Experience Manager Guide

Operations Center 5.5

April 3, 2014

Legal Notices

Contents

About This Guide

The *Experience Manager Guide* explains how to configure and manage the Operations Center Experience Manager™. It describes the product architecture and explains how to use Experience Manager features.

Audience

This guide is intended for system administrators who are responsible for setting up and administering the Experience Manager within the Operations Center environment. It is assumed that they are familiar with Operations Center and its user interface.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the *User Comments* feature at the bottom of each page of the online documentation.

Additional Documentation & Documentation Updates

This guide is part of the Operations Center documentation set. For the most recent version of the *Experience Manager Guide* and a complete list of publications supporting Operations Center, visit our Online Documentation Web Site at Operations Center 5.5 online documentation.

The Operations Center documentation set is also available as PDF files on the installation CD or ISO; and is delivered as part of the online help accessible from multiple locations in Operations Center depending on the product component.

Additional Resources

We encourage you to use the following additional resources on the Web:

- NetIQ User Community (https://www.netiq.com/communities/): A Web-based community with a variety of discussion topics.
- NetIQ Support Knowledgebase (https://www.netiq.com/support/kb/product.php?id=SG_XOPERATIONSCENTER_1_2): A collection of in-depth technical articles.
- NetIQ Support Forums (https://forums.netiq.com/forumdisplay.php?26-Operations-Center): A Web location where product users can discuss NetIQ product functionality and advice with other product users.

Technical Support

You can learn more about the policies and procedures of NetIQ Technical Support by accessing its Technical Support Guide (https://www.netiq.com/Support/process.asp#_Maintenance_Programs_and).

Use these resources for support specific to Operations Center:

- Telephone in Canada and the United States: 1-800-858-4000
- Telephone outside the United States: 1-801-861-4000
- E-mail: support@netiq.com (support@netiq.com)
- Submit a Service Request: http://support.novell.com/contact/ (http://support.novell.com/contact/)

Documentation Conventions

A greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path. The > symbol is also used to connect consecutive links in an element tree structure where you can either click a plus symbol (+) or double-click each element to expand them.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a forward slash to preserve case considerations in the UNIX* or Linux* operating systems.

A trademark symbol (®, ™, etc.) denotes a NetIQ or Novell trademark. An asterisk (*) denotes a third-party trademark.

1 Experience Manager Overview

NetIQ Operations Center Experience Manager proactively monitors Web site and Web application availability in real time, from the user perspective. You use Experience Manager to identify and resolve potential infrastructure issues before customers experience problems. Experience Manager emulates end-user business processes against applications on a 24x7 basis, including Web and non-Web environments, and applications.

Use Experience Manager to locate and resolve the following common sources of performance problems:

- Transaction response times
- Sources of availability failures
- Network performance bottlenecks
- Slow-loading Web page components

Traditional management tools provide a partial picture of the current state of a network, application, or system component because they cannot differentiate between degrees of availability. For example, a monitoring system might indicate that all applications and network resources are available, while in reality, end users are experiencing unacceptable response time.

Experience Manager works with Operations Center to process and gather application and performance metrics from a wide variety of sources. Use Experience Manager to:

- Establish a response time baseline
- Manage response time, synthetic testing, and application performance from a single application

Experience Manager's and Operations Center Business Service Management can evaluate the state of business views in real-time and on a continuing basis so you can prioritize repairs and identify performance trends.

- Section 1.1, "How Experience Manager Works," on page 9
- Section 1.2, "Product Components," on page 11

1.1 How Experience Manager Works

Experience Manager measures and evaluates performance based on Service Views defined in Operations Center. The data that determines performance originates from:

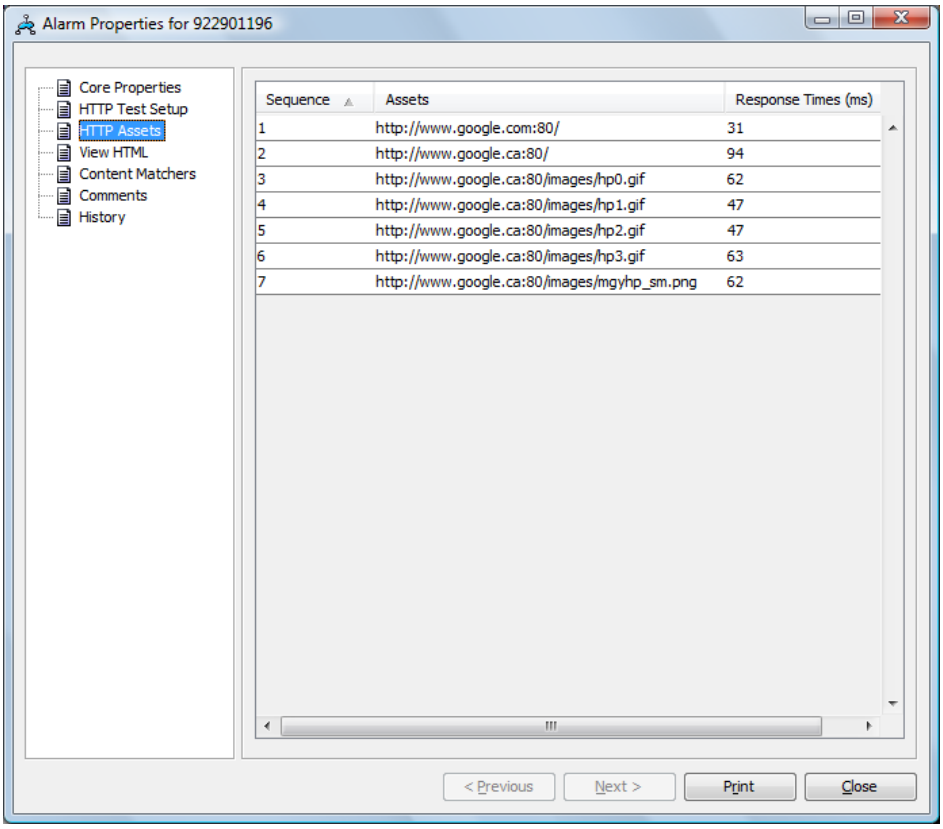
- Section 1.1.1, "Synthetic Testing," on page 10
- Section 1.1.2, "End-User Response Times," on page 11

1.1.1 Synthetic Testing

Synthetic testing simulates user transactions and transaction levels. For example, synthetic tests can simulate the steps taken by a user to view a Web page, send e-mail, or use FTP to retrieve a file.

Experience Manager can run these simulations from diverse points inside and outside of a firewall. From each point, it can test any number of transactions for one or more users. For example, Figure 1-1 shows assets for a synthetic HTTP transaction and provides a detailed list of page component load times.

Figure 1-1 *Alarm Properties Assets*



1.1.2 End-User Response Times

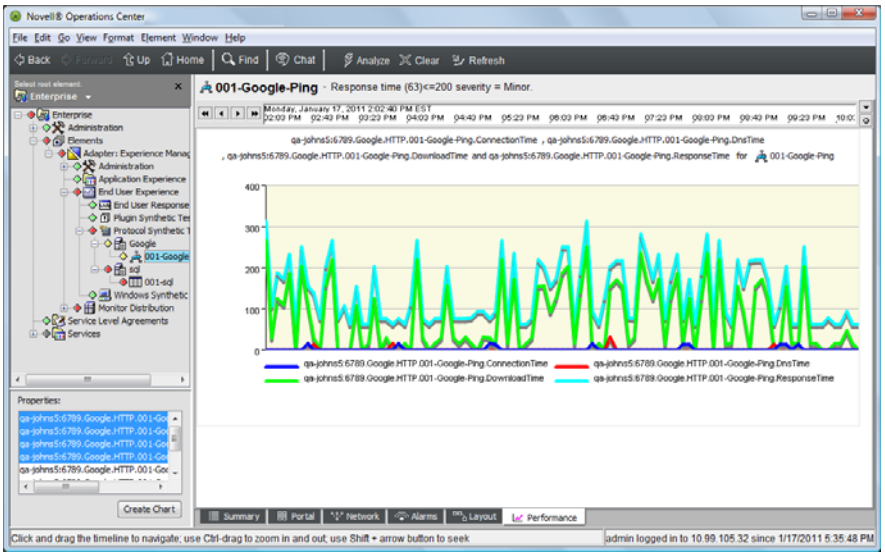
Experience Manager monitors user response times by using server-side instrumentation to determine:

- Completion time for a transaction
- Differences between the actual and baseline response time

Experience Manager monitors Web-related actions such as page downloads, user logins, and completion of Web transactions without any modifications to the browser environment or agents on end users' machines.

Use Experience Manager and Operations Center to compare and evaluate various metrics on a Web site or to compare response times for more than one Web page. Experience Manager records the metrics over time so you can chart graph performance over time in the Operations Center Performance view. For example, Figure 1-2 charts various response times for the Google Web site.

Figure 1-2 *Performance View Chart*

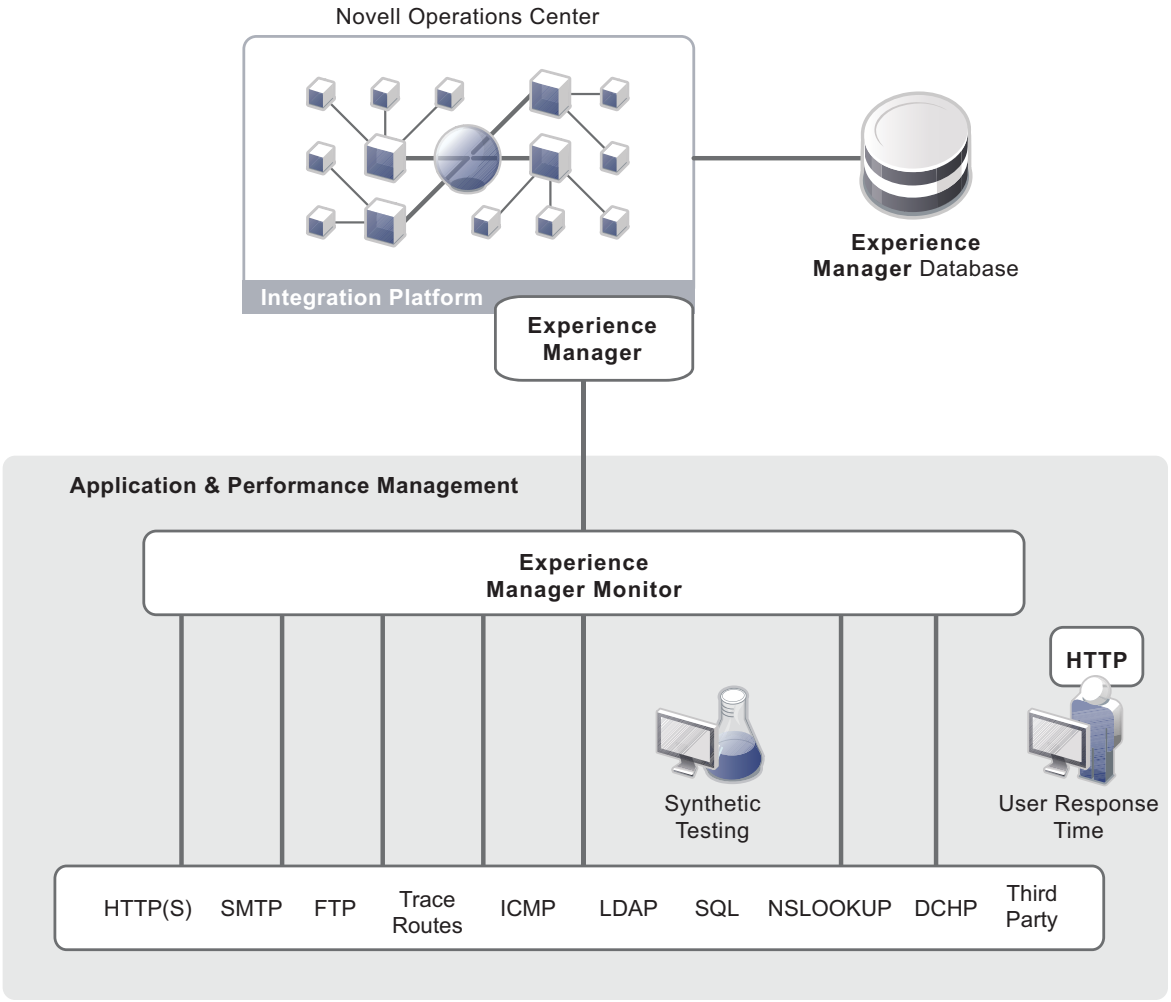


1.2 Product Components

The Experience Manager architecture consists of the following components:

- Section 1.2.1, "The Experience Manager Adapter," on page 12
- Section 1.2.2, "The Experience Manager Database," on page 12
- Section 1.2.3, "Experience Manager Monitors," on page 13

Figure 1-3 *Experience Manager Architecture*



1.2.1 The Experience Manager Adapter

The Experience Manager adapter manages:

- Connectivity with the Experience Manager Monitors
- Configuration information for each subscribed Experience Manager Monitor
- Hierarchy information used to display elements as they are aggregated from the Experience Manager Monitors (using the Managed Objects Definition Language [MODL])
- Data persistence configuration information for all data aggregated from the monitors (including connectivity and purging information)
- All test configuration information that is stored in the database

1.2.2 The Experience Manager Database

The Experience Manager database stores the usage information from Experience Manager Monitors and test configuration information. Supported databases include: Oracle, Microsoft SQL Server, PostgreSQL, Sybase, and DB2.

