responds within the configured timeout period, the job worker will parse the HTTP response for each of the string values specified by the collection of `query` properties. The property `query.query1` will be parsed first, followed by `query.query2` if it exists, and so on. All of the query conditions must be found in the HTTP response or the job worker will assume there is an alert condition. If one of the query conditions contains the mnemonic `[URL]`, the mnemonic will be replaced with the value of the `connection` meta-property.
If the job worker determines all query conditions have been met then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

If the job worker encounters one of the following conditions a notification is made via email to the email addresses configured for the `destination` meta-property:

- The HTTP URL cannot be reached for any reason other than there is no internet connection.
- The URL is addressable, but there is no response from the server at all.
- There is no HTTP response from the server within the time specified by the connectionTimeout property.
- Any one of the strings specified in the `query` properties is not found in the HTTP response.

ProcessWebViewJob Grammars

The ProcessWebViewJob job contains a grammar which can be used in any of the job queries. The grammar is expressed as the mnemonic value `[URL]`. When the job is executed, the value of the mnemonic `[URL]` is replaced with the value of the server URL which has been configured with the `connection` meta property.

ProcessQueryJob

The ProcessQueryJob is designed to perform a SQL query against a database. The query is assumed to return rows only when there is an alert condition. If the query does not return any rows, then the application assumes there is no alert condition.
This job is designed to find status or process information in a database and report the related records.

ProcessQueryJob job type properties

The ProcessQueryJob job type contains the following properties with the related definition:

- dateFormat: The `dateFormat` property is used to express the date format used in a SQL query. The date format is needed for the grammar used in the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.
- alertNoRecords: If this property is set to a value of 'true' then an alert will be sent if this job query does not return any rows. The alert will provide the job title and subject with a description of 'No data to report'.
- sumColumns: This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a comma.
  A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

ProcessQueryJob job properties

When the ProcessQueryJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: One query is required, and only one query is supported. The query represents a SQL query that may return rows. If the query returns any rows, they are considered to be an alert condition and a notification is generated.
- notifySubject: When an alarm is determined by the ProcessQueryJob job type, the subject of the email is created from this property. This property can contain

one job variable, which is the name of the server being monitored as specified by the systemID property. To have the server name added to this property, add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.

- notifyBody: This property is not needed by this job type. If there is an alarm condition then the body of the notification email will contain the rows that resulted from the SQL query.
- type.dateFormat: The date format string used for the `$lastIntervalDate` and `$intervalDate` job type grammars.

ProcessQueryJob process

The ProcessQueryJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Batch Process monitor">
    <description> Monitors BillingCenter to produce notification when a
          batch process has fired off.</description>
    <connection> query_bc_connect_string </connection>
    <scanIntervalSeconds>1800</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
       <query1>select ph.StartDate, ph.CompleteDate, bt.NAME, ph.OpsPerformed,
           ph.FailureReason from bc_processhistory ph
           INNER JOIN bctl_batchprocesstype bt
           on ph.ProcessType = bt.ID
       </query1>
    </queries>
    <notifySubject> Batch process results from %s </notifySubject>
    <notifyBody> </notifyBody>
     <type name="ProcessQueryJob">
         <dateFormat>yyyy-MM-dd HH:mm:ss</dateFormat>
    </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to make a SQL query using the database connection string specified by the `connection` property.
If the query returns any rows, the job worker will invoke the class specified by the queryFilter property and pass in the rows. The query filter class will then read the rows from the query and format them for the body of the notification.
If the SQL query does not return any rows then there is no alert condition and no notification is sent. The job worker will then go to sleep and the entire process starts over again.

ProcessQueryJob Grammars

The ProcessQueryJob job utilizes the `dateFormat` property and a grammar in the `queries.query1` property to express the date format used in a SQL query. The date format is needed for the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.

The dateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used by the database being queried by the ProcessQueryJob job type. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.

The format of this grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The queries.query1 Grammar

This job property specifies the sql query or queries that should be run at each interval. The query can have one or more of the following grammar values included:

- `$lastIntervalDate`: Represents the datetime value of the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:03, then `$lastIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$priorIntervalDate`: Represents the datetime value of the job which ran before the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$priorIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$nextIntervalDate`: Represents the datetime value of the next job which will ran after the current interval. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$nextIntervalDate` is 12/12/2012 at 11:09.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$intervalDate`: Represents the datetime value of the current interval this job is running. The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.

The grammar for the job interval date is designed for use in a SQL query. If HEYMon is running a sql query at a regular interval, it may be necessary to limit the query to the same time interval as the job.

For example, if you wanted to run a SQL query every 5 minutes that sends an alert only when the query returns 1 or more rows, you may want to configure SQL similar to the following:

```
SELECT col1, col3, col4 FROM tableA WHERE hasRun=1 AND
LAST_UPDATE_DATE>='$lastIntervalDate'
```

When the job is executed, this query will be converted into the following string with the last interval date and time set as follows:

```
SELECT col1, col3, col4 FROM tableA WHERE hasRun=1 AND
LAST_UPDATE_DATE>='2013-03-25 14:23:00'
```

The alertNoRecords Grammar

This job property allows the configuration of an alert message when the database query returns no rows. Normally, when this job executes the configured SQL query and no rows are returned, the job does not provide any notification.
If this property is set to a value of ‘true’, then an alert will be sent if the job query does not return any rows. The alert will provide the job title and subject with a description of ‘No data to report’.

The sumColumns Grammar

This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a

comma.
A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

ProcessTimedQueryJob

The ProcessTimedQueryJob is designed to perform a SQL query against a database at a specific time of the day. The query is assumed to return rows when there is an alert condition. Inversely, if the query does not return rows when executed at the configured time, no alert notification is sent.
This job is designed to read information in a database and report the query results using the metadata and column names specified in the sql query.

ProcessTimedQueryJob job type properties

The ProcessTimedQueryJob job type contains the following properties with the related definition:

- runTime: This property is used to determine the time of day that the job should be run. The property must be expressed as the time in a 24-hour format, such as 14:27 for 2:47 PM.
- dateFormat: The `dateFormat` property is used to express the date format used in a SQL query. The date format is needed for the grammar used in the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.
- alertNoRecords: If this property is set to a value of 'true' then an alert will be sent if this job query does not return any rows. The alert will provide the job title and subject with a description of 'No data to report'.
- sumColumns: This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a comma.
  A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

ProcessTimedQueryJob job properties

When the ProcessTimedQueryJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.

- queries.query1: One query is required, and only one query is supported. The query represents a SQL query that may return rows. If the query returns any rows, they are considered to be an alert condition and a notification is generated.
- notifySubject: When an alarm is determined by the StatisticTimedQueryJob job type, the subject of the email is created from this property.
- notifyBody: When an alert condition is determined by the StatisticTimedQueryJob this property dictates what the body of the alert should be. This property must contain one job variable which is the actual statistic value. To have the statistic value added to this property, add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email and the statistical value being queried will not be in the notification.
  It is recommended to always have the '%s' variable as the minimum value for this property.
- type.dateFormat: The date format string used for the `$lastIntervalDate` and `$intervalDate` job type grammars.

ProcessTimedQueryJob process

The ProcessTimedQueryJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="Policies in pending status daily report">
      <description>Performs a SQL query to determine the number of policies
      in a pending status the end of each day.</description>
      <connection>query_twiatdw_connect_string</connection>
       <systemID>twiatdw_db</systemID>
       <scanIntervalSeconds>120</scanIntervalSeconds>
       <destination>email_notify_bc</destination>
       <queries>
          <query1>select policy_number, policy_version, policy_state
                 from PR_POLICY_DESCRIPTION
                 WITH (NOLOCK) where STATUS = '%pend%'
                 and EXPIRATION_DATE > '$intervalDate'
          </query1>
       </queries>
       <notifySubject>Policies in pending status daily report</notifySubject>
       <notifyBody>%s</notifyBody>
       <type name="ProcessTimedQueryJob">
           <dateFormat>yyyy-MM-dd HH:mm:ss</dateFormat>
           <runTime>10:39</runTime>
       </type>
   </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to make a SQL query using the database connection string specified by the `connection` property.
If the query returns any rows, a query filter class will then read the rows from the query and format them for the body of the notification.

If the SQL query does not return any rows then there is no alert condition and no notification is sent. The job worker will then go to sleep and the entire process starts over again.

ProcessTimedQueryJob Grammars

The ProcessTimedQueryJob job utilizes the `dateFormat` property and a grammar in the `queries.query1` property to express the date format used in a SQL query. The date format is needed for the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.

The dateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used by the database being queried by the ProcessTimedQueryJob job type. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.

The format of this grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The runTime Grammar

This job property specifies the time of the day when this statistic query should be executed.
The property should be set to a standard time value in 24-hour format, delimited by a colon:

```
14:27
```

The `runTime` job property determines the time when the sql is executed. This differs from other job types in that the work is not performed on each job interval.

The alertNoRecords Grammar

This job property allows the configuration of an alert message when the database query returns no rows. Normally, when this job executes the configured SQL query and no rows are returned, the job does not provide any notification.
If this property is set to a value of 'true', then an alert will be sent if the job query does not return any rows. The alert will provide the job title and subject with a description of 'No data to report'.

The sumColumns Grammar

This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a comma.
A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

The queries.query1 Grammar

This job property specifies the sql query or queries that should be run at each interval. The query can have one of the following grammar value included if needed:

- `$lastIntervalDate`: Represents the datetime value of the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:03, then `$lastIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$priorIntervalDate`: Represents the datetime value of the job which ran before the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$priorIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$nextIntervalDate`: Represents the datetime value of the next job which will ran after the current interval. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$nextIntervalDate` is 12/12/2012 at 11:09.

The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.

- `$intervalDate`: Represents the datetime value of the current interval this job is running. The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.

The grammar for the job interval date is designed for use in a SQL query. If HEYMon is running a sql query at a regular interval, it may be necessary to limit the query to the same time interval as the job.

For the ProcessTimedQueryJob it is recommended to use only the `$intervalDate` value in SQL queries. When the job is executed, the configured query value will be converted to include the current date and time.

ProcessDayOfWeekQueryJob

The ProcessDayOfWeekQueryJob is designed to perform a SQL query against a database at a specific time of the day. The query is assumed to return rows when there is an alert condition. Inversely, if the query does not return rows when executed at the configured time, no alert notification is sent.
This job is designed to read information in a database and report the query results using the metadata and column names specified in the sql query.

ProcessDayOfWeekQueryJob job type properties

The ProcessDayOfWeekQueryJob job type contains the following properties with the related definition:

- runTime: This property is used to determine the time of day that the job should be run. When it is the time specified by this property and the day of the week defined by the dayOfWeek property, this job will attempt to execute the configured query.
  The property must be expressed as the time in a 24-hour format, such as 14:27 for 2:47 PM.
- dayOfWeek: The property that dictates which day of the week the job should be run. The property should be set to the day of the week, with a value like 'monday', 'thursday', and so on.
- dateFormat: The `dateFormat` property is used to express the date format used in a SQL query. The date format is needed for the grammar used in the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.
- alertNoRecords: If this property is set to a value of ‘true’ then an alert will be sent if this job query does not return any rows. The alert will provide the job title and subject with a description of ‘No data to report’.
- sumColumns: This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a comma.
  A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

ProcessDayOfWeekQueryJob job properties

When the ProcessDayOfWeekQueryJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.

- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: One query is required, and only one query is supported. The query represents a SQL query that may return rows. If the query returns any rows, they are considered to be an alert condition and a notification is generated.
- notifySubject: When an alarm is determined by the ProcessDayOfWeekQueryJob job type, the subject of the email is created from this property.
- notifyBody: When an alert condition is determined by the ProcessDayOfWeekQueryJob this property dictates what the body of the alert should be. This property must contain one job variable which is the actual statistic value. To have the statistic value added to this property, add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email and the statistical value being queried will not be in the notification.
  It is recommended to always have the '%s' variable as the minimum value for this property.
- type.dateFormat: The date format string used for the `$lastIntervalDate` and `$intervalDate` job type grammars.

ProcessDayOfWeekQueryJob process

The ProcessDayOfWeekQueryJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="Policies in pending status daily report">
      <description>Performs a SQL query to determine the number of policies
      in a pending status the end of each day.</description>
      <connection>query_twiatdw_connect_string</connection>
       <systemID>twiatdw_db</systemID>
       <scanIntervalSeconds>120</scanIntervalSeconds>
       <destination>email_notify_bc</destination>
       <queries>
          <query1>select policy_number, policy_version, policy_state
                 from PR_POLICY_DESCRIPTION
                 WITH (NOLOCK) where STATUS = '%pend%'
                 and EXPIRATION_DATE > '$intervalDate'
          </query1>
       </queries>
       <notifySubject>Policies in pending status daily report</notifySubject>
       <notifyBody>%s</notifyBody>
       <type name="ProcessDayOfWeekQueryJob">
           <dateFormat>yyyy-MM-dd HH:mm:ss</dateFormat>
          <runTime>10:39</runTime>
           <dayOfWeek>Friday</dayOfWeek>
       </type>
   </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to make a SQL query using the database connection string specified by the `connection` property.
If the query returns any rows, a query filter class will then read the rows from the query and format them for the body of the notification.
If the SQL query does not return any rows then there is no alert condition and no notification is sent. The job worker will then go to sleep and the entire process starts over again.

ProcessDayOfWeekQueryJob Grammars

The ProcessDayOfWeekQueryJob job utilizes the `dateFormat` property and a grammar in the `queries.query1` property to express the date format used in a SQL query. The date format is needed for the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.

The dateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used by the database being queried by the ProcessDayOfWeekQueryJob job type. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.

The format of this grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30

s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The runTime Grammar

This job property specifies the time of the day when this statistic query should be executed.
The property should be set to a standard time value in 24-hour format, delimited by a colon:

```
14:27
```

The `runTime` job property determines the time when the sql is executed. This differs from other job types in that the work is not performed on each job interval.

The dayOfWeek Grammar

The property that dictates which day of the week the job should be run. When it is the time specified by the `runTime` property and the day of the week defined by the `dayOfWeek` property, this job will attempt to execute the configured query.

The alertNoRecords Grammar

This job property allows the configuration of an alert message when the database query returns no rows. Normally, when this job executes the configured SQL query and no rows are returned, the job does not provide any notification.
If this property is set to a value of ‘true’, then an alert will be sent if the job query does not return any rows. The alert will provide the job title and subject with a description of ‘No data to report’.

The sumColumns Grammar

This property is used to express if any of the columns in the SQL query result should be summed at the end of the alert notification. Set this property to one of the column names in the SQL query. Multiple column names can be defined separated by a comma.
A total can be calculated for numeric data only. If one or more specified columns cannot be evaluated as a number then the sumColumn will not have a value on the alert.

The queries.query1 Grammar

This job property specifies the sql query or queries that should be run at each interval. The query can have one of the following grammar value included if needed:

- `$lastIntervalDate`: Represents the datetime value of the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:03, then `$lastIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$priorIntervalDate`: Represents the datetime value of the job which ran before the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$priorIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$nextIntervalDate`: Represents the datetime value of the next job which will ran after the current interval. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$nextIntervalDate` is 12/12/2012 at 11:09.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$intervalDate`: Represents the datetime value of the current interval this job is running. The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.

The grammar for the job interval date is designed for use in a SQL query. If HEYMon is running a sql query at a regular interval, it may be necessary to limit the query to the same time interval as the job.

For the ProcessDayOfWeekQueryJob it is recommended to use only the `$intervalDate` value in SQL queries. When the job is executed, the configured query value will be converted to include the current date and time.

StatisticTimedQueryJob

The StatisticTimedQueryJob is designed to perform a SQL query against a database at a specific time of the day. The query is assumed to return one row that contains a statistical value. If the query does not return any rows, then the application assumes there is no statistic and therefore no notification condition.
This job is designed to read information in a database and report the count, totals, or other singular statistical values.

StatisticTimedQueryJob job type properties

The StatisticTimedQueryJob job type contains the following properties with the related definition:

- runTime: This property is used to determine the time of day that the job should be run. The property must be expressed as the time in a 24-hour format, such as 14:27 for 2:47 PM.
- dateFormat: The `dateFormat` property is used to express the date format used in a SQL query. The date format is needed for the grammar used in the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.