See the *Operations Center 5.5 Getting Started Guide* for supported database versions.

For instructions on configuring Experience Manager for the database, see Section 3.1, “Configuring Windows Servers for Single Sign On (SSO),” on page 23 and Section 3.2, “Creating a Experience Manager Adapter,” on page 24.

1.2.3 Experience Manager Monitors

Experience Manager Monitors are responsible for communication between the Operations Center server and the targets used to collect performance metrics or run synthetic testing of HTTP, HTTPS, FTP, SMTP/POP3, LDAP, SQL, TRACEROUTE, ICMP, DHCP, and NSLOOKUP sessions, as well as third-party custom tests. Based on deployment and scalability requirements, install Experience Manager Monitors on a single machine or distribute them among multiple machines.

2 Installing Experience Manager

Install Experience Manager as part of the customized Operations Center software installation.

It is necessary to install the Experience Manager software directly on each host machine.

2.1 Requirements for the Experience Manager

The Experience Manager Monitor is validated to install and run on the same platforms as the Operations Center server. For a list of supported platforms and minimum server requirement see *Operating Systems* and *Requirements for Experience Manager* in the *Operations Center 5.5 Getting Started Guide*.

Actual requirements can vary depending on the types of tests setup, the amount of users being simulated, thread counts, etc. NetIQ Consulting can provide a recommendation after understanding the environment and setting up some test cases.

2.2 Installing the Experience Manager

During a custom installation of the Operations Center software, select the *Experience Manager* check box to install Experience Manager. The Experience Manager installation automatically includes installation of the Experience Manager Recorder.

If you are installing Experience Manager on the same machine as the Operations Center server, it should be installed into a separate directory from the Operations Center server.

Table 2-1 describes the settings you need to configure during the installation process.

Table 2-1 *Experience Manager Installation Settings*

Experience Manager Setting	Description
Monitor Name	The fully qualified name of the server on which Experience Manager Monitor is installed. When Experience Manager Monitors are added to the Experience Manager adapter later (from the Add Monitor dialog box in the Operations Center console), it is important to use the same Experience Manager Monitor name as specified in this installation dialog box.
Monitor Port	The port that the monitor listens to for information from Operations Center. The default is 6789. **IMPORTANT:** If another program uses the specified port, the Experience Manager Monitor does not successfully start or run.
Monitor Web Port	The port used by the Experience Manager Monitor's Web server to gather end user support data. The default is 8080. **IMPORTANT:** If another program uses the specified port, the Experience Manager Monitor cannot successfully start or run. Be sure to take into consideration ports used by other Web servers on the monitor host.
Monitor Daemon Thread Port	The port on which the Experience Manager daemon listens for the Experience Manager Monitor start, stop, and recycle commands issued from the Operations Center Experience Manager adapter. The default is 7777.
Monitor Password	Optional password required to access the Experience Manager monitor. If your are defining a password, note that when managing the Experience Manager Monitor from Operations Center, the password is required.
Install Experience Manager as a Windows Service	If you are performing a Windows installation, select *Yes* to register the Experience Manager Monitor with Windows and start it automatically every time Windows is started. For non-Windows installations, select *No*. It is necessary to have administrative privileges on the machine in order to install the monitor as a Windows Service.

2.3 Port Usage

This section defines the ports and port requirements for running Experience Manager:

- Section 2.3.1, "Monitor Ports," on page 17
- Section 2.3.2, "Recorder Ports," on page 17
- Section 2.3.3, "Operations Center Ports," on page 17

IMPORTANT: The Experience Manager Monitor software must be installed on each machine where data collection from end-user response time or application monitoring is directed, or from where synthetic tests execute.

2.3.1 Monitor Ports

Monitor ports are used for communication with the Experience Manager Monitor. They defined during the installation process and are stored in the `/OperationsCenter_ExperienceManager_install_path/config/monitor.properties` file Table 2-2 describes the default values.

Table 2-2 *Ports Used by the Experience Manager Monitor*

Port Property	Default	Description
Monitor.Port	6789	The port that the main monitor server listens on for incoming requests from Operations Center.
Monitor.DaemonPort	7777	The port used by the Operations Center server to start, stop, or restart the monitor.
Monitor.WebPort	8080	The port that the Web server listens on for incoming requests from Web clients.

2.3.2 Recorder Ports

Open the ports defined in the `/OperationsCenter_ExperienceManager_install_path/config/recorder.properties` file if you are using the Experience Manager Monitor to record Web transactions from a remote browser. Table 2-3 describes the recorder ports.

Table 2-3 *Recorder Ports*

Port Property	Default	Description
Recorder.Port	8777	Identifies the port that the recorder listens on for incoming HTTP requests from Web clients.
Recorder.SSLListeningPort	8999	Identifies the port that the recorder listens on for incoming HTTPS requests from Web clients.

2.3.3 Operations Center Ports

For Experience Manager to communicate with the Operations Center Experience Manager adapter, the adapter properties for the listening port must be open on the Operations Center server. The default port is 6790.

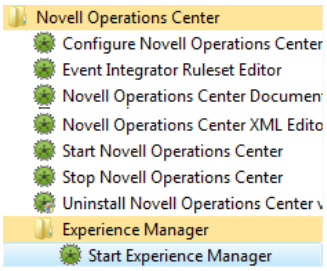
2.4 Starting the Experience Manager Monitor

It is necessary to start the Experience Manager Monitor before trying to connect to it from the Operations Center server.

A connection to the Operations Center server is unnecessary for the Experience Manager Monitor to run and collect data, assuming the Monitor has some saved tests. The Monitor can run and collect data even if the Operations Center server is down. When the Operations Center server does start, all data uploads to the server from the Experience Manager Monitor.

To start a Experience Manager Monitor:

1. Verify that the Operations Center server to which the monitor points is up and running.
2. Do one of the following:
   - **On Windows:** Do one of the following:
     1. **Start as an application:** From the desktop on the machine on which the monitor software is installed, click *Start* > *Programs* > *NetIQ Operations Center* > *Experience Manager* > *Start Experience Manager*.



     2. **Start as a Windows service:** Start the monitor through the Windows Services dialog box.
   - **On Unix:** From the `/OperationsCenter_ExperienceManager_install_path/bin` directory, run `Start_Daemon &` and then run `Start_Monitor &`.

   The Monitor starts.

2.5 Checking Experience Manager Monitor Status

To verify the Experience Manager Monitor status without opening the Services console:

1. Add the `/OperationsCenter_ExperienceManager_install_path/bin` directory for the Experience Manager installation to the class path.
2. Do one of the following:
   - **On Windows:** Enter `monitorstatus.bat` at the commend prompt.
   - **On UNIX:** Run `monitorstatus`.

   If the Experience Manager Monitor is running, an output similar to the following displays:

```
D:/OperationsCenter_ExperienceManager_install_path/bin>monitorstatus.bat
Checking Monitor: 6789
Monitor Returned STATUS:AUTH
Name = mycomputer.company.com
Port = 6789
IP Address = 206.55.26.213
Web Port = 8080
Daemon Port = 7777
Native Window's Event Port = 6777
```

2.6 Changing Experience Manager Monitor Passwords

Changing the password for a monitor is a multi-step process that also requires updating the password in Operations Center console so the monitor can be managed.

To change the monitor password:

1. Stop the Experience Manager monitor.

   For instructions, see Section 4.5.1, "Stopping a Monitor and the Daemon," on page 36.

2. Do the following to run the change password script:

   2a. From the command prompt, navigate to the `/OperationsCenter_ExperienceManager_install_path/bin` directory for the Experience Manager Monitor installation directory and enter the following command:

   ```
   changepassword.bat
   ```

   2b. Type the current and new passwords as prompted. A password change notification displays:

   ```
   Password set in /OperationsCenter_ExperienceManager_install_path/config/
   monitor.properties Password has been reset. Please restart monitor.
   ```

3. Restart the monitor to update the password change.

   For instructions, see Section 4.5.2, "Starting/Restarting a Monitor and the Daemon," on page 37.

4. Do the following to update the password in the Operations Center console:

   4a. In the Operations Center Explorer pane, right-click the monitor name, then click *Monitor Configuration* > *Admin* > *Change Password*.

   4b. Type the old password and the new password.

5. Manage the Experience Manager monitor.

   For instructions, see Section 4.4, "Managing Experience Manager Monitors," on page 35.

2.7 Dynamic Code Updates to Experience Manager Monitor Installations

Experience Manager Monitors function independently from the Operations Center server. However, a connection between the Experience Manager Monitor and the Operations Center server is required for the transfer of performance data or configuration information. If the connection is absent, the monitor continues to function. After reestablishing the connection, information automatically uploads to the server.

Experience Manager Monitors dynamically retrieve new code from the Operations Center server during the restart process for code updates and patches.

To verify that updates were successfully downloaded for a monitor, check the `/logs/bootstrap.trc` file on the machine where the monitor is installed.

2.8 Applying Patches to the Experience Manager Installation

To apply patches for a Experience Manager installation to the Experience Manager Monitor or Recorder, as directed by Support (https://www.netiq.com/support/):

- Section 2.8.1, "Applying Experience Manager Patches," on page 20
- Section 2.8.2, "Using Central Patch Distribution," on page 20

2.8.1 Applying Experience Manager Patches

In a default configuration, patches are applied to the monitor directly.

To apply an Experience Manager patch:

1. Stop Experience Manager.
2. Copy the patch file to the Experience Manager installation directory.
3. Issue the following command:

   ```
   java -jar filename
   ```
4. Start Experience Manager.

2.8.2 Using Central Patch Distribution

A central patch distribution facility is also available where patches are installed on the Manage Objects server and the Experience Manager daemon checks for updates and automatically applies them.

When updates are detected, they are download and cached locally. If the monitor is running, the daemon stops the monitor, apply the updates and restarts it. The daemon will wait for the recorder to exit normally before applying the updates.

Using this configuration, patches that normally would be applied to the monitor directly, can be applied to the Operations Center server indicated as the update location and they are distributed as described above. The options for central patch distribution are configured in `Daemon.properties` file.

To apply a patch using central patch distribution:

1. Do the following to prepare for central patch distribution:
   - **1a** Open the Daemon.properties file:
   - **1b** Edit the following line to indicate where the daemon is to look for updates:

     ```
     # Root URL of location from which Experience Manager updates can be loaded.
     Leave blank if updates are only applied locally. Note the final / slash is
     important!
     ```

     bem.update.location=http://*serverName*:*serverPort*/bem/

1c Edit the following line to set how often, in minutes, the daemon looks for updates:

```
# frequency (in minutes) with which polling for updates should be done.
BEM.update.poll.interval=1
```

2 Copy the patch into the Operations Center installation directory on the Operations Center server.

There is no need to stop Experience Manager or the Operations Center server when applying patches in this method.

For information regarding restarting the Experience Manager Monitor, see Section 4.5, "Stopping, Starting, and Recycling Experience Manager Monitors," on page 36.

3 The Experience Manager Adapter

An adapter is a Operations Center interface with the underlying management systems in a technology infrastructure. The Experience Manager adapter is required to communicate with monitors.

The Experience Manager adapter is automatically installed as part of the Operations Center installation. For instructions on installing Operations Center, see the *Operations Center 5.5 Server Installation Guide*.

3.1 Configuring Windows Servers for Single Sign On (SSO)

The Experience Manager adapter can be configured to use Server Domain Authentication for single sign on with Microsoft SQL databases. Windows servers require additional configuration for SSO functionality.

To configure Windows servers for Single Sign On with Operations Center MSSQL databases:

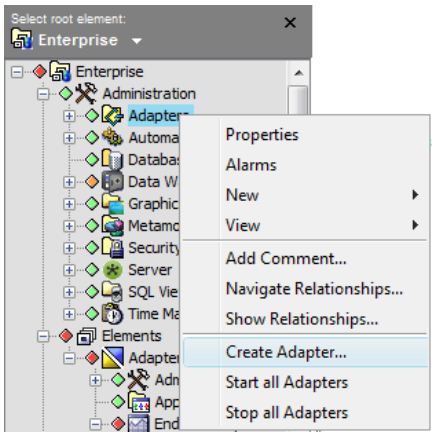
1. Stop the Operations Center server.
2. Copy one of the following files into the directory containing the `java.exe` used by the Operations Center server:
   - **On a 32-bit JVM:** *`OperationsCenter_install_path`*`/mssql/win32/ntlmauth.dll`
   - **On a 64-bit JVM:** *`OperationsCenter_install_path`*`/mssql/win64/ntlmauth.dll`
3. Start the Operations Center server.
4. Specify the domain, username and password when configuring the Experience Manager adapter with a Microsoft SQL database:
   - Leave the username and password properties blank to attempt to connect using the current Windows user account.
   - If running Operations Center as a service, the service must use the same domain user account as the database.