StatisticTimedQueryJob job properties

When the StatisticTimedQueryJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: One query is required, and only one query is supported. The query represents a SQL query that may return rows. If the query returns any rows, they are considered to be an alert condition and a notification is generated.
- notifySubject: When an alarm is determined by the StatisticTimedQueryJob job type, the subject of the email is created from this property.
- notifyBody: When an alert condition is determined by the StatisticTimedQueryJob this property dictates what the body of the alert should be. This property must contain one job variable which is the actual statistic value. To have the statistic value added to this property, add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email and the statistical value being queried will not be in the

notification.
It is recommended to always have the '%s' variable as the minimum value for this property.

- type.dateFormat: The date format string used for the `$lastIntervalDate` and `$intervalDate` job type grammars.

StatisticTimedQueryJob process

The StatisticTimedQueryJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
   <job name="Daily Policies In-Force count">
         <description>Performs a SQL query to determine the number of policies in
force at the end of each day.</description>
         <connection>query_twiatdw_connect_string</connection>
          <systemID>twiatdw_db</systemID>
          <scanIntervalSeconds>120</scanIntervalSeconds>
          <destination>email_notify_bc</destination>
          <queries>
             <query1>select count(distinct POLICY_ID) from PR_POLICY_DESCRIPTION
                    WITH (NOLOCK) where STATUS = 'Issued'
                    and EXPIRATION_DATE > '$intervalDate'
             </query1>
          </queries>
          <notifySubject>Daily Policies In-Force Count</notifySubject>
          <notifyBody>%s</notifyBody>
          <type name="StatisticTimedQueryJob">
              <dateFormat>yyyy-MM-dd HH:mm:ss</dateFormat>
             <runTime>10:39</runTime>
          </type>
      </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to make a SQL query using the database connection string specified by the `connection` property.
If the query returns any rows, the job worker will invoke the class specified by the queryFilter property and pass in the rows. The query filter class will then read the rows from the query and format them for the body of the notification.
If the SQL query does not return any rows then there is no alert condition and no notification is sent. The job worker will then go to sleep and the entire process starts over again.

StatisticTimedQueryJob Grammars

The StatisticTimedQueryJob job utilizes the `dateFormat` property and a grammar in the `queries.query1` property to express the date format used in a SQL query. The date format is needed for the job query to specify if the query should be run with a date in the current interval, or a date in the last interval.

The dateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used by the database being queried by the StatisticTimedQueryJob job type. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.

The format of this grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The runTime Grammar

This job property specifies the time of the day when this statistic query should be executed.
The property should be set to a standard time value in 24-hour format, delimited by a colon:

```
14:27
```

The `runTime` job property determines the time when the sql is executed. This differs from other job types in that the work is not performed on each job interval.

The queries.query1 Grammar

This job property specifies the sql query or queries that should be run at each interval. The query can have one of the following grammar value included if needed:

- `$lastIntervalDate`: Represents the datetime value of the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:03, then `$lastIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$priorIntervalDate`: Represents the datetime value of the job which ran before the last interval this job was executed. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$priorIntervalDate` is 12/12/2012 at 11:00.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$nextIntervalDate`: Represents the datetime value of the next job which will ran after the current interval. If a HEYMon job runs every three minutes and it is 12/12/2012 at 11:06, then `$nextIntervalDate` is 12/12/2012 at 11:09.
  The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.
- `$intervalDate`: Represents the datetime value of the current interval this job is running. The datetime value is formatted in the SQL using the format specified by the `dateFormat` job type property.

The grammar for the job interval date is designed for use in a SQL query. If HEYMon is running a sql query at a regular interval, it may be necessary to limit the query to the same time interval as the job.

For the StatisticTimedQueryJob it is recommended to use only the `$intervalDate` value in SQL queries. When the job is executed, the configured query value will be converted to include the current date and time.

ProcessErrorLogJob

The ProcessErrorLogJob is designed to parse text files and find patterns which could indicate an alert condition.
For applications which create log files, this job allows a way to find process status and/or error conditions in the logs and report those as alert conditions.

ProcessErrorLogJob job type properties

The ProcessErrorLogJob job type contains the following properties with the related definition:

- logTempDirectory: The meta-property that determines the directory where any log files will be copied before parsing. To minimize impact on the system being monitored, HEYMon will copy log files to a local temporary directory before parsing the file for alert conditions.
- logLineKey: A grammar that allows a key value to be calculated for each line in the log file. A key value is used to identify log file lines which have already been parsed. When HEYMon determines a line in the log contains an alert condition it calculates a key value for the line and checks if an alert has already been sent for that key.
  In most log files, the key should be the date/time stamp and some part of the log data to ensure that each line appears unique.
- logLineRead: A numeric value specifying the number of lines to read from the log when there is an alert condition. In many logs, errors are output with numerous lines of detail. The logLineRead property allows a way to capture multiple lines of data which are after the line with the alert condition so that this data can be sent out in the body of the alert notification.
- logLineDateFormat: A date formatting value that expresses the way dates are written to the log file. This value is used by HEYMon to locate log data which is current to the time and date. This value allows HEYMon to only report errors within the last time interval.
- keyDelimiter: The delimiter character or characters utilized for the positional expressions in a job query. If the keyDelimiter is set to a comma (,) and the job query has the expression `$2=value`, then `$2` relates to the string '`goo`' in the value '`foo,goo,you,too`'.
- queryFilter1: A string containing values that may be in an error message. When the ProcessErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.
- queryFilterSubject1: A string containing the subject to use in alerts if a specific error condition is found by matching the string in queryFilter1. When the ProcessErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.

ProcessErrorLogJob job properties

When the ProcessErrorLogJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: Only one query is required. The query represents a grammar that expresses the string value in the line of a log file which indicates an alert condition.
- notifySubject: When an alarm is determined by the ProcessErrorLogJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used by the ProcessErrorLogJob job. When an alert condition is found, the body of the notification will contain the lines in the log file.

ProcessErrorLogJob process

The ProcessErrorLogJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Error monitor">
    <description>Monitors the BillingCenter log files to send a notification
       when an error has occured</description>
    <connection>file_bc_app_log</connection>
    <systemID>bc_server</systemID>
    <scanIntervalSeconds>300</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
       <query1>$4=ERROR</query1>
    </queries>
    <notifySubject>Errors found in BC log file from %s</notifySubject>
    <notifyBody> </notifyBody>
    <type name="ProcessErrorLogJob">
       <logTempDirectory>file_tmp_dir</logTempDirectory>
       <logLineKey>$1$2$3$4</logLineKey>
       <logLineRead>5</logLineRead>
        <keyDelimiter> </keyDelimiter>
       <logLineDateFormat>yyyy-MM-dd HH</logLineDateFormat>
```

```
    </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to copy the log file specified by the `connection` property to the location specified by the `logTempDirectory` job type property.
If the file can be found and copied to the logTempDirectory, the job worker will use the format of the logLineDateFormat property to find log entries which have occurred since the last scan interval. Next, the job worker will parse the log file for the string value specified by the grammar in the `query.query1` property. If the log file contains the specified string, the job worker will assume there is an alert condition.
When there is an alert condition, the number of lines specified by the `logLineRead` job type property are sent in the body of the notification.
If the job worker determines the string is not present in the log file then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessErrorLogJob Grammars

The ProcessErrorLogJob job utilizes the `logLineKey` and the `queries.query1` properties to express positional parameters in a space-delimited string. The grammar is the same as utilized by UNIX shell parameters, as well as shell tools such as sed and awk.
The grammar is based on the concept of a line in a text file. The line may have one or more spaces which represent different parts of the text content. An example log file line is as follows:

```
2012-04-12 07:58:16,453 ERROR Increasing runlevel to 'GUIDEWIRE_STARTUP'
```

In the log file line there are 6 places where space characters delimit parts of the text.
To refer to the first section of text in the line, the value `$1` should be used. To refer to the second section of text, the value `$2` should be used, and so on.
The `logLineKey` job type property should contain values that represent which sections of the log file line should be used as a unique key.
The `queries.query1` property should contain values that represent how to determine if a log file line contains an alert condition.

The logLineKey grammar

A key value is used to identify log file lines which have already been parsed. When HEYMon determines a line in the log contains an alert condition it calculates a key value for the line and checks if an alert has already been sent for that key.
Using the aforementioned example, we would want the first three sections of the line as a key since they will always be unique. Specifying the `logLineKey` property as `$1$2$3` would be the same as creating this string as the log line key:

```
2012-04-1207:58:16,453ERROR
```

The key value is not used for an alert condition and is not part of the alert notification. The key is used internally by HEYMon to ensure that a log-based alert notification is not sent out more than one time.

The keyDelimiter grammar

The keyDelimiter property does not have a specific grammar. It is used as a property that dictates the delimiter character or characters utilized for positional expressions in a job query. If the keyDelimiter is set to a comma (,) and the job query has the expression `$2=value`, then `$2` relates to the string '`goo`' in the value '`foo,goo,you,too`'.

The logLineRead grammar

A numeric value is used for this property to represent the number of log lines to show in alerts. When HEYMon determines there is an alert condition in a log file, it will also read the number of lines specified by this property and return that data with the alert.
The value of logLineRead must be an integer value which is at least 1. It is not recommended to use a large value for this property.

The queries.query1 grammar

A value is required to indicate when the content of a log line contains an alert condition. As a log file is read, each line in the log is checked for the value expressed by the `queries.query1` property. The grammar for this property allows the following types of evaluations:

- equals: Does a string in the log file line equal a value in the property?
- contains: Does a string in the log file line contain a value in the property?

As an example, we may want to find entries in the following log sample that have the string '`ERROR`':

```
2012-04-12 07:58:16,453 ERROR Increasing runlevel to 'GUIDEWIRE_STARTUP'
```

In the example the string indicating an error is at position 3. To write the grammar for the `queries.query1` property to find similar log lines we would set the property to this:

```
$3=ERROR
```

It may also be necessary to find a string which is part of another string. As an example, we may want to find entries in the following log sample that have the string '`[ERROR]`':

```
[ERROR]@[30 Jan 2012 00:04:27,031]: Storing JMSQueueStatistic
```

Since the string is part of another string in position 1, the grammar for the `queries.query1` property to find similar log lines would be as follows:

```
$1contains([ERROR])
```

In this case, `$1` represents the string '`[ERROR]@[30`' and we want to know if the string '`[ERROR]`' is anywhere within `$1`.

The logLineDateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used in the log file being monitored. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.
The format of the grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The keyDelimiter grammar

This grammar is used to specify the delimiter used by the queries.query1 job property. When the queries.query1 job property specifies a positional index (like $4 for the fourth position in a line), it is based on the delimiter specified by the keyDelimiter property. For example, if the ProcessErrorLogJob was configured to read the third item in a log line entry based on commas, then the keyDelimiter property would be set to the value ‘,’.

StatisticErrorLogJob

The StatisticErrorLogJob is designed to parse text files and find patterns which can be utilized as a statistic.
For applications which create log files, this job allows a way to find statistical data in the logs and report those as notifications.
The StatisticErrorLogJob differs from the ProcessErrorLogJob in that it will process each statistic as a separate notification condition. For example, if the StatisticErrorLogJob finds two statistic values in a log file it will send out two notifications. This is done so that there is one statistic per notification.

StatisticErrorLogJob job type properties

The StatisticErrorLogJob job type contains the following properties with the related definition:

- logTempDirectory: The meta-property that determines the directory where any log files will be copied before parsing. To minimize impact on the system being monitored, HEYMon will copy log files to a local temporary directory before parsing the file for alert conditions.
- logLineKey: A grammar that allows a key value to be calculated for each line in the log file. A key value is used to identify log file lines which have already been parsed. When HEYMon determines a line in the log contains an alert condition it calculates a key value for the line and checks if an alert has already been sent for that key.
  In most log files, the key should be the date/time stamp and some part of the log data to ensure that each line appears unique.
- logLineDateFormat: A date formatting value that expresses the way dates are written to the log file. This value is used by HEYMon to locate log data which is current to the time and date. This value allows HEYMon to only report errors within the last time interval.
- queryFilter1: A string containing values that may be in an error message. When the StatisticErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.
- queryFilterSubject1: A string containing the subject to use in alerts if a specific error condition is found by matching the string in queryFilter1. When the StatisticErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.

StatisticErrorLogJob job properties

When the StatisticErrorLogJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: Only one query is required. The query represents a grammar that expresses the string value in the line of a log file which indicates an alert condition.
- notifySubject: When an alarm is determined by the StatisticErrorLogJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used by the StatisticErrorLogJob job. When an alert condition is found, the body of the notification will contain the lines in the log file.

StatisticErrorLogJob process

The StatisticErrorLogJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Error monitor">
    <description>Monitors the BillingCenter log files to send a notification
       when an error has occured</description>
    <connection>file_bc_app_log</connection>
    <systemID>bc_server</systemID>
    <scanIntervalSeconds>300</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
       <query1>$4=PolicyIssued</query1>
    </queries>
    <notifySubject>Statistics found in BC log file from %s</notifySubject>
    <notifyBody>%s</notifyBody>
    <type name="StatisticErrorLogJob">
       <logTempDirectory>file_tmp_dir</logTempDirectory>
       <logLineKey>$1$2$3$4</logLineKey>
        <keyDelimiter> </keyDelimiter>
       <logLineDateFormat>yyyy-MM-dd HH</logLineDateFormat>
    </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to copy the log file specified by the `connection` property to the location specified by the `logTempDirectory` job type property. If the file can be found and copied to the logTempDirectory, the job worker will use the format of the logLineDateFormat property to find log entries which have occurred since the last scan interval. Next, the job worker will parse the log file for the string value specified by the grammar in the `query.query1` property. If the log file contains the specified string, the job worker will assume there is a statistic notification condition.
If the job worker determines the string is not present in the log file then there is notification. The job worker will then go to sleep and the entire process starts over again.

StatisticErrorLogJob Grammars

The StatisticErrorLogJob job utilizes the `logLineKey` and the `queries.query1` properties to express positional parameters in a space-delimited string. The grammar is the same as utilized by UNIX shell parameters, as well as shell tools such as sed and awk.
The grammar is based on the concept of a line in a text file. The line may have one or more spaces which represent different parts of the text content. An example log file line is as follows:

```
2012-04-12 07:58:16,453 INFO Policy 3423429 Issued
```

In the log file line there are 6 places where space characters delimit parts of the text.
To refer to the first section of text in the line, the value `$1` should be used. To refer to the second section of text, the value `$2` should be used, and so on.
The `logLineKey` job type property should contain values that represent which sections of the log file line should be used as a unique key.
The `queries.query1` property should contain values that represent how to determine if a log file line contains an alert condition.

The logLineKey grammar

A key value is used to identify log file lines which have already been parsed. When HEYMon determines a line in the log contains a notification condition it calculates a key value for the line and checks if an alert has already been sent for that key.
Using the aforementioned example, we would want the first three sections of the line as a key since they will always be unique. Specifying the `logLineKey` property as `$1$2$3` would be the same as creating this string as the log line key:

```
2012-04-1207:58:16,453ERROR
```

The key value is not used for an alert condition and is not part of the alert notification. The key is used internally by HEYMon to ensure that a log-based alert notification is not sent out more than one time.

The queries.query1 grammar

A value is required to indicate when the content of a log line contains an alert condition. As a log file is read, each line in the log is checked for the value expressed by the `queries.query1` property. The grammar for this property allows the following types of evaluations:

- equals: Does a string in the log file line equal a value in the property?
- contains: Does a string in the log file line contain a value in the property?

As an example, we may want to find entries in the following log sample that have the string '`ERROR`':

```
2012-04-12 07:58:16,453 ERROR Increasing runlevel to 'GUIDEWIRE_STARTUP'
```

In the example the string indicating an error is at position 3. To write the grammar for the `queries.query1` property to find similar log lines we would set the property to this:

```
$3=ERROR
```

It may also be necessary to find a string which is part of another string. As an example, we may want to find entries in the following log sample that have the string '`[ERROR]`':

```
[ERROR]@[30 Jan 2012 00:04:27,031]: Storing JMSQueueStatistic
```

Since the string is part of another string in position 1, the grammar for the `queries.query1` property to find similar log lines would be as follows:

```
$1contains([ERROR])
```

In this case, `$1` represents the string '`[ERROR]@[30`' and we want to know if the string '`[ERROR]`' is anywhere within `$1`.

The logLineDateFormat grammar

This grammar is used to allow HEYMon to know the date/time format used in the log file being monitored. HEYMon will calculate the date/time in which it last ran, and use that value as a starting point in the log file to search for alert conditions. This is done so that HEYMon will always be looking for alarms in the current timeframe, and not reporting issues which were already reported and/or occurred in the past.
The format of the grammar follows that of the Java SimpleDateFormat, and can be described as follows:

Letter:	Date/Time component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189

d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

For more details and examples of this grammar, see Appendix A at the end of this document.