   For information on adapter properties used to configure domain authentication with the Experience Manage adapter, continue to Creating a Experience Manager Adapter.

3.2 Creating a Experience Manager Adapter

To create a Experience Manager adapter:

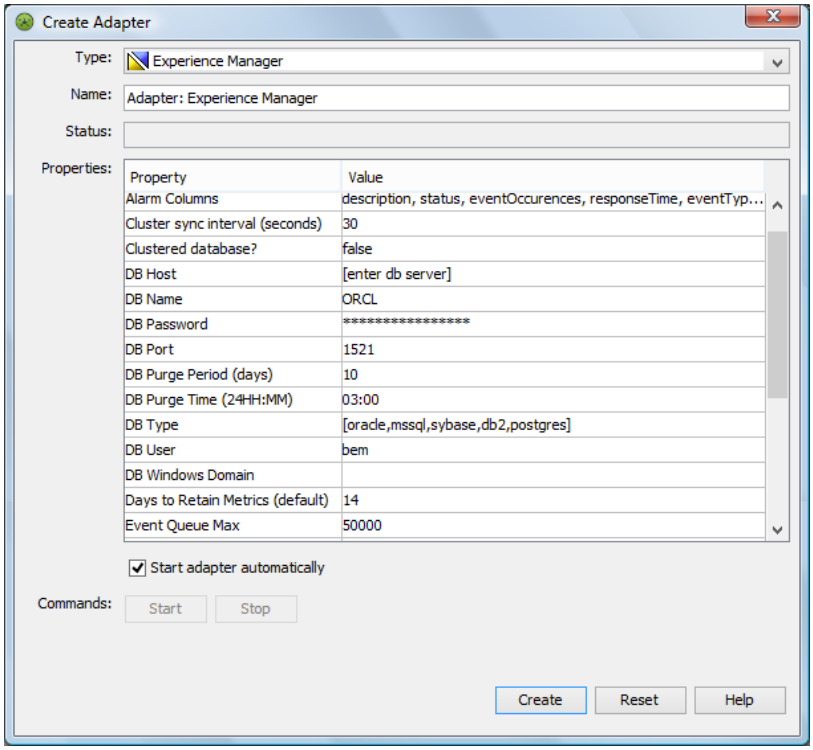
1. In the *Explorer* pane, expand the *Administration* root element.
2. Right-click *Adapters*, then click *Create Adapter*:



The Create Adapter dialog box opens.

3. Click the *Type* drop-down list, then select *Operations Center Experience Manager*.

The default name, `Adapter: Experience Manager`, appears in the Name text box.



4 In the `Name` text box, edit the default to refine the Experience Manager adapter name.

   Default adapter properties display in the Properties table.

5 Edit the property values as required. Refer to Table 3-1 on page 25 for detailed descriptions of the adapter properties.

   Leave the *Start adapter automatically* check box selected to start the adapter automatically when the Operations Center server starts. To always start the adapter manually, clear the check box.

6 Click the *Create* button.

   The *Start* button becomes active.

7 Click the *Start* button to start the adapter.

8 After creating and starting the adapter, click the *Close* button to close the dialog box.

   The new adapter displays in the list of elements in the Explorer pane, under *Administration* > *Adapters*.

3.3 Experience Manager Adapter Properties

Table 3-1 provides a description of Experience Manager Adapter properties.

Table 3-1 *Experience Manager Adapter Properties*

Property	Description
Accept events from unmanaged monitors?	Specifies if events are accepted from unmanaged monitors. Default is `false`.
AlarmColumns	A comma-separated list that determines which alarm columns display and the order in which the alarm items display (date/time, rule, etc.) in the Alarms window. A suggested list: `status, Class, eventOccurrences, responseTime, responseTimeAvg, responseTimeLow, responseTimeHigh`
Cluster sync interval (seconds)	The frequency in seconds that cluster members poll the database for changes to test configuration. This setting does not impact alarms. Defalut is `30`.
Clustered Database?	Indicates there is more than one Experience Manager adapter sharing the database in a clustered Operations Center environment. Set to `true` when there are multiple adapters running.
DB Host	The hostname on which the database resides. Usage information from Experience Manager Monitors resides in this database. If multiple Experience Manager adapters use the same database server, each should have a unique set of Experience Manager tables.
DB Name	The database name.
DB Password	The password used to access the database. If using Microsoft SQL Server with domain authentication, specify the password of the Windows user account, or leave blank (on Windows) to attempt to use the credentials of the currently active Windows user account.
DB Port	The port number on which the database host listens.

Property	Description
DB Purge Period (days)	The age of performance data to purge from the database. For example, specify 21 to purge all information older than 21 days. Performance data is deleted from APM_EVENT_EXTENSIONS_TAB. A value of never prevents the purge from occurring.
DB Purge Time (24HH:MM)	The time of day to purge the database. Use the 24-hour format of *hours*:*minutes*. For example, 17:00 starts the purge at 5:00 p.m.
DB Type	The type of database. Specify `db2`, `oracle`, `mssql`, `postgres`, or `sybase`. Note the following database specific configurations necessary for Experience Manager: ◆ For DB2, modify the `bemschema.dbscript` file to associate APM_EVENTS_TAB with a `tablespace` that has a buffer page size of 32 KB. ◆ For Sybase, verify that Sybase is configured with a minimum page size of 8K.
DB User	The user ID for accessing the database. If using Microsoft SQL Server with domain authentication, specify the username of the Windows user account, or leave blank (on Windows) to attempt to use the credentials of the currently active Windows user account.
DB Windows Domain	The domain to use for domain authentication with single sign on (used when defining a Microsoft SQL Server database).
Days to Retain Metrics (default)	All bulk metric data has an expiration date after which it is purged from the Experience Manager database. Specify the default number of days that HVMetricsbulk metric data is kept. Note that specific metrics data can be configured for a different retention period in the monitor-side script.
Event Queue Max	The maximum number of events allowed in the queue. The default is 50,000.
HierarchyFile	A file in the `/OperationsCenter_install_path/database/examples` directory that contains an XML description of the hierarchy of elements that is built beneath the element that represents the adapter. The default is `bemHierarchy.xml`. Before modifying the hierarchy file, copy the example file to the `/OperationsCenter_install_path/database` directory and change the location reference. This protects changes from being overwritten during software updates. For information, see *Using the HierarchyFile* in the *Operations Center 5.5 Adapter and Integration Guide*. The `bemHierarchy.xml` file no longer handles the setting of response time thresholds for scenarios. This functionality is now defined in scenario definitions.
Script.onError	Specify a script that executes if the adapter fails for any reason. The script can print the reason for the failure as a msg; for example: log.info(msg).
Script.onInitialized	A script that executes when the adapter is first initialized. All the Script. properties are optional.
Script.onStarted	A script that executes whenever the adapter starts, either manually or automatically when the Operations Center server starts.
Script.onStopped	A script that executes after manually stopping the adapter.

Property	Description
Secure Communications (SSL)	If True, encrypts communications traffic between the adapter and Experience Manager Monitors using SSL. If False, does not encrypt communications. This value must be the same as the setting for `Monitor.SecureCommunications` in the monitor's */OperationsCenter_ExperienceManager_install_path*`/config/monitor.properties` file. Both default to false.
Server Listener Port	The port on which the adapter listens for events from the monitors. The default is 6790.
Socket Timeout (seconds)	The number of seconds of unresponsive connection time on monitors before timed out. Default is `30`.
StylesheetFile	The style sheet file applies against the HierarchyFile as a style markup and produces the final output. It is located in the */OperationsCenter_install_path*`/database` directory.
Use Alarm Times for Cond Changes	Specifies the date/time stamp to use for any alarm data stored by the data warehouse. If True, takes the date/time information directly from the alarm information. If False, uses the date/time when the Operations Center server receives the alarm.

3.4 Starting the Adapter

The Experience Manager adapter can start automatically when Operations Center starts. Select the *Start adapter automatically* check box described in Section 3.2, "Creating a Experience Manager Adapter," on page 24 or in Section 3.6, "Modifying Adapter Properties," on page 28. Otherwise, start the adapter manually.

To start the adapter manually:

1. In the *Explorer* pane, expand the *Administration* root element > *Adapters*.
2. Right-click the adapter name and select *Start Adapter*.

   The condition indicator beside the adapter changes to green, indicating it has started.

3.5 Stopping an Adapter

To stop an adapter:

1. In the Explorer pane, expand the *Administration* root element > *Adapters*.
2. Right-click the adapter name and select *Stop Adapter*.

   The condition indicator beside the adapter changes to gray, indicating it has stopped.

3.6 Modifying Adapter Properties

To view or edit adapter properties:

1. In the Explorer pane, right-click the adapter name, then click *Properties*.

   The Status property page opens.
2. In the left pane, click *Adapter*.

   The Adapter property page opens.
3. Edit the property values, such as the *Server Listener Port* or the *Start adapter automatically* check box.
4. Click the *Apply* button to update the property values.

3.7 Checking Adapter Status

In the Explorer pane, a diamond-shaped, color-coded indicator identifies an adapter's condition or state.

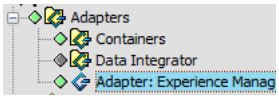


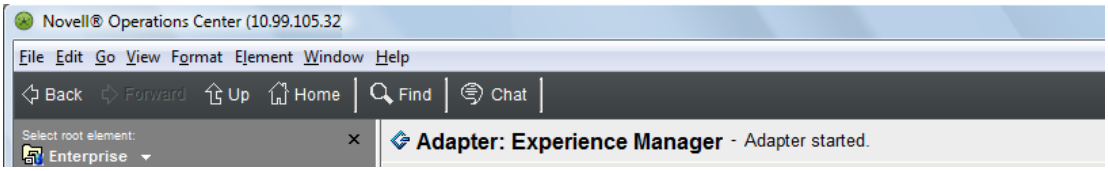
Table 3-2 describes the color codes of the adapter condition icons.

Table 3-2 *Adapter Condition Icons*

Color	Description
Red	The status of the associated element is Critical.
Orange	The status of the associated element is Major.
Yellow	When the adapter is in the process of starting or stopping, the status is Minor.
Blue	The status of the associated element is Informational.
Green	When the adapter is started and running, the status is OK.
Gray	When the adapter is stopped, the status is Unknown.
Brown	The status of the associated element is Unmanaged.

The adapter status is also summarized in a description on the View pane title bar for the adapter's *Administration* > *Adapter* element. In the illustration below the title bar indicates the adapter has started:

Figure 3-1 *View Title Bar*



To hide or display the condition indicators:

In the Explorer pane, right-click the background and select *Show Condition Indicators*.

3.8 Deleting an Adapter

IMPORTANT: Deleting an adapter does not delete the associated performance data from the database schema. To delete performance data, remove the Experience Manager Monitor before deleting the adapter.

To delete an adapter:

1. In the Explorer pane, expand the *Administration* root element > *Adapters*.
2. Right-click an adapter, then click *Delete Adapter*.

   A confirmation dialog box opens.
3. Click the *Yes* button.

   The adapter disappears from the Explorer pane.

4 Managing Experience Manager Monitors

After creating the Experience Manager adapter, the next step is to add and manage the Experience Manager Monitors. The Experience Manager Monitors establish communication between the Operations Center server and the targets for collecting performance metrics or running synthetic tests.

Each Experience Manager Monitor supports end-user response times and synthetic testing of sessions, as well as third-party custom tests.

Install the Experience Manager Monitor software on each machine from which data is collected for end user response time or application monitoring, or from which synthetic tests are executed. For directions on installing the Experience Manager Monitor, see Chapter 2, "Installing Experience Manager," on page 15.

The following sections provide instructions on installing and managing Experience Manager monitors:

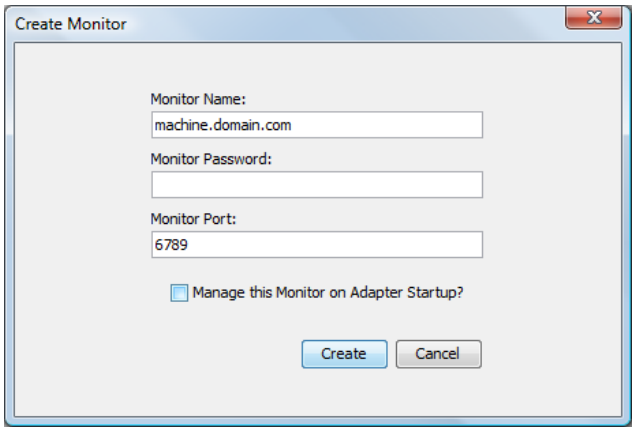
4.1 Adding a Experience Manager Monitor

To add an Experience Monitor to the adapter:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration*.
2. Right-click *Monitor Administration*, then click *Create Monitor*.

   The Add Monitor dialog box opens.

3 In the Create Monitor dialog box, specify the fully qualified name of the host where the new monitor resides and the port number on which the monitor listens:



Optionally, specify the password.

To view the Experience Manager Monitor name, password, and port specified during installation, open the `/OperationsCenter_ExperienceManager_install_path/config/monitor.properties` file on the machine where the Experience Manager Monitor is installed.