The keyDelimiter grammar

This grammar is used to specify the delimiter used by the queries.query1 job property. When the queries.query1 job property specifies a positional index (like $4 for the fourth position in a line), it is based on the delimiter specified by the keyDelimiter property. For example, if the StatisticErrorLogJob was configured to read the third item in a log line entry based on commas, then the keyDelimiter property would be set to the value ‘,’.

ProcessSequentialErrorLogJob

The ProcessSequentialErrorLogJob is designed to parse text files and find patterns which could indicate an alert condition.
For applications which create log files which do not contain date stamps, this job allows a way to find process status and/or error conditions in the logs and report those as alert conditions.

ProcessSequentialErrorLogJob job type properties

The ProcessSequentialErrorLogJob job type contains the following properties with the related definition:

- logTempDirectory: The meta-property that determines the directory where any log files will be copied before parsing. To minimize impact on the system being monitored, HEYMon will copy log files to a local temporary directory before parsing the file for alert conditions.
- logLineRead: A numeric value specifying the number of lines to read from the log when there is an alert condition. In many logs, errors are output with numerous lines of detail. The logLineRead property allows a way to capture multiple lines of data which are after the line with the alert condition so that this data can be sent out in the body of the alert notification.
- logLineStart: A numeric value specifying the line number in the log to start reading from after a restart of HEYMon. This is an optional property that can be used to reset where HEYMon should read in a log file. Since this job type is designed to read log file data that does not have dates, after restarting HEYMon it may be necessary to configure the line number in the log file to start reading from.
  If the value of logLineStart is a line number that does not exist in the log file, then HEYMon will seek back to the beginning of the log and look for errors from there.
- queryFilter1: A string containing values that may be in an error message. When the ProcessSequentialErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.
- queryFilterSubject1: A string containing the subject to use in alerts if a specific error condition is found by matching the string in queryFilter1. When the ProcessSequentialErrorLogJob job type finds an alert condition it will check through all the queryFilterX properties to see if they contain values in the error. If the values are contained in the error, then HEYMon assumes this is a specific error type and will use the property queryFilterSubjectX as the subject of the emailed alert.
  To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.

ProcessSequentialErrorLogJob job properties

When the ProcessSequentialErrorLogJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: Only one query is required. The query represents a grammar that expresses the string value in the line of a log file which indicates an alert condition.
- notifySubject: When an alarm is determined by the ProcessSequentialErrorLogJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used by the ProcessSequentialErrorLogJob job. When an alert condition is found, the body of the notification will contain the lines in the log file.

ProcessSequentialErrorLogJob process

The ProcessSequentialErrorLogJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Error monitor">
    <description>Monitors the BillingCenter log files to send a notification
       when an error has occured</description>
    <connection>file_bc_app_log</connection>
    <systemID>bc_server</systemID>
    <scanIntervalSeconds>300</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
       <query1>$4=ERROR</query1>
    </queries>
    <notifySubject>Errors found in BC log file from %s</notifySubject>
    <notifyBody> </notifyBody>
    <type name="ProcessSequentialErrorLogJob">
       <logTempDirectory>file_tmp_dir</logTempDirectory>
       <logLineRead>5</logLineRead>
        <logLineStart>32001</logLineStart>
    </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will attempt to copy the log file specified by the `connection` property to the location specified by the `logTempDirectory` job type property.
If the file can be found and copied to the `logTempDirectory`, the job worker will use the `logLineRead` property to find the line number to start reading from. Next, the job worker will parse the log file for the string value specified by the grammar in the `query.query1` property. If the log file contains the specified string, the job worker will assume there is an alert condition.
When there is an alert condition, the number of lines specified by the `logLineRead` job type property are sent in the body of the notification.
If the job worker determines the string is not present in the log file then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessSequentialErrorLogJob Grammars

The ProcessSequentialErrorLogJob job utilizes the `queries.query1` property to express positional parameters in a space-delimited string. The grammar is the same as utilized by UNIX shell parameters, as well as shell tools such as sed and awk.
The grammar is based on the concept of a line in a text file. The line may have one or more spaces which represent different parts of the text content. An example log file line is as follows:

```
07:58:16,453 ERROR Increasing runlevel to 'GUIDEWIRE_STARTUP'
```

In the log file line there are 5 places where space characters delimit parts of the text.
To refer to the first section of text in the line the value `$1` should be used. To refer to the second section of text (*the word ERROR*), the value `$2` should be used, and so on.
The `queries.query1` property should contain values that represent how to determine if a log file line contains an alert condition.

The logLineStart grammar

A key value is used to identify the line number to start reading from after HEYMon has been restarted. This is required for the ProcessSequentialErrorLogJob since the job should be used for reading log files that do not have date stamp. HEYMon keeps track of which line numbers it has read to prevent sending duplicate error notifications.

The logLineRead grammar

A numeric value is used for this property to represent the number of log lines to show in alerts. When HEYMon determines there is an alert condition in a log file, it will also read the number of lines specified by this property and return that data with the alert.

The value of logLineRead must be an integer value which is at least 1. It is not recommended to use a large value for this property.

The queries.query1 grammar

A value is required to indicate when the content of a log line contains an alert condition. As a log file is read, each line in the log is checked for the value expressed by the `queries.query1` property. The grammar for this property allows the following types of evaluations:

- equals: Does a string in the log file line equal a value in the property?
- contains: Does a string in the log file line contain a value in the property?

As an example, we may want to find entries in the following log sample that have the string '`ERROR`':

```
07:58:16,453 ERROR Increasing runlevel to 'GUIDEWIRE_STARTUP'
```

In the example the string indicating an error is at position 2. To write the grammar for the `queries.query1` property to find similar log lines we would set the property to this:

```
$2=ERROR
```

It may also be necessary to find a string which is part of another string. As an example, we may want to find entries in the following log sample that have the string '`[ERROR]`':

```
[ERROR]@[00:04:27,031]: Storing JMSQueueStatistic
```

Since the string is part of another string in position 1, the grammar for the `queries.query1` property to find similar log lines would be as follows:

```
$1contains([ERROR])
```

In this case, `$1` represents the string '`[ERROR]@[00:04:27,031]:`' and we want to know if the string '`[ERROR]`' is anywhere within `$1`.

FoundInIntervalJob

The FoundInIntervalJob is designed run SQL queries that represent work items within an application. For example, Guidewire applications and many modern systems contain message queues that are backed by a database. The message queues can be queried for a record count, and if the record count does not change or decrease over time then an alert condition can be created.
The primary use of the FoundInIntervalJob is to execute a SQL query over multiple time intervals to determine if data has changed. Using the prior example, if the number of rows returned from the message queue tables does not decrease over time, or continues to rise, then an alert condition is created.

FoundInIntervalJob job type properties

The FoundInIntervalJob job type contains the following properties with the related definition:

- numberOfIntervals: The meta-property that determines the number of intervals that the SQL query returns rows. If the numberOfIntervals property was set to 3, then if the sql query for this job returns rows 3 times, HEYMon will assume there is an alert condition.
- queryCondition: A grammar that allows configuration of how to evaluate the result of the SQL query in each interval. The queryCondition property can be used to check the result of a SQL query for no rows, multiple rows, and even values in each row.
  If the queryCondition is found to be true in each of the intervals, then HEYMon will assume there is an alert condition.

FoundInIntervalJob job properties

When the FoundInIntervalJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the JDBC database connection string.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the configured SQL query.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: Only one query is required. The query represents a grammar that expresses the string value in the line of a log file which indicates an alert condition.
- notifySubject: When an alarm is determined by the FoundInIntervalJob job type, the subject of the email is created from this property. This property can contain

one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.

- notifyBody: When an alarm is determined by the FoundInIntervalJob job type, the body of the email is created from this property. This property can contain two job variables. The first variable will be the result of the SQL query, and the second variable will be the total number of minutes for the job to run.
  To have the query result and duration value added to this property, add two variables '%s' in the property string. If the property string does not contain any variables it will still be used as the subject of the alert email.

FoundInIntervalJob process

The FoundInIntervalJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="BC Event Message monitor">
   <description>Monitors the BillingCenter Event Message queue and notifies
     when there are messages in the queue.</description>
   <connection>query_bc_connect_string</connection>
   <systemID>bc_server</systemID>
   <scanIntervalSeconds>60</scanIntervalSeconds>
   <destination>email_notify_bc</destination>
   <queries>
      <query1>select count(*) from bc_message</query1>
   </queries>
   <notifySubject>There are %s Event Messages in the queue.</notifySubject>
   <notifyBody>There are %s Event Messages in the queue.
      The messages showed up about %s minutes ago and have not been released
      as of the time on this email.</notifyBody>
   <type name="FoundInIntervalJob">
      <numberOfIntervals>3</numberOfIntervals>
      <queryCondition>notnull,$1!=0</queryCondition>
   </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will execute the SQL query specified by the `queries.query1` property. The results of the query are compared to the grammar in the `queryCondition` job property. If the query result matches the queryCondition property then HEYMon remembers this and goes to sleep until the next interval.
If the results of the sql query match the queryCondition in each consecutive interval, then HEYMon assumes there is an alert condition.
If the job worker determines the query result does not match the value of the `queryCondition` job property for each consecutive interval then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

FoundInIntervalJob Grammars

The FoundInIntervalJob job utilizes the `queryCondition` properties to evaluate if data returned by the SQL query matches a specific condition. The grammar is similar to that which is utilized by UNIX shell parameters, as well as shell tools such as sed and awk. The grammar is based on the concept of a sql resultset. A resultset can have two states which are supported by the following grammars:

- null: The SQL query returned no rows.
- notnull: The SQL query returned one or more rows.

A resultset can also be checked for some of its data. To check for a value in a column returned from the SQL query use the positional notation of $1 for the first column. For example, assume the job executes this SQL query:

```
SELECT firstName, secondName, title FROM myTable WHERE ID=4
```

If the query returned any rows, we could check the value of the title by using this grammar for the queryCondition job type property:

```
$3=Manager
```

This would match any rows where the third column has the value of 'Manager'. The grammar also supports the concept of inequality. We can use this grammar to ensure the records do not return any titles of Manager:

```
$3!=Manager
```

The queryCondition job property allows for combinations of grammars, separated by a comma. To check that the result of the query contained records and that one of the records matches the value 'dog' in the second column, the following grammar can be used:

```
notnull,$2=dog
```

ProcessWindowsServiceJob

The ProcessWindowsServiceJob is designed to check the running status of a Windows service.
The primary use of the ProcessWindowsServiceJob is to monitor that a Windows service is running, and to create an alert if the service stops or does not respond for any reason.
The ProcessWindowsServiceJob can only be used when the HEYMon engine is running in Windows. This job does not use an Agent to retrieve Windows Service status. Windows Services are queried across the network using the Service Control Manager, sc.exe.
For the best security, or if the HEYMon engine is not running on Windows, use ProcessSystemInfoJob with HEYMon Agent enabled to check Windows Services.

ProcessWindowsServiceJob job type properties

The ProcessWindowsServiceJob job type contains the following properties with the related definition:

- outputTempDirectory: The meta-property that determines the directory where any service status requests are redirected to a file. To be aware of the status when the job ran previously, the service status is stored in a text file.

ProcessWindowsServiceJob job properties

When the ProcessWindowsServiceJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the name of the Windows service to be monitored. The name of a Windows service is display in the list of Services in Control Panel, Administrative tools.
- systemID: The name of the server or application being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing the check for a running service.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: Queries are not used by the ProcessWindowsServiceJob job. The queries and query1 elements are still required in the job xml configuration file.
- notifySubject: When an alarm is determined by the ProcessWindowsServiceJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.

- notifyBody: When an alarm is determined by the ProcessWindowsServiceJob job type, the body of the email is created from this property. This property can contain two job variables. The first variable will be the result of the SQL query, and the second variable will be the total number of minutes for the job to run. To have the query result and duration value added to this property, add two variables '%s' in the property string. If the property string does not contain any variables it will still be used as the subject of the alert email.

ProcessWindowsServiceJob process

The ProcessWindowsServiceJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="Titas Windows service monitor">
    <description>Monitors the TITAS/JBoss Windows service to send a
     notification when the service is not running.</description>
    <connection>titas_service_name</connection>
    <systemID>titas_server</systemID>
    <scanIntervalSeconds>225</scanIntervalSeconds>
    <destination>email_notify_bc</destination>
    <queries>
      <query1> </query1>
    </queries>
    <notifySubject>TITAS/JBoss Windows service not running on %s</notifySubject>
    <notifyBody> </notifyBody>
    <type name="ProcessWindowsServiceJob">
      <outputTempDirectory>file_tmp_dir</outputTempDirectory>
    </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will execute a process using sc.exe to request the status of the service. The results of the process are redirected to a temporary file and then parsed to determine the current state of the service.
If the service reports it is running then HEYMon remembers this and goes to sleep until the next interval.
If the service reports it is not running, or the service is not defined, then HEYMon assumes there is an alert condition and a notification is sent out.
The job worker will then go to sleep and the entire process starts over again.

ProcessWindowsServiceJob Grammars

The ProcessWindowsServiceJob job does not contain any grammars.

ProcessSystemInfoJob

The ProcessSystemInfoJob utilizes a software agent which is installed on each system being monitored by HEYMon. The agent will lookup the following server metrics:

- Available RAM
- Available Disk space
- Running Processes
- Running Windows Services

The primary use of the ProcessSystemInfoJob is to monitor server metrics to create alerts for certain conditions. The ProcessSystemInfoJob can also be used to gather server statistics over time.

ProcessSystemInfoJob job type properties

The ProcessSystemInfoJob job type contains the following properties with the related definition:

- logLineRead: A numeric value specifying the number of lines to read from the log when there is an alert condition. In many logs, errors are output with numerous lines of detail. The logLineRead property allows a way to capture multiple lines of data which are after the line with the alert condition so that this data can be sent out in the body of the alert notification.
- keyDelimiter: The delimiter character or characters utilized for the positional expressions in a job query. If the keyDelimiter is set to a comma (,) and the job query has the expression `$2=value`, then `$2` relates to the string '`goo`' in the value '`foo,goo,you,too`'.

ProcessSystemInfoJob job properties

When the ProcessSystemInfoJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: The meta-property that refers to the port number that HEYMonAgent is using. If improperly defined port 7169 is used.
- systemID: The name of the server or servers being monitored. Multiple server names must be separated by a comma.
  HEYMon will try to contact the HEYMonAgent on the server. Each server listed in the systemID property must have the HEYMonAgent installed and running. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between executing checks of the HEYMonAgents installed on the configured servers.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: The query represents a grammar that expresses which server metrics are being checked, and if they are configured as a statistic or as an alert condition.

- notifySubject: When an alarm is determined by the ProcessSystemInfoJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: When an alarm is determined by the ProcessSystemInfoJob job type, the first line in the body of the email alert is created from this property This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the body of the alert email.

ProcessSystemInfoJob process

The ProcessSystemInfoJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
   <job name="Server Metrics Monitor for DEV servers">
         <description>Reads stats for Memory, Disk Space, Processes, and Win
Services.</description>
         <connection>heymon_agent_port</connection>
          <systemID>heymon_agent_devservers</systemID>
          <scanIntervalSeconds>300</scanIntervalSeconds>
          <destination>email_notify_bc</destination>
          <queries>
             <query1>DISKSPACE:C<1024</query1>
          </queries>
          <notifySubject>Daily Policies In-Force Count</notifySubject>
          <notifyBody>%s</notifyBody>
          <type name="ProcessSystemInfoJob" useagent="true">
             <keyDelimiter> </keyDelimiter>
          </type>
      </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will execute the SQL query specified by the `queries.query1` property. The results of the query are compared to the grammar in the `queryCondition` job property. If the query result matches the queryCondition property then HEYMon remembers this and goes to sleep until the next interval.
If the results of the sql query match the queryCondition in each consecutive interval, then HEYMon assumes there is an alert condition.
If the job worker determines the query result does not match the value of the `queryCondition` job property for each consecutive interval then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessSystemInfoJob Grammars

The ProcessSystemInfoJob job utilizes the `queries.queryx` properties to evaluate which metrics are needed from HEYMon Agent, and when an alert condition exists.

The keyDelimiter grammar

The keyDelimiter property does not have a specific grammar. It is used as a property that dictates the delimiter character or characters utilized for positional expressions in a job query. If the keyDelimiter is set to a comma (,) and the job query has the expression `$2=value`, then `$2` relates to the string '`goo`' in the value '`foo,goo,you,too`'.