Monitor Property	Description
Monitor Name	The name of the host on which the monitor resides. This must match the name specified during the Experience Manager Monitor installation process. To edit the name entered during installation, edit the monitor name in the `/OperationsCenter_ExperienceManager_install_path/config/monitor.properties` file on the machine where the Experience Manager Monitor is installed.
Monitor Password	The optional password used to administer Experience Manager from Operations Center. This must match the password specified during the monitor's installation process. If it does not match, access is denied. If no password was specified during installation, none is required when adding the monitor.
Monitor Port	The port that the main monitor server listens on for incoming requests from Operations Center. Must match the port specified during the monitor's installation process. The default port is 6789.

4 When the Experience Manager adapter starts, select the *Manage this Monitor on Adapter Startup?* check box to automatically establish communication between the monitor and Operations Center.

To manually establish communication, leave the check box deselected.

5 Click the *Create* button.

The new monitor displays under the Monitor Administration element in the Explorer pane.

4.2 Viewing Experience Manager Monitor Status and Metrics

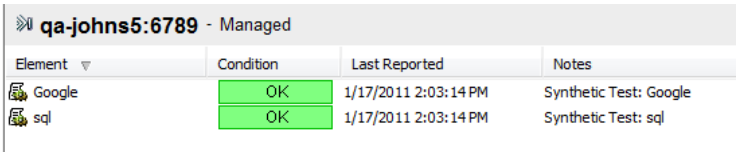
The Experience Manager Monitor status is updated automatically. In the Explorer pane, the colored diamonds beside the Experience Manager Monitors indicate the status. Table 4-1 describes the color codes.

Table 4-1 *Experience Manager Monitor Status Colors*

Status Color	Description
Green	A connection exists between Operations Center and the Experience Manager Monitor.
Gray	The monitor has not started, or if it has started, it might have a connection problem. If the monitor can successfully push alarms to the Operations Center server, the indicator changes back to green.
Yellow	A heartbeat communication problem with the monitor is probable.
Red	A heartbeat communication with the monitor has failed 10 times.

The monitor status is also summarized on the View pane title bar. The condition indicators display and the title bar indicates the adapter has started:

Figure 4-1 *The Operations Center Title Bar Shows Status for the Monitor Element*



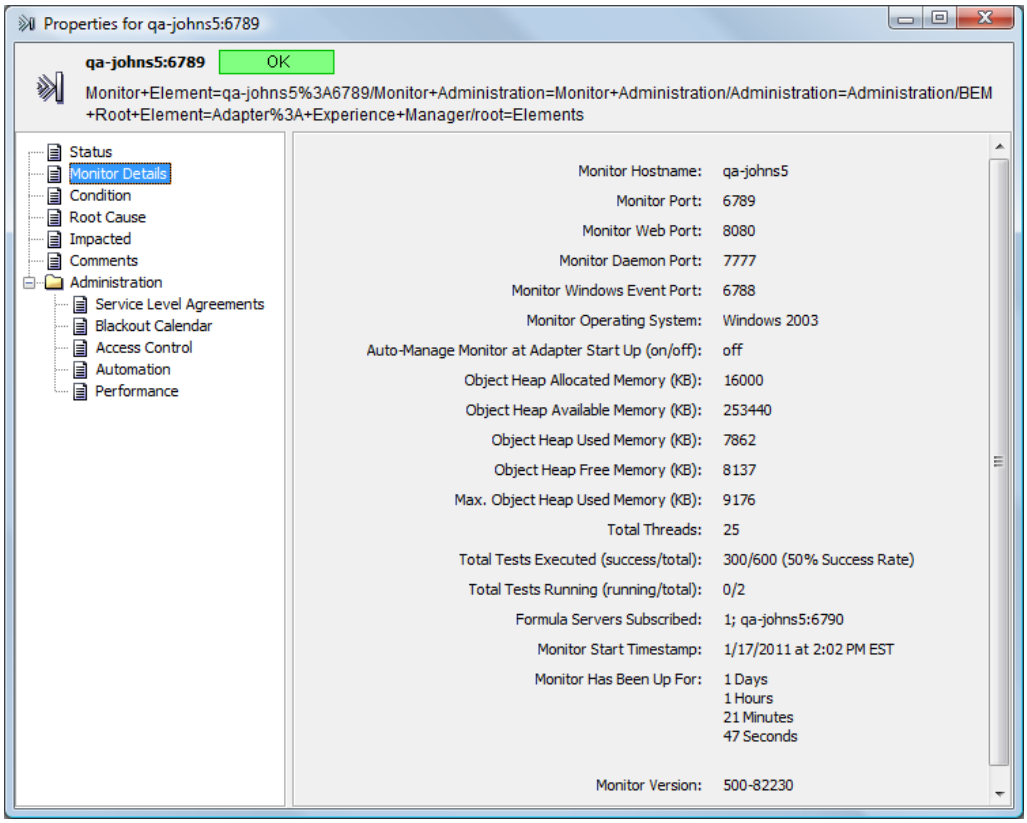
NOTE: To hide or display the status indicators, right-click anywhere in the Explorer pane background, then select or deselect *Show Condition Indicators*. When the option is selected, the indicators display.

To view monitor metrics and statistics in the monitor's property pages:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click the monitor, then click *Properties*. The Status property page opens.

3 In the left pane, click *Monitor Details*.

The Monitor Details property page displays all metrics for the monitor:



4.3 Automatically Managing Monitors

You can automatically establish communication between a Experience Manager Monitor and the Operations Center server upon Experience Manager adapter startup.

- Section 4.3.1, “Enabling the Auto-Manage Feature for a Experience Manager Monitor,” on page 34
- Section 4.3.2, “Disabling the Auto-Manage Feature for a Experience Manager Monitor,” on page 35

4.3.1 Enabling the Auto-Manage Feature for a Experience Manager Monitor

To enable the auto-manage feature for a Experience Manager Monitor:

1 In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.

2 Right-click a monitor, then click *Monitor Configuration* > *Auto Manage* > *On*.

When the Experience Manager adapter starts, communication is automatically established between the Experience Manager Monitor and the Operations Center server.

4.3.2 Disabling the Auto-Manage Feature for a Experience Manager Monitor

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click a monitor, then click *Monitor Configuration* > *Auto Manage* > *Off*.

   Communication between a monitor and the Operations Center server is not established until the *Monitor Configuration* > *Admin* > *Manage* command is selected (see Section 4.4, "Managing Experience Manager Monitors," on page 35).

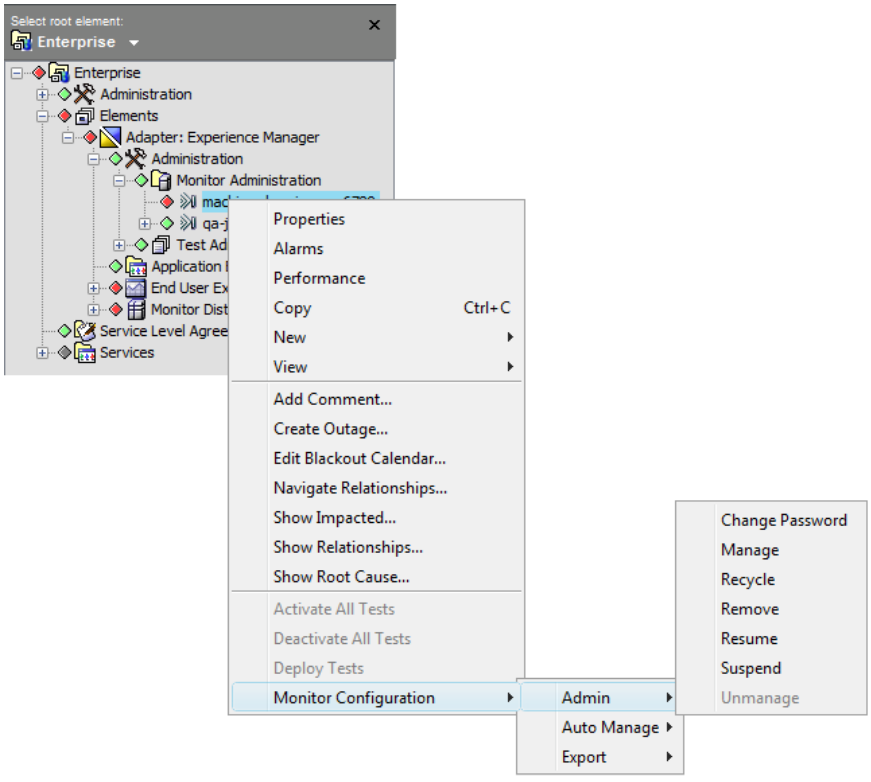
4.4 Managing Experience Manager Monitors

If communication between a Experience Manager Monitor and the Operations Center server is not established automatically, as described in Section 4.3, "Automatically Managing Monitors," on page 34, use a command to establish communication.

To manage a Experience Manager Monitor:

1. Verify that the Experience Manager adapter has started in Operations Center.
2. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
3. Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Manage*.

   The monitor status indicator should change from gray to green after communication has been established.



4.5 Stopping, Starting, and Recycling Experience Manager Monitors

Use menu commands to stop and start a Experience Manager Monitor from Operations Center without needing to telnet to the actual server:

4.5.1 Stopping a Monitor and the Daemon

You can stop the Experience Manager Monitor remotely from the Operations Center console.

To stop a Experience Manager Monitor through the Operations Center console:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Suspend*. The monitor status indicator changes to gray.

To stop a Experience Manager Monitor from the Experience Manager server:

1. For windows, do one of the following:
   - From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Stop Experience Manager*.
   - Click the `\OperationsCenter_ExperienceManager_install_path\bin\Stop Monitor.exe` file.
2. For UNIX, do the following:
   - Run `Stop_Monitor` from the `/OperationsCenter_ExperienceManager_install_path/bin` directory.

To stop both the Experience Manager Monitor and the Daemon from the Experience Manager server:

1. For windows, do one of the following:
   - From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Stop Experience Manager*.
   - Click the `\OperationsCenter_ExperienceManager_install_path\bin\Stop Experience Manager.exe` file.
2. For UNIX, do the following:
   - From the `/OperationsCenter_ExperienceManager_install_path/bin` directory, run `Stop_Daemon` and then run `Stop_Monitor`.

To stop the Experience Manager daemon from the Experience Manager server:

1. For windows, do one of the following:
   - From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Stop Experience Manager Daemon*.
   - Click the `\OperationsCenter_ExperienceManager_install_path\bin\Stop Daemon.exe` file.

2 For UNIX, do the following:

- Run `Stop_Daemon` from the `/OperatinsCenter_ExperienceManager_install_path/bin` directory.

4.5.2 Starting/Restarting a Monitor and the Daemon

You can start the Experience Manager Monitor remotely from the Operations Center console.

To restart a Experience Manager Monitor through the Operations Center console:

1 In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.

2 Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Resume*. The monitor status indicator changes from gray to green.

To start a Experience Manager Monitor from the Experience Manager server:

1 For windows, do one of the following:

- From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Start Experience Manager*.
- Click the `\OperationsCenter_ExperienceManager_install_path\bin\Start Monitor.exe` file.

2 For UNIX, do one of the following:

- Run `Start_Monitor` from the `/OperationsCenter_ExperienceManager_install_path/bin` directory.

To start both the Experience Manager Monitor and the Daemon from the Experience Manager server:

1 For windows, do one of the following:

- From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Start Experience Manager*.
- Click the `\OperationsCenter_ExperienceManager_install_path\bin\Start Experience Manager.exe` file.

2 For UNIX, do one of the following:

- Run `Start_Experience_Manager` from the `/OperationsCenter_ExperienceManager_install_path/bin` directory.

To start the Experience Manager daemon from the Experience Manager server:

1 For windows, do one of the following:

- From the Windows start menu, go to *Program Files* > *NetIQ Operations Center* > *Experience Manager* and click *Start Experience Manager Daemon*.
- Click the `\OperationsCenter_ExperienceManager_install_path\bin\Start Daemon.exe` file.

2 For UNIX, do one of the following:

- Run `Start_Daemon` from the `/OperationsCenter_ExperienceManager_install_path/bin` directory.

4.5.3 Recycling a Monitor

The Recycle menu command stops and then restarts the monitor.

To recycle a monitor:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Recycle*. The monitor status indicator changes to gray, then back to green.

4.6 Unmanaging Monitors

To stop communications between the monitor and Operations Center:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Unmanage*. The monitor status indicator changes to gray.

4.7 Removing Monitors

IMPORTANT: When removing a Experience Manager Monitor from a Operations Center client, verify that no other clients are using the Monitor before removing archived data from the database.

To remove a Experience Manager Monitor:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Monitor Administration*.
2. Right-click a monitor, then click *Monitor Configuration* > *Admin* > *Remove*. A confirmation dialog box opens.
3. Click the *Yes* button to confirm removing the monitor. A second confirmation dialog box opens.
4. Click one of the following buttons to determine what happens to the archived event data for the monitor.
   - Click the *Yes* button to delete all collected data.
   - Click the *No* button to leave the archived data intact.

   The monitor disappears from the Explorer pane.

4.8 Forwarding Events to Non-Operations Center Subscribers

Each Experience Manager Monitor can forward events to non-Operations Center subscribers in XML or IBM* Tivoli* Enterprise Console T/EC format. To implement this, edit the settings in the `/OperationsCenter_ExperienceManager_install_path/config/monitor.properties` file.