The queries.query1 grammar

The ProcessSystemInfoJob can read any of the server metrics available from HEYMon Agent. Server metrics can be collected as statistics, or can be evaluated for various alert conditions.
The following values for the queries.query1 grammar represent the metrics available from HEYMon Agent:

- DISKSPACE: The remaining, available disk space in MegaBytes (MB).
- TOTALMEMORY: The total memory installed as reported by the operating system in MegaBytes (MB).
- AVAILABLEMEMORY: The total memory available (unused) as reported by the operating system in MegaBytes (MB).
- PROCESS: An operating system process, based on the executable which spawned the process.
- SERVICE: A Windows Service process.
- WINEVENT: A Windows Event Log on a remote server.

For each server metric that HEYMon should collect, one query grammar should be configured. For example, if we had a HEYMon job that monitored disk space and available memory we would need two `queryx` properties as follows:

```
<queries>
   <query1>DISKSPACE:</query1>
   <query2>AVAILABLEMEMORY:</query2>
</queries>
```

When the `queryx` properties are configured with only the metric names as in the aforementioned example, HEYMon will consider the data to be a statistic and will send one alert for each `queryx`. In the above example, two alerts will be sent out – one with the disk space and another with the available memory.

The `queryx` properties can contain conditional statements which indicate to HEYMon when to send out an alert. The conditional statements differ slightly based on which metric is being configured:

- DISKSPACE, AVAILABLEMEMORY: Can be tested for values that are greater-than or less-than a numerical value in MegaBytes (MB).

Windows Example:
```
DISKSPACE:C<1024
```
Linux/MacOSX Example:
```
DISKSPACE:/dev/local0<1024
```
Sends an alert when the disk space on the drive is less than 1024MB.
For Windows disks, the drive letter must be used.
For UNIX, Linux, and MacOSX, the forward-slash delimited mount path must be used.
All other path types (UNC, Novell, NFS, AppleTalk, etc.) are not evaluated by HEYMonAgent.
Example:
```
AVAILABLEMEMORY:<2048
```
Sends an alert when the available memory in the operating system is less than 2048MB.
NOTE: It is not recommended to use the Equals comparator(=). Diskspace and RAM will rarely hit a fixed point for more than a few milliseconds, and therefore HEYMon may never be able to find a condition that is equal to a fixed value.

- PROCESS, SERVICE: Can be tested to see if running or not.
  Example:
  ```
  PROCESS:=c:\path\path2\file.exe
  ```
  Sends an alert if the process file.exe is running.
  Example:
  ```
  SERVICE:!=AWindowsService
  ```
  Sends an alert if the AWindowsService is not running.
  NOTE: Only the use of equals (=) or not equals (!=) can be used with PROCESS and SERVICE.
- WINEVENT: Can be configured like the ProcessErrorLogJob to recognize patterns in the Windows Event Log that represent alert conditions or statistics
  Example:
  ```
  WINEVENT:$3=ERROR
  ```
  Sends an alert if the third space-delimited element contains the string ‘ERROR’.
  This Windows Event Log would be recognized as an alert:
  ```
  2012-04-12 07:58:16,453 ERROR Increasing runlevel to
  'GUIDEWIRE_STARTUP'
  ```

Note that if any of the metrics are improperly configured, HEYMon will write the details to its log file and ignore the configuration element

ProcessSVNChangeJob

The ProcessSVNChangeJob is designed to connect to a Subversion source code repository and check files for changes and check directories for file additions or deletions.
The job support SVN 1.6, 1.7, and 1.8, and supports the http, svn, or file protocols.

ProcessSVNChangeJob job type properties

The ProcessSVNChangeJob job type contains the following properties with the related definition:

- login: The login value used to access the Subversion repository.
- passVal: The password value used to access the Subversion repository.
- notifyOnce: If set to the value 'true', this job type will only send one notification for each file change. If not set to a value of 'true', HEYMon will continue to send alerts for any and all file changes after first discovery.

ProcessSVNChangeJob job properties

When the ProcessSVNChangeJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: This property is not used for the ProcessSVNChangeJob. It should still be defined in job.properties, and can be set to any value or no value.
- systemID: The name of the SVN server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between checking the SVN server for changes.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: At least one query is required. The query represents the full path and protocol used to check SVN for changes.
  To have this job monitor a **file** for changes, use a full SVN path to the file:
  `http://mySvnServer/svn/trunk/ProjectA/src/util/myFile.dat`
  To have this job monitor a **directory** for changes, use a full SVN path to the directory, terminated with a slash character (/):
  `http://mySvnServer/svn/trunk/ProjectB/data/`
- notifySubject: When an alarm is determined by the ProcessSVNChangeJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the date and time of any changes being monitored. To have the date and time added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used for the ProcessSVNChangeJob. It should still be defined in job.properties, and can be set to any value or no value.

ProcessSVNChangeJob process

The ProcessSVNChangeJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="SVN file change monitor">
  <description>Generate alerts for modifications in a SVN repo.</description>
  <connection>Foo_cnn</connection>
  <systemID>svn_server</systemID>
  <scanIntervalSeconds>30</scanIntervalSeconds>
  <destination>notify_bob</destination>
  <queries>
     <query1>http://svnServer/svn/trunk/productA/src/utils/file.dat</query1>
     <query2>http://svnServer/svn/trunk/productB/src/content/</query2>
  </queries>
  <notifySubject>SVN File was modified on %</notifySubject>
  <notifyBody>%s</notifyBody>
  <type name="ProcessSVNChangeJob">
      <login>person</login>
      <passVal>password_here</passVal>
  </type>
</job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
At the end of the duration, the job worker will connect to the SVN server using the `login` and `passVal` job type properties. If the login is successful, the job worker will check for changes in the files or directories specified by the collection of `query` properties. The property `query.query1` will be parsed first, followed by `query.query2` if it exists, and so on.
If changes are found in any of the `query` properties then HEYMon will consider the changes as an alert condition.
If the job worker determines there are no changes, then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.
If the job worker encounters one of the following conditions a notification is made via email to the email addresses configured for the `destination` meta-property:

- A SVN file was changed in any way, for any reason.
- A file was added or removed from an SVN directory, for any reason.

ProcessSVNChangeJob Grammars

The ProcessSVNChangeJob job contains a grammar to determine if alerts should be sent more than one time.

The notifyOnce grammar

If set to the value 'true', this job type will only send one notification for each file change. If not set to a value of 'true', HEYMon will continue to send alerts for any and all file changes after first discovery.

ProcessDriveMountJob

The ProcessDriveJob is designed to check for existing drive mounts and send alert notifications if any of the mounts change.
The ProcessDriveMountJob can be used to detect changes with any type of drive mount:
- Drive letters in Windows
- USB storage devices
- Media devices such as CD and DVD
- Remote mounts in Linux, Mac Osx, and UNIX.
- Network file mounts like Windows shared drives, NFS mounts, and Samba.

ProcessDriveMountJob job type properties

The ProcessDriveMountJob job type contains the following properties with the related definition:

- exception: A grammar which allows the exclusion of certain types of drive mounts from being monitored.
- notifyOnce: If set to the value 'true', this job type will only send one notification for each drive mount change. If not set to a value of 'true', HEYMon will continue to send alerts for drive mount changes after first discovery.

ProcessDriveMountJob job properties

When the ProcessDriveMountJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: This property is not used for the ProcessDriveMountJob. It should still be defined in job.properties, and can be set to any value or no value.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between checking the server for drive mount changes.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: The query represents an optional flag to determine if alerts are needed for added mounts and/or for deleted mounts. Leave this property set as NULL to be notified about all drive mount changes. If only interested in drives that have been added, set this query property to 'added'. If only interested in drives that have been deleted, set this query property to 'deleted'.
- notifySubject: When an alarm is determined by the ProcessDriveMountJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the date and time of any changes being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.

- notifyBody: This property is not used for the ProcessDriveMountJob. It should still be defined in job.properties, and can be set to any value or no value.

ProcessDriveMountJob process

The ProcessDriveMountJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="Drive mount change notification">
    <description>Create alert if drive mount is added/removed.</description>
    <connection>server_string</connection>
     <systemID>lock_server</systemID>
     <scanIntervalSeconds>30</scanIntervalSeconds>
     <destination>email_admins</destination>
     <queries>
        <query1>added,deleted</query1>
     </queries>
     <notifySubject>Drive Mount change on %s</notifySubject>
     <notifyBody></notifyBody>
     <type name="ProcessDriveMountJob">
         <exception>NETWORK</exception>
         <notifyOnce>false</notifyOnce>
     </type>
 </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
When the job worker is executed the first time, it scans the system for all available drive mounts and saves the list for comparison during future work cycles. The job worker then goes to sleep until the next work cycle.
At the end of the sleep duration, the job worker will check for changes in the current list of drive mounts.
If changes are found in any of the drive mounts, then HEYMon will consider the changes as an alert condition.
If the job worker determines there are no changes, then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessDriveMountJob Grammars

The ProcessDriveMountJob job contains grammars to control whether alerts should be sent for drive mounts that have been added and/or whether alerts should be sent for drive mounts that have been deleted. There is also a grammar used to specify types of drive mounts which should be ignored.

The queries.query1 grammar

To control whether alerts should be sent for added or deleted drive mounts, use one of the following configurations for the queries.query1 property:

- `<query1> </query1>`: This is the NULL, or default setting. The ProcessDriveMountJob will check for drive mounts that have been added or deleted.
- `<query1>added</query1>`: This is the 'added' setting. The ProcessDriveMountJob will only check for drive mounts that have been added.
- `<query1>deleted</query1>`: This is the 'deleted' setting. The ProcessDriveMountJob will only check for drive mounts that have been deleted.
- `<query1>added,deleted</query1>`: This is the default setting. The ProcessDriveMountJob will check for drive mounts that have been added or deleted.

The exception grammar

This grammar is utilized to create an exception for the types of drive mounts being monitored.
Valid values are:

- NETWORK: Do not send an alert if a network drive mount changes. This is generally used when network drive mounts do not matter, or if network mounts appear and disappear due to network traffic.
- MEDIA: Do not send an alert if a media drive mount changes. Media drives are usually CD, DVD, or Blu-Ray devices.
- STORAGE: Do not send an alert if a storage mount changes. Storage mounts are usually USB memory devices, external hard drives, and portable devices like tablets and phones.

The notifyOnce grammar

If set to the value 'true', this job type will only send one notification for each drive mount change. If not set to a value of 'true', HEYMon will continue to send alerts for drive mount changes after first discovery.

ProcessDirectoryContentJob

The ProcessDirectoryContentJob is designed to check for files in a specific directory and send alert notifications if any files are added or removed.

ProcessDirectoryContentJob job type properties

The ProcessDirectoryContentJob job type contains the following properties with the related definition:

- filePattern: A grammar which allows a file mask or regular expression that dictates which files to monitor.
- notifyOnce: If set to the value 'true', this job type will only send one notification for each drive mount change. If not set to a value of 'true', HEYMon will continue to send alerts for drive mount changes after first discovery.

ProcessDirectoryContentJob job properties

When the ProcessDriveMountJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: This property is not used for the ProcessDirectoryContentJob. It should still be defined in job.properties, and can be set to any value or no value.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between checking the server for directory file changes.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: The query represents the full path to the directory being monitored for file additions and deletions.
- notifySubject: When an alarm is determined by the ProcessDirectoryContentJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used for the ProcessDirectoryContentJob. It should still be defined in job.properties, and can be set to any value or no value.

ProcessDirectoryContentJob process

The ProcessDirectoryContentJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
     <job name="Directory Content change monitor">
         <description>Generate alert if a file is added/removed from
directory.</description>
         <connection>query_bc_connect_string</connection>
          <systemID>bc_app_server</systemID>
          <scanIntervalSeconds>1200</scanIntervalSeconds>
          <destination>email_notify_al</destination>
          <queries>
             <query1>c:\tmp\HEYMon_test</query1>
           </queries>
          <notifySubject>Directory content on %s changed</notifySubject>
          <notifyBody>%s</notifyBody>
          <type name="ProcessDirectoryContentJob">
            <filePattern1>txt</filePattern1>
            <notifyOnce>false</notifyOnce>
          </type>
      </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
When the job worker is executed the first time, it scans the system for all available files in the specified directory(ies) and saves the list for comparison during future work cycles. The job worker then goes to sleep until the next work cycle.
At the end of the sleep duration, the job worker will check for changes in the current list of files.
If files have been added or removed from the specified directory, then HEYMon will consider the changes as an alert condition.
If the job worker determines there are no changes, then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessDirectoryContentJob Grammars

The ProcessDirectoryContentJob job contains a grammar to control the files that are checked in each of the specified directories. There is also a grammar to control if HEYMon should notify once of changes, or keep notifying of changes.

The filePattern grammar

This optional grammar is used to limit which files that HEYMon looks at to determine directory content changes. To monitor all files in a directory, this property should not be defined or should be set to a NULL value.
To monitor all files with a given file extension, include one or more file extension values separated by a comma. To only monitor .TXT and .DOC files, set the filePattern property like this:
`<filePattern>txt,doc</filePattern>`
A standard Regular Expression can also be used. The expression will be matched against the file name.

The notifyOnce grammar

If set to the value 'true', this job type will only send one notification for each file change. If not set to a value of 'true', HEYMon will continue to send alerts for any and all file changes after first discovery.

ProcessFileChangeJob

The ProcessFileChangeJob is designed to check if a file is modified or deleted.

ProcessFileChangeJob job type properties

The ProcessFileChangeJob job type contains the following properties with the related definition:

- notifyOnce: If set to the value 'true', this job type will only send one notification for each file change. If not set to a value of 'true', HEYMon will continue to send alerts for file changes after first discovery.

ProcessFileChangeJob job properties

When the ProcessFileChangeJob job type is configured for a given job, the following job properties are utilized in the capacity as documented:

- connection: This property is not used for the ProcessFileChangeJob. It should still be defined in job.properties, and can be set to any value or no value.
- systemID: The name of the server being monitored. This name is usually used in the subject and body of the email alerts so that administrators know the affected server name.
- scanIntervalSeconds: The number of seconds between checking the server for file modifications.
- destination: The meta-property that refers to the email address that alerts should be sent for this job.
- queries.query1: The query represents the full path and file name being monitored for modifications.
- notifySubject: When an alarm is determined by the ProcessFileChangeJob job type, the subject of the email is created from this property. This property can contain one job variable, which is the name of the server being monitored. To have the server name added to this property add the variable '%s' in the property string. If the property string does not contain the variable it will still be used as the subject of the alert email.
- notifyBody: This property is not used for the ProcessFileChangeJob. It should still be defined in job.properties, and can be set to any value or no value.

ProcessFileChangeJob process

The ProcessFileChangeJob job type is driven by the job properties which have been configured by an administrator. For an example of the process, the following job properties will be used:

```
<job name="File change monitor">
    <description>Generate alerts if files are modified or deleted.</description>
    <connection>query_bc_connect_string</connection>
     <systemID>bc_app_server</systemID>
     <scanIntervalSeconds>1200</scanIntervalSeconds>
     <destination>email_notify_al</destination>
     <queries>
        <query1>c:\tmp\HEYMon_test\chart_query_issue.txt</query1>
      </queries>
     <notifySubject>File was modified on %s</notifySubject>
     <notifyBody>%s</notifyBody>
     <type name="ProcessFileChangeJob">
        <notifyOnce>false</notifyOnce>
     </type>
 </job>
```

The process begins when the HEYMon application is started.
A job worker thread is created for the job, and the thread goes to sleep for the duration specified by the `scanIntervalSeconds` property.
When the job worker is executed the first time, it scans the system for the properties of the configured file(s) and saves the details for comparison during future work cycles.
The job worker then goes to sleep until the next work cycle.
At the end of the sleep duration, the job worker will check for changes in the configured file(s) properties.
If any property of the file changes, then HEYMon will consider the changes as an alert condition.
If the job worker determines there are no changes, then there is no alert condition and no notification. The job worker will then go to sleep and the entire process starts over again.

ProcessFileChangeJob Grammars

The ProcessFileChangeJob job contains a grammar to control if HEYMon should notify once of changes, or keep notifying of changes.

The notifyOnce grammar

If set to the value 'true', this job type will only send one notification for each file change.
If not set to a value of 'true', HEYMon will continue to send alerts for any and all file changes after first discovery.

Appendix A - logLineDateFormat grammar examples

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, ''yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700