- Section 4.8.1, “Storing and Forwarding Events,” on page 39
- Section 4.8.2, “Forwarding T/EC Events from the Experience Manager Monitor,” on page 40

4.8.1 Storing and Forwarding Events

Store events or forward them to non-Operations Center subscribers in XML or T/EC format. Filter event information based on column name.

The following code shows a forwarded T/EC event:

```
### EVENT ###
ICMP;
eventID="-694373732";
createDate="2004-09-13 13:09:57.415";
alarmSeverity="OK";
responseTime="4";
description="Success: Response Time = 4ms";
eventType="synIcmpAlarm";
appName="test1";
hostName="boilermakers.mosol.com";
hostPort="6789";
scenarioName="002-ping";
icmpSynHost="www.yahoo.com";
icmpSynMaxCount="5";
END
### END EVENT ###
```

The following code shows a forwarded XML event:

```
<event eventID="-1069746093"
createDate="2004-09-13 13:10:59.054"
alarmSeverity="OK"
responseTime="28"
description="Success: Response Time = 28ms"
eventClass="TRACERT"
eventType="synTracertAlarm"
appName="test1"
hostName="boilermakers.mosol.com"
hostPort="6789"
scenarioName="003-tr"
tracertSynHost="www.yahoo.com"
tracertSynMaxCount="30"
/>
```

To set up event storage or forwarding:

1. Modify the */OperationsCenter_ExperienceManager_install_path*/config/monitor.properties file. Table 4-2 describes the various event forwarding properties and the valid values.

Table 4-2 *Event Forwarding Properties*

Property	Description
Event.forwarding.output	If True, forward events using Event.forwarding.format, in addition to the Experience Manager events sent to the Operations Center server. Specify True or False.
Event.forwarding.format	The format used for storage or transmission. Specify XML or TEC.
Event.forwarding.type	Specifies whether the monitor stores or sends the event. Set it to TRANSMIT or STORE.
Event.forwarding.outputfile	If Event.forwarding.type equals STORE, specify the file format for storing events: TEC or XML. For example, d:/*OperationsCenter_ExperienceManager_install_path*/events/tec_file.txt.

Property	Description
Event.forwarding.host	If Event.forwarding.type equals TRANSMIT, specify the hostname to which the events are transmitted.
Event.forwarding.port	If Event.forwarding.type equals TRANSMIT, specify the port number on the host to use for transmission.
Event.forwarding.filterout	Event-specific data used to filter the event before forwarding. Use a semicolon to delimit values when specifying more than one filter. Refer to the Experience Manager Hierarchy file in the */OperationsCenter_install_path/*database/examples directory for filter examples. The following core fields cannot be filtered: eventide, createDate, alarmSeverity, responseTime, description, eventType, appName, hostname, and hostPort. For example, to filter out the HTTP code and bytes from all HTTP tests: Event.forwarding.filterout=httpSynCode;httpSynBytes

4.8.2 Forwarding T/EC Events from the Experience Manager Monitor

When the Experience Manager Monitor directly forwards events to IBM Tivoli Enterprise Console (T/EC) servers, carriage returns (<CR>) and tab characters appear after the $ character. This can cause the T/EC reception plug-in for IBM Micromuse* Netcool*/OMNIBUS to improperly map alarm properties to their alarms. The default hard-coded header and footer sequences match the following:

```
Event.forwarding.TEC.header=\\n### EVENT ###\\n
Event.forwarding.TEC.footer=END\\n### END EVENT ###\\n
```

To correct this, modify the following properties in the */OperationsCenter_ExperienceManager_install_path/*config/monitor.properties file. Configure both properties with embedded line feed (0x0A) characters using the sequence \\n.

- **Event.forwarding.TEC.header:** Specifies the header data sent at the beginning of a TEC event.
- **Event.forwarding.TEC.footer:** Specifies footer data sent at the end of a TEC event.

4.9 Troubleshooting the Experience Manager Monitor

Heap dump and thread dump utilities are available as WebOps via default settings in the */OpeationsCenter_ExperienceManager_install_path/*config/monitor.properties file.

For more information about WebOps, see Section 12.3, “Invoking Scripts through the Experience Manager-Monitor Web Server (WebOps),” on page 134.

To diagnose Experience Manager Monitor problems:

1. Open a Web browser.
2. To dump heap to a file in the /bin directory, go to:

   monitor_Web_Server_Address/Plugins/invoke.pl?op=heapDump

   The full file path is returned to the browser.
3. To dump threads to the browser, go to:

   monitor_Web_Server_Address/Plugins/invoke.pl?op=threadDump

5 Creating Test Scenarios

Test scenarios include accessing and performing actions on multiple Web sites, logging into e-mail, accessing FTP sites, and performing more network-centric tasks such as ICMP, traceroute or nslookup transactions. The Experience Manager adapter provides the ability to reuse and share scenarios among test definitions.

This section explains the different types of test scenarios and how to create shared scenarios for use in synthetic tests:

5.1 Creating Test Scenarios

A Experience Manager Monitor executes tests based on one or more scenarios. A scenario executes a specific function by using protocols including HTTP, HTTPS, DHCP, FTP, LDAP, SMTP/POP3, TRACEROUTE, ICMP or DNSLOOKUP.

5.1.1 General Test Scenario Definition Steps

There are certain general steps to define a test scenario:

1. Create a scenario.
2. For each scenario, specify the test attributes based on a scenario type.
3. Specify standard attributes to apply to each scenario.
4. Add the scenarios to test definitions.

   See Chapter 7, “Creating Synthetic Tests Using the HTTP Recorder,” on page 97 for instructions on defining tests.

5.1.2 Test Scenario Types

Table 5-1 lists the available scenario types and the types of actions each scenario performs.

Table 5-1 *Test Scenario Types*

Scenario Type	Description
DHCP	Receives and sends Dynamic Host Configuration Protocol (DHCP) offer requests to the DHCP server based on a Media Access Control (MAC) Address.
DNS Lookup	A Domain Name Service (DNS) test. Translates domain names to the corresponding IP addresses.
E-Mail	Records the length of time required to send an e-mail message to a mail server. Select POP3 or IMAP Inbox server types. The mail test deletes the e-mail message after performing the test.
FTP	Records the time used to upload or download a specified file to/from the File Transfer Protocol (FTP) server and then deletes the uploaded/downloaded file. Uses FTP to communicate with a specific device and records the length of time required to acknowledge that communications successfully opened and the file was transferred.
HTTP(s)	Records the time required to access and download the contents of a Web page. Uses HyperText Transfer Protocol (HTTP), which is the underlying protocol used by the World Wide Web, to access Web servers and records the response times for retrieving and receiving a specific Web page, or posting data to it.
ICMP	This ping test uses Internet Control Message Protocol (ICMP) to test if an Internet connection is active.
LDAP	Performs user authentication (validates user name and password) to test the availability of the LDAP directory server.
SQL	Records the response time required to perform a database query. The SQL test performs SQL SELECT statements against a database. It is possible to define the actual result set for the query to match. Measures the response time of the executed query from start to finish for historical trending. In addition, can gather and promote each selected value to alarm properties (or element properties), for use in trending and analysis.
Third Party	Enables developers to write tests that are executed by the Experience Manager Monitor's test engine.
Trace Route	Traces a packet from a computer to an Internet host, showing how many hops the packet required to reach the host and how long each hop took. If visiting a Web site and pages display slowly, use trace route to identify where the longest delays occur. The test records the time and number of hops that it takes to reach the specified host machine. It also records the address of each hop that it makes reroute.

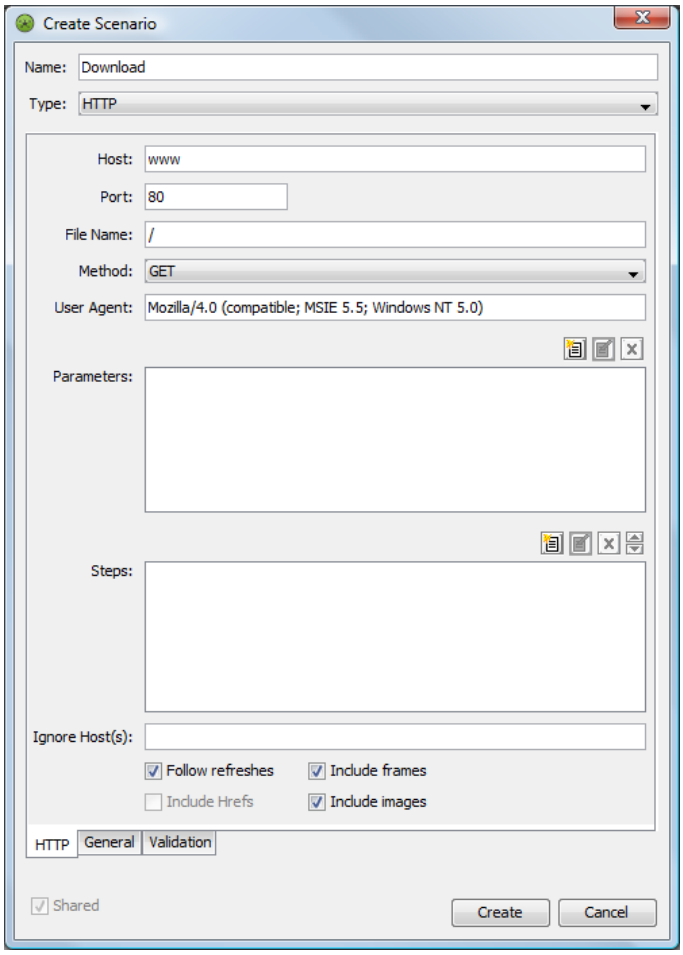
5.2 Shared Scenarios

The Experience Manager adapter provides the ability reuse and leverage scenarios among test definitions. Create these shared scenarios independently of test definitions.

To create a shared scenario:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration*.
2. Right-click *Shared Scenarios*, then click *Create Scenario*.

   The Create Scenario dialog box opens.



3. In the *Name* text box, specify the scenario name.
4. Select the *Type* drop-down list and select the s type.

   The scenario tab name updates to match the selected test type. (For example, the *HTTP* tab in the previous figure.)

5 Specify the appropriate test attributes in the scenario type tab.

(For example, the *HTTP* tab in the previous figure).

Scenario Tab	Additional Reference for Attributes
FTP	See Section 5.5.1, “FTP Scenario Attributes,” on page 48.
DHCP	See Section 5.5.2, “DHCP Scenario Attributes,” on page 48.
HTTP(s)	See Section 5.5.3, “HTTP(S) Scenario Attributes,” on page 49, Section 5.5.4, “HTTP(S) Parameters,” on page 50, and Section 5.5.5, “Handling Dynamic Web Pages,” on page 53.
ICMP	See Section 5.5.6, “ICMP Scenario Attributes,” on page 60.
LDAP	See Section 5.5.7, “LDAP Scenario Attributes,” on page 60.
E-Mail	See Section 5.5.8, “E-Mail Scenario Attributes,” on page 60.
DNS Lookup	See Section 5.5.9, “DNS Lookup Scenario Attributes,” on page 61.
SQL	See Section 5.5.10, “SQL Scenario Attributes,” on page 61.
Third Party	See Section 5.5.12, “Third-Party Scenario Attributes,” on page 65.
Trace Route	See Section 5.5.11, “Trace Route Scenario Attributes,” on page 64.

6 Click the *General* tab.

See Section 5.3, “Specifying General Test Scenario Attributes,” on page 45 for details on editing this tab.

7 Click the *Validation* tab.

See Section 5.4, “Specifying a Response Time,” on page 46 for details on editing this tab.

8 Click the *Create* button to create the new test scenario.

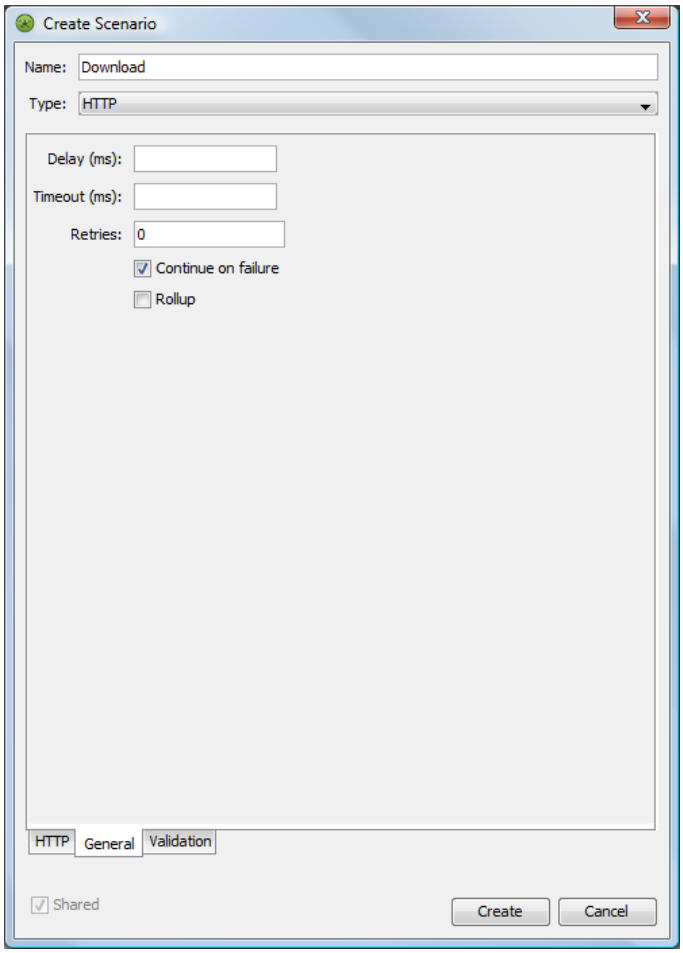
The new scenario displays beneath the Shared Scenarios element.

Scenarios are added to test definitions. See Chapter 6, “Creating and Deploying Synthetic Tests,” on page 71 for details.

5.3 Specifying General Test Scenario Attributes

To specify general test scenario attributes:

1 Click the *General* tab in the Create Scenario dialog box to display the *General* tab:



2 Fill in the following fields:

Delay (ms): The length of time in milliseconds to wait before the scenario executes. If blank, there is no delay.

Retries: The number of attempts (greater than zero) to test before failing. If the test succeeds within this count, there is no failure.

Continue on failure: Select this check box to allow the test to continue executing the next scenario if the current scenario fails for any reason. If this check box is cleared and the scenario fails, all subsequent scenario events on the Operations Center server have an UNKNOWN condition and a note indicating the scenario did not execute because of the failure of the scenario.

Rollup: Select this check box to aggregate all scenarios prior to the current scenario and send a roll-up response time event.

5.4 Specifying a Response Time

An alarm can occur based on the response time in a scenario. Specify the acceptable time range for a response in the *Validation* tab in the Create Scenario dialog box.

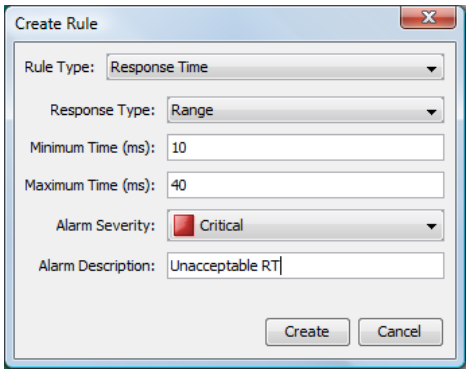
To set rules for response time validation:

1. Click the *Validation* tab in the Create Scenario dialog box to display the *Validation* tab.
2. Click the *New Rule* icon to open the Create Rule dialog box.
3. Click the *Rule Type* drop-down list and select *Response Time* if it is not already selected.
4. Click the *Response Type* drop-down list and select a response type:

   Range: Generates an alarm when the response time is within a specified time range. Specify the time range by using the *Minimum* and *Maximum Time* (in milliseconds) text boxes. Also select the *Alarm Severity* drop-down list and select a severity to set for the alarm issued when the response time is within the specified time range.

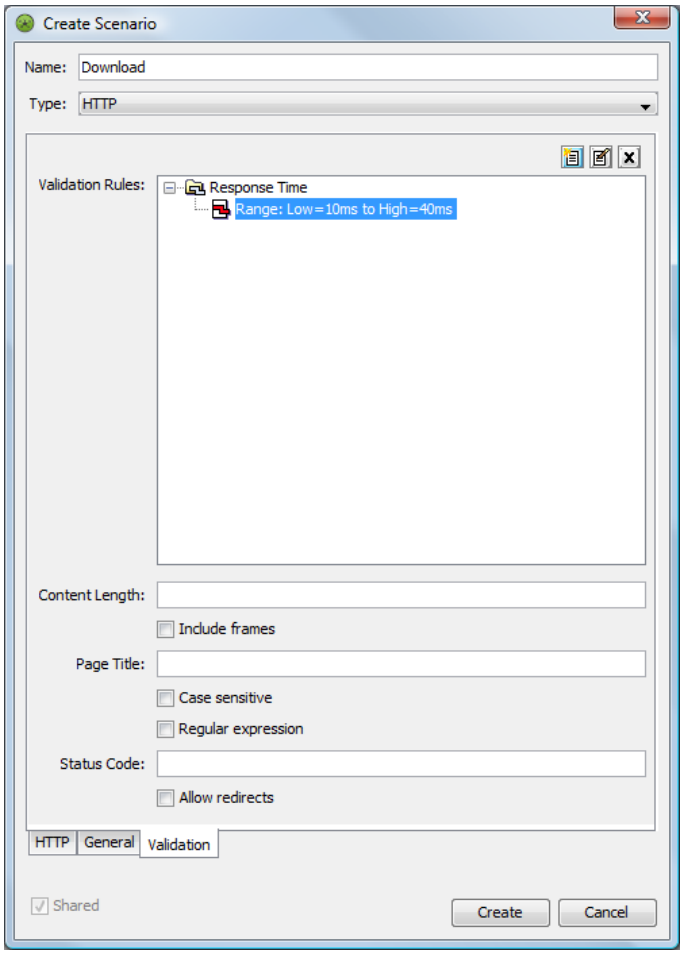
   Upper Threshold: Generates an alarm when the response time exceeds the specified maximum time interval.

   Lower Threshold: Generates an alarm when the response time is less than the specified minimum time interval.
5. If the Response Type is Lower or Upper Threshold, select the *Alarm Severity* drop-down list and select a severity to set for the alarm issued when a response time rule is violated.
6. In the *Alarm Description* text box, type a description for the alarm:



7 Click the *Create* button to save the rule.

The rule displays in the *Validation* tab:



5.5 Specifying Scenario-Specific Attributes

The attributes available vary for each type scenario. The attributes display in the Create Scenario dialog box, in the third tab whose name matches the selected scenario type, such as FTP or HTTP. This section explains the attributes for each scenario that display in the tab:

5.5.1 FTP Scenario Attributes

Table 5-2 describes the FTP scenario attributes.

Table 5-2 *FTP Scenario Attributes*

Attribute	Specify
Host	Name of the FTP server to test.
File Name	File to upload or download from the FTP server. Precede the file name with the “/” (forward slash) character. Date macros can define File Name values. See Section 5.5.13, “Using Date Macros within Scenarios,” on page 67 for more information.
Method	Method used to interact with the FTP server.
	GET (uploads the selected file from the FTP server).
	PUT (downloads the selected file to the FTP server).
Type	Type of transmission that occurs: BINARY or ASCII.
Socks Host	SOCKS proxy hostname.
Socks Port	Port number on which SOCKS proxy host listens.

5.5.2 DHCP Scenario Attributes

Table 5-3 describes the DHCP scenario attributes.

Table 5-3 *DHCP Scenario Attributes*

Attribute	Specify
MAC Address	Address that network adapters use to uniquely identify themselves on a LAN. MAC addresses are 12-digit hexadecimal numbers.
Server IP	IP address of the server. If the field is left blank, 255.255.255.255 is used as the default broadcast address.
Port	Port used by the server for DHCP broadcasts. If the field is left blank, port 67 is used.

5.5.3 HTTP(S) Scenario Attributes

The Hrefs attribute (Web pages that are linked from the host Web page and show statistics) are no longer supported in Experience Manager. Table 5-4 describes HTTP and HTTPS scenario attributes.

Table 5-4 *HTTP and HTTPS Scenario Attributes*

Attribute	Specify
Host	Web address to test.
Port	Port used by the Web server for HTTP requests. Defaults to 80 for HTTP tests. Defaults to 443 for HTTPS tests.
File Name	File to access on the Web site (that is, the specific Web page). Precede the file name with the "/" (forward slash) character. Date macros can define File Name values. See Section 5.5.13, "Using Date Macros within Scenarios," on page 67 for more information.
Method	Method used to interact with the Web page: **GET:** A request for a file **POST:** Send text to the Web page
User Agent	User-defined string that is inserted into the browser "user agent" informational text area. Defaults to Experience Manager Monitor. Often used to identify the source of a transaction as a synthetic test rather than a real user.
Parameters	Parameter definitions that do the following: ◆ Add customized header information into the HTTP request ◆ Define a unique piece of HTML response data to capture and use in forms for subsequent test scenarios ◆ Specify form values ◆ Post raw data. See Section 5.5.4, "HTTP(S) Parameters," on page 50.
Steps	Instructions that allow a scenario to crawl through a set of Web pages by extracting the URL link for a specific HREF, REFRESH, FRAME, and/or FORM. See Section 5.5.5, "Handling Dynamic Web Pages," on page 53.
Ignore Hosts	Enter one or more hostnames, separated by commas, to ignore during test execution. Partial hostnames are acceptable. For example, double-click matches doubleclick.net and ads.doubleclick.com. This field applies only to HTTP assets referenced by the page (that is, images included on the page) that are not explicitly named in the scenario or its steps. Use this feature to exclude double-click ads that appear in Web pages. Excluding hosts that the Web engine might execute against avoids making calls that impact response time measurements for the double-click ads. However, it does include all other assets to which a Web page refers.
Follow refreshes	Select the check box to have the test follow any page containing a REFRESH HTTP-EQUIV `<meta>` tag to redirect the Web browser to a different URL.
Include frames	Select the check box to include all frame pages that are part of the Web page and display statistics for these in the *Assets* tab of the test record (alarm).
Include images	Select the check box to include all images that are a part of the Web page and display statistics for these in the *Assets* tab of the test record (alarm).

5.5.4 HTTP(S) Parameters

Adding parameters to HTTP(S) scenarios enables the following customizations:

Adding Header Information

An HTTP transaction consists of headers that specify information such as the action required of the server, the type of data being returned, or a status code.

Use the Add Header parameter to add customized header information in the HTTP request or manipulate existing HTTP headers by overriding values. For example, setting name to Accept-Transfer-Encoding and value to None informs the Web server that no transfer encoding is necessary.

Another use for the header information is to arbitrarily add cookies to a synthetic session by adding cookies explicitly via the *Add Header* option.

Data Capture

Data capture defines a unique piece of HTML response data or header date that a user might want to capture and use dynamically for form field data in subsequent scenarios for an HTTP test. If the name of the data capture is in variable format (surrounded by curly braces {}), then the captured data is stored in a variable and is available for the remainder of the test.

Form Fields

Specify form fields for an HTTP test scenario to add form data for a URL request. Web site forms receive the information required to properly access the page and run the test.

For example, to search for content on a specific page, first log in to access the page. Use form fields to have the test log in before performing other tests on the site.

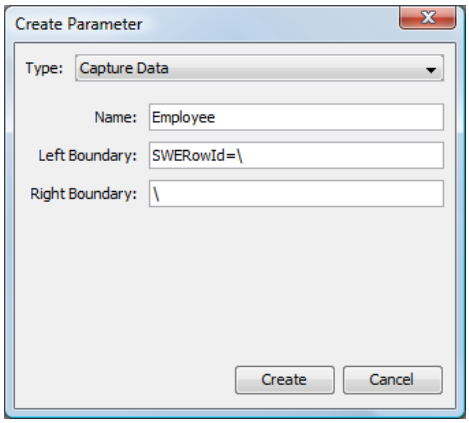
Post Raw Data

When specifying an HTTP POST, you can post form field data to the server or post it as raw data. Normally, XML data is URL-encoded, which a server can not process. The Post Raw Data feature enables the Experience Manager Monitor to send XML data to servers that require an XML string as the post data.

Defining a Parameter for the HTTP(S) Test Scenario

To define a parameter for the HTTP(S) test scenario:

1 In the Create Scenario dialog box, in the *HTTP(S)* tab, click the  *New Parameter* icon to open the Create Parameter dialog box:



2 Select the *Type* drop-down list and select the type of action to be performed.

The following table provides explanations of each parameter type:

Parameter Type	Function	Next Steps
Add Header	Adds customized header information, manipulates existing HTTP headers by overriding values or adds cookies.	◆ Type a name for the header in the *Name* text box. ◆ Type the value for the header in the *Value* text box. ◆ Add cookies explicitly by typing `Set-Cookie:` (ending colon is required) in the *Name* text box and setting the value pair in the *Value* text box (for example, `myname=David`).
Capture Data	Captures data.	◆ Type the name for the data capture in the *Name* text box. ◆ To define how the test searches for data to capture, specify the `left_boundary` and `right_boundary`. For example, if `left_boundary="SWERowId=\"` and the `right_boundary="\"` and if any of the HTML response data from within all scenarios of a test contains `[SWERowId=\"1-999\"]`, the data captured is `[1-999]`. ◆ When a marked `form_field` or cookie with `use_capture_data` contains this value, a dynamic substitution occurs when running the scenarios.
Post Raw Data	Posts raw data.	◆ In the *Data* text box, specify the raw data to send. ◆ Use date macros to define values for posting raw data. See Section 5.5.13, “Using Date Macros within Scenarios,” on page 67 for more information. ◆ Select the *Data is XML* check box if the data is in XML format.

Parameter Type	Function	Next Steps
Submit Form Field	Submits a form field.	◆ Specify the *Name* and *Value* that the Web page form field expects. ◆ Date macros can define values for form fields. See Section 5.5.13, "Using Date Macros within Scenarios," on page 67 for more information. ◆ Select the *URL encoded* check box if the value is already URL-encoded. ◆ Select the *Use data capture* check box to replace the value with the captured data. If this option is selected, the *Value* specified above must be the name of the captured data. ◆ Select the *Use cookie data* check box to have the form field value contain the value for the cookie. If selected, the *Value* specified above must be the name of the cookie. ◆ If both *Use data capture* and *Use cookie data* are selected, the test engine attempts to fill the dynamic value from the cookie and then from the captured data store. ◆ Select the *Value is Password* check box to encrypt the form field because it is a user password field.

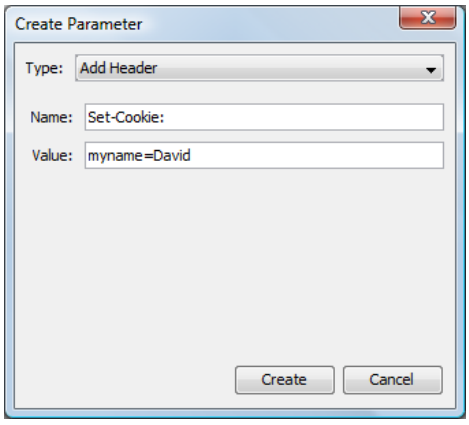
3 Click the *Create* button.

The Create Parameter dialog box closes and the new parameter displays in the *Parameter* list.

Adding Header Information

To add header information:

1 In the Create Parameter dialog box, select the *Type* drop-down list, then click *Add Header*:



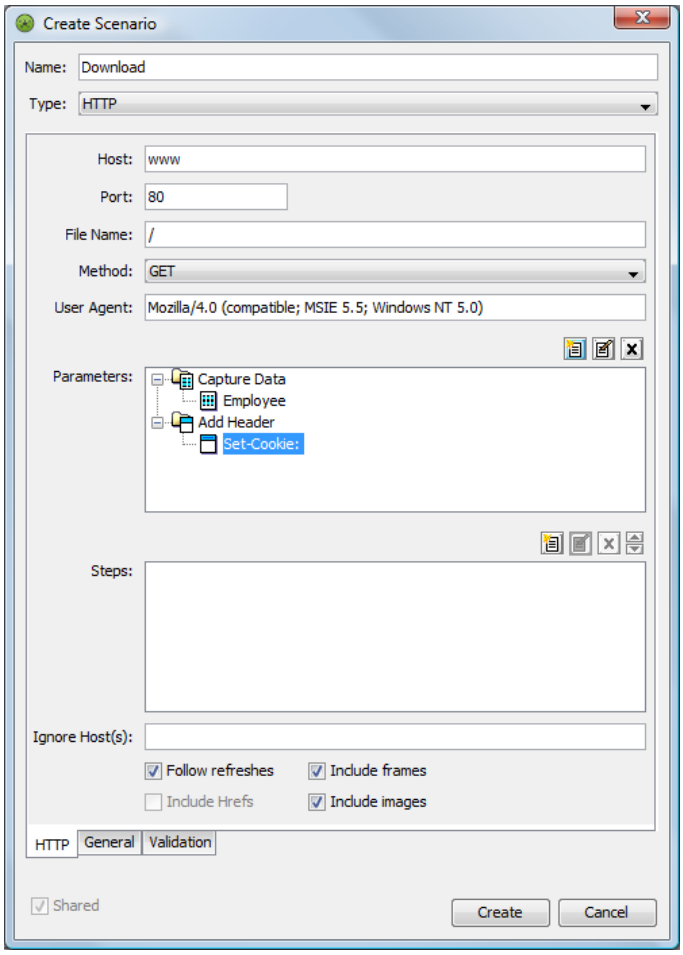
2 Type the header name in the *Name* text box.

3 Type the header value in the *Value* text box:

- Add cookies explicitly by typing `Set-Cookie:` (ending colon is required) in the `Name` text box and setting the value pair in the Value text box (for example, `myname=David`).
- Use date macros to define values for the header. See Section 5.5.13, "Using Date Macros within Scenarios," on page 67 for more information.

4 Click the *Create* button.

The Create Parameter dialog box closes and the new parameter displays in the *Parameters* list in the Create Scenario dialog box:



5.5.5 Handling Dynamic Web Pages

To perform multiple HTTP tests under one scenario, use step tags. The steps instruct a scenario to crawl through a set of Web pages by extracting the URL link for a specific HREF, REFRESH, FRAME, FORM-GET, or FORM-POST.

Steps define a series of sequential actions where each relies on the previous. When a step fails, then all remaining steps are not executed. Step tags apply to the current scenario only, as they are performed separate from and without impact to the next scenario.

For example, assume an HTTP scenario performs a search for "NetIQ" on Google. The engine parses the HTML response and searches for the HREF link of "NetIQ" and then proceeds to www.netiq.com.

Additionally, it is possible to validate the following aspects of a Web page:

- The title of the Web page
- The length of the Web page's contents
- The Web page's return code
- Specific content within the Web page

To set up dynamic Web pages:

- "Setting Step Tags for the HTTP(S) Test Scenario" on page 54
- "Specifying Response Time Validation for the Step" on page 55
- "Specifying Rules for Cookie or Content Match Validations" on page 58

Setting Step Tags for the HTTP(S) Test Scenario

To set step tags for the HTTP(S) test scenario:

1. In the Create Scenario dialog box, in the *HTTP(S)* tab, click the *New Step* icon. The Create Step dialog box opens.
2. Type a step name in the *Name* text box.
3. Select the *Type* drop-down list and select a step type:

   Href: Finds its URL from `<a href="url">`. In the *Reference* text box, specify the name of the link on the Web page. For example, the reference is `<b>NetIQ</b>` and the URL is http://www.netiq.com if the HTML code is:

   ```
   <a href="http://www.netiq.com"><b>NetIQ</b></a>
   ```

   The reference must include all syntax between the `<a>` and `</a>` tags.

   Frame: Finds its URL from `<frame src="url">`. In the Reference text box, specify the `src` value of the frame tag. For example, the reference and URL are `overview-frame.html` if the HTML code is:

   ```
   <FRAME src="overview-frame.html" name="packageListFrame">
   ```

   Refresh: Finds its URL from `<meta http-equiv="refresh" content="X;URL=url …>`. In the Reference text box, specify the URL portion of the content value of the `<meta>` tag. For example, the reference and URL are `mlb_scoreboard.jsp?ymd=20030812` if the HTML code is:

   ```
   <meta http-equiv="refresh" content="180;URL=mlb_scoreboard.jsp?ymd=20030812">
   ```

   form-get: Finds its URL from `<form action="url">`.

   form-post: Posts its URL to `<form action="url">`. Specify a value in the Reference text box.

   For form-get and form-post, specify the reference as the name of the *Submit* button of the corresponding input `type=submit` tag. For example, the reference is Google Search and the URL is the search from the form action, if the HTML code is:

   ```
   <form action="/search" name=f>
   <input type=submit value="Google Search" name=btnG>
   </form>
   ```

   Define the form_field values in the parameter definitions. See Section 5.5.4, "HTTP(S) Parameters," on page 50.
4. Create parameters by clicking the *New Parameter* icon to open the Create Parameter dialog box.
5. Select the *Type* drop-down list and select a parameter type:

   Capture Data: Specify the name for the data capture in the *Name* text box, and the left and right boundaries in the *Left Boundary* and *Right Boundary* text boxes.

   Submit Form Field: Specify the form name in the *Name* text box and an optional value in the *Value* text box. Select the check boxes for *URL encoded*, *Use data capture*, *Use cookie data*, and/or *Value is password*.

Post Raw Data: Specify the data in the *Data* text box. Select the *Data is XML* check box if the data is XML code.

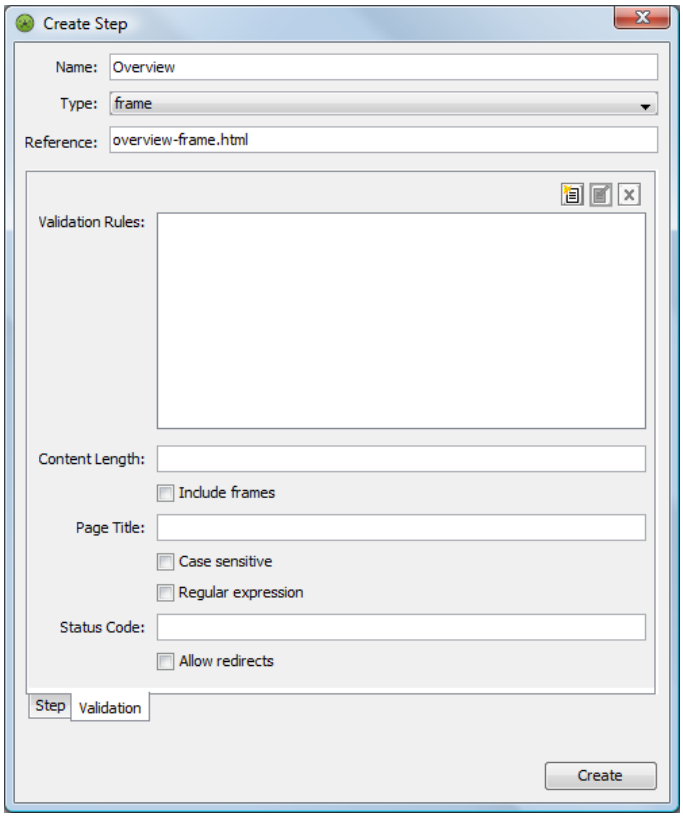
6 To save the parameter, click the *Create* button.

The Create Parameter dialog box closes and the new parameter displays in the *Parameter* list.

Specifying Response Time Validation for the Step

To specify response time validation for the step:

1 In the Create Step dialog box, click the *Validation* tab to display the *Validation* tab:



2 Click the *New Rule* icon to open the Create Rule dialog box.

3 Select the *Rule Type* drop-down list, then click *Response Time*.

Response time is the default for all scenarios except HTTP and HTTPS.

4 Select the *Response Type* drop-down list and select one of the following:

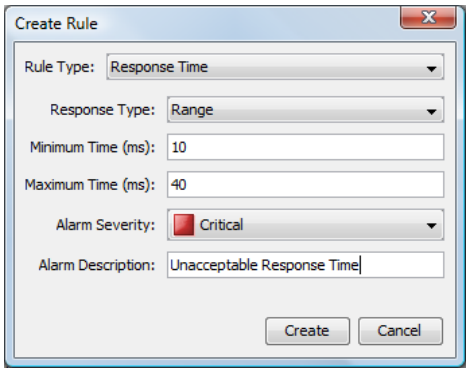
Range: Generates an alarm when the response time is within the specified time range. Specify the time range using the *Minimum Time* and *Maximum Time* (in milliseconds) text boxes. Select the *Alarm Severity* drop-down list and select a severity to set for the alarm issued when the response time is within the specified time range.

Upper Threshold: Generates an alarm when the response time exceeds the specified maximum time interval.

Lower Threshold: Generates an alarm when the response time is less than the specified minimum time interval.

5 If the Response Type is Lower or Upper Threshold, select the *Alarm Severity* drop-down list and select a severity to set for the alarm issued when a response time rule violation occurs.

6 In the *Alarm Description* text box, specify a description for the issued alarm:



7 Click the *Create* button to create the rule.

The rule displays in the *Validation Rules* list in the *Validation* tab.

8 In the *Validation* tab, provide information for the following HTTP content that returns from Web server and can be validated:

Content Length: An element that defines the HTML document's content length. Select the *Include frames* check box to calculate the content length for all frames including main HTML document. The content length does not include images or other types of application/binary data.

Page Title: An element that defines the HTML document's title. Select the *Case Sensitive* check box if the match must be case sensitive. The test fails if the value does not match.

Select the *Regular Expressio*n check box if case does not matter. The regular expression entered is used to validate the syntax.

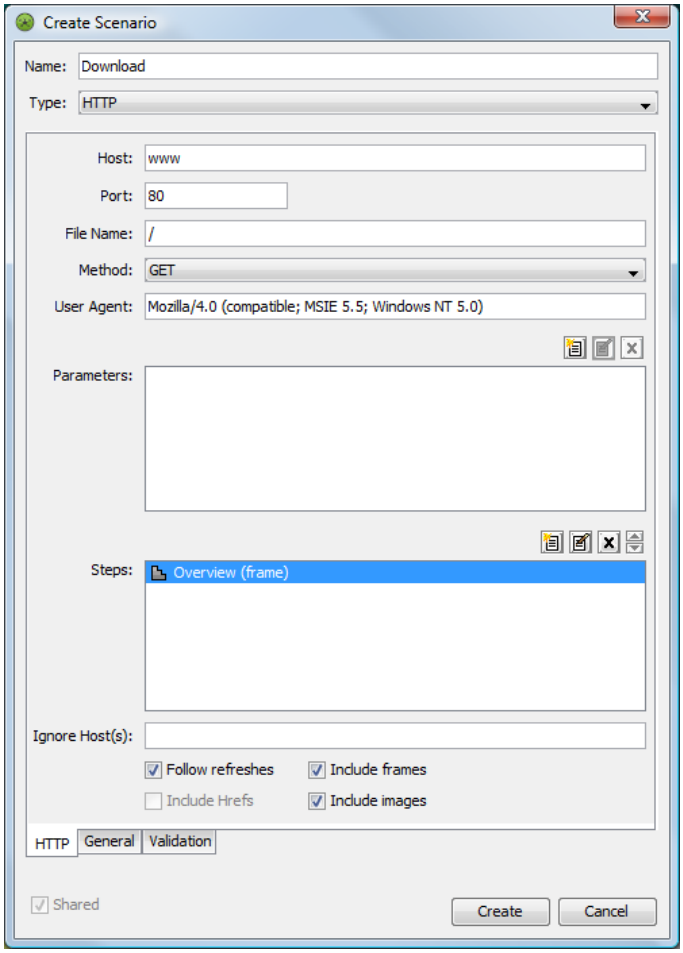
Status Code: An element that defines the HTML document's return status code. Type the valid HTTP response code. The test fails if the codes do not match.

Select the *Allow redirects* check box to allow page redirects without failing the test. For example, assume the HTTP code is set to 200 and this check box is selected. The test does not fail if, through the gathering of URL assets, there are one or more redirects that have a code of 302.

9 Click the *Create* button to create the step.

The Create Step dialog box closes and the step definition displays in the *Steps* list in the Create Scenario dialog box.

10 Because steps occur in the order in which they display, you can click the *Move Up* and *Move Down* icons to reorder selected steps:



11 Click the *Create* button to create the scenario.

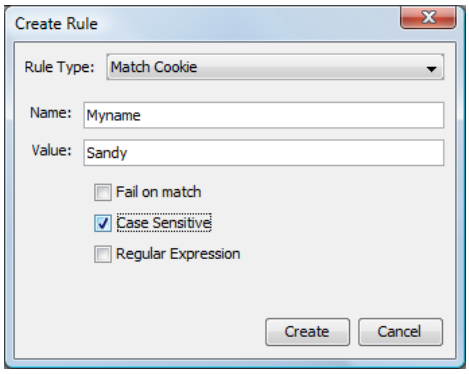
The new scenario displays under the Shared Scenarios element in the Explorer pane.

Specifying Rules for Cookie or Content Match Validations

When defining HTTP and HTTPS test scenarios, it is possible to define rules for performing specific actions based on cookie or content matches.

To specify rules for cookie or content match validations:

1. In the Create Scenario dialog box, click the *Validation* tab to display the *Validation* tab.
2. Click the *New Rule* icon to open the Create Rule dialog box:



3. Select the *Rule Type* drop-down list and select one of the following:
   - To validate cookies received from Web pages, select *Match Cookie*.
   - To validate page content, select *Match Content*.
4. In the *Name* text box, type the page field name or cookie field name.
5. In the *Value* text box, type the cookie or content value.
6. Select the following check boxes as needed:

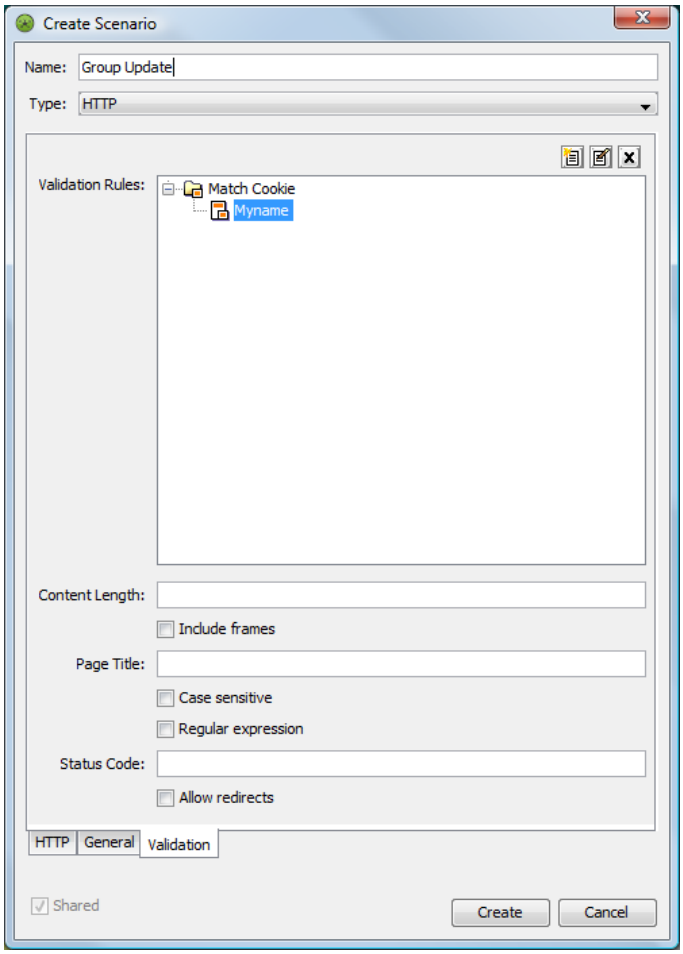
   Fail on match: Causes the scenario to fail if the rule content is matched.

   Case Sensitive: Specifies that matching is case sensitive. Clear the check box if the value specified is a regular expression.

   Regular Expression: Identifies the value specified as a regular expression.

7 Click the *Create* button to create the rule.

The rule displays in the *Validation* tab:



8 In the *Validation* tab, type information for the following HTTP content that returns from Web server and can be validated:

Content Length: An element that defines the HTML document's content length. Select the *Include frames* check box to calculate the content length for all frames including main HTML document. The content length does not include images or other types of application/binary data.

Page Title: An element that defines the HTML document's title. Select the *Case Sensitive* check box if the match must be case sensitive. The test fails if the value does not match.

Select the *Regular Expression* check box if case does not matter. The regular expression entered validates the syntax.

Status Code: An element that defines the HTML document's return status code. Type the valid HTTP response code. The test fails if the codes do not match.

Select the *Allow redirect*s check box to allow page redirects without failing the test. For example, assume the HTTP code is set to 200 and this check box is selected. The test does not fail if, through the gathering of URL assets, there are one or more redirects that have a code of 302.

9 Click the *Create* button to create the scenario.

The new scenario displays under the Shared Scenarios element in the Explorer pane.

5.5.6 ICMP Scenario Attributes

Table 5-5 describes the ICMP scenario attributes.

Table 5-5 *ICMP Scenario Attributes*

Attribute	Specify
Host	Machine to echo/ping.
Count	Number of echo requests to send.

5.5.7 LDAP Scenario Attributes

Table 5-6 describes the LDAP scenario attributes.

Table 5-6 *LDAP Scenario Attributes*

Attribute	Specify
URL	Location and port of the LDAP server to which the Operations Center server connects.
Base DN	Distinguished name for the root naming context. For example, if the server has the root naming context `dc=example, dc=com`, then set the basedn property to this value.
Initial Context	Starting context for performing LDAP extended operations and controls. If it is left blank, the default is `com.sun.jndi.Ldap.LdapCtxFactory`.
Authentication	Security level as follows: ◆ **None:** No user authentication is performed. ◆ **Simple:** Simple credentials are required. ◆ **Strong:** Strong credentials are required. If it is left blank, the default is Simple.
Principal	Identity of the principal for authenticating the caller to the service. The format of this property depends on the authentication scheme. The user name and password in an associated authentication identity overwrite any value entered for this property.
Credentials	Credentials of the principal for authenticating the initial context of the service. The value of this property depends on the authentication scheme. For example, the value of the authentication scheme might be a hashed password, cleartext password, key, or certificate. The user name and password in an associated authentication identity overwrite any value entered for this property.
Security Protocol	Security protocol to use.
Organizational Unit	Organizational unit under the base DN where the user entries reside.

5.5.8 E-Mail Scenario Attributes

Table 5-7 describes the e-mail scenario attributes.

Table 5-7 E-Mail Scenario Attributes

Attribute	Specify
SMTP Server	Address of the mail server used to send mail.
Inbox Server Type	Type of e-mail server. Select POP3 or IMAP.
Inbox Server	Address of the mail server used to receive mail.
Mail Address	Actual e-mail user destination address.
Message Body Size	The size in KB of the e-mail message. Defaults to 20.
Require SMTP Authentication	Select the check box if SMTP authentication is required when running the e-mail test.

An e-mail test scenario attempts to send an e-mail to a specified mail address. If it is successful reading the e-mail from the recipient's in box, Experience Manager deletes the e-mail.

If the test fails to read and delete the e-mail, there are two probable causes:

- **Authentication problem:** Invalid credentials always prevent the reading and deletion of an e-mail. Verify that the scenario uses a valid set of user credentials associated with the destination mailbox.
- **Client conflict:** A thick e-mail client accesses the same user account specified in the e-mail test scenario and removes the e-mail from the server before it is read and deleted by Experience Manager. Verify that another client does not use the same e-mail address.

5.5.9 DNS Lookup Scenario Attributes

Table 5-8 describes the DNS lookup attributes.

Table 5-8 DNS Lookup Scenario Attributes

Attribute	Specify
Query	Hostname or IP address to resolve.
Name Server	Name (or IP address) of the DNS server.
DNS port	DNS lookup port. Defaults to 53.
Transport	Transports the specified protocol. The default is UDP. You can set it to TCP.
Enable reverse lookup	Select the *reverselookup* check box to invert the address, append `in-addr.arpa`, and look up the PTR (pointer) data instead.
Enable recursion	Select the *recursion* check box to ask for a recursive answer to the query. Defaults to a selected check box.

5.5.10 SQL Scenario Attributes

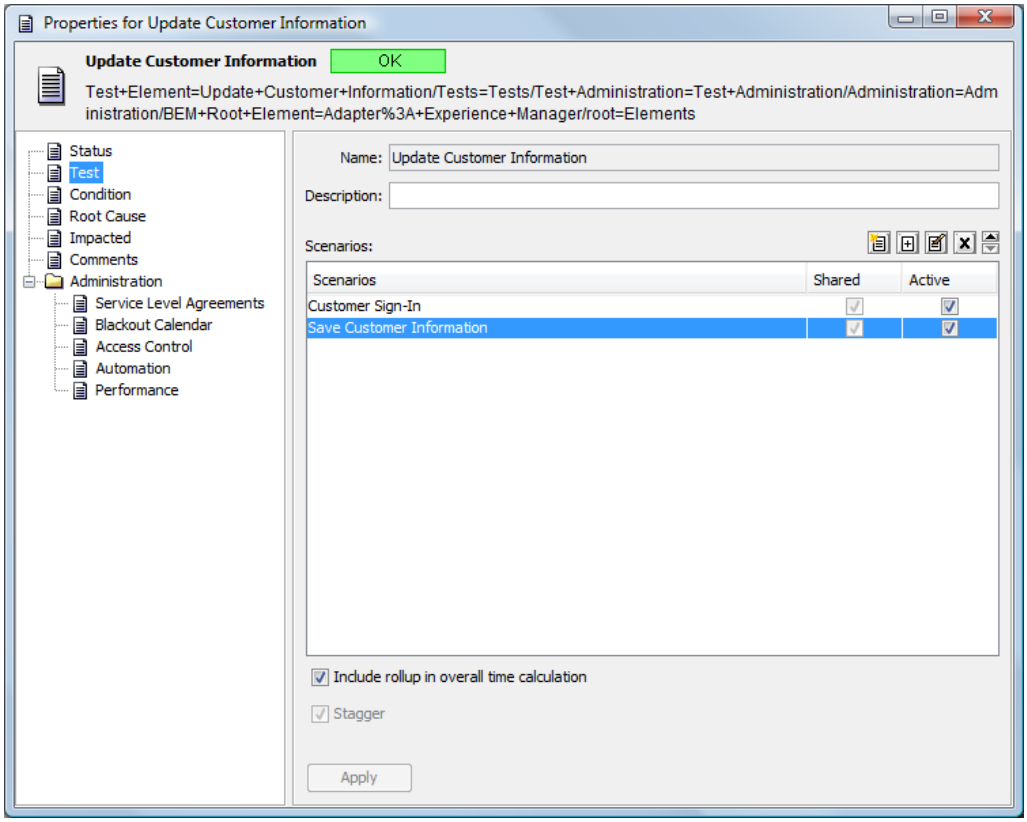
Table 5-9 describes the SQL scenario attributes.

Table 5-9 *SQL Scenario Attributes*

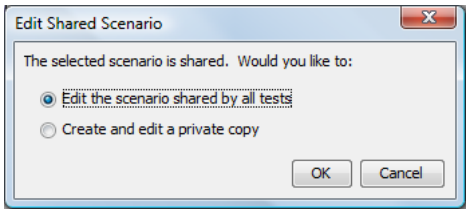
Attribute	Specify
Host	Name of the host server.
Query	Actual query statement to execute. Use date macros to define values for the query. See Section 5.5.13, "Using Date Macros within Scenarios," on page 67 for more information.
DB Type	Type of database. Specify oracle, sybase, db2, or mssql7.
DB Name	Name of the database.
DB Port	Port number used by the database.

To define a result set to match the SQL test:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration* > *Tests*.
2. Right-click an SQL test, then click *Properties* to open the Status property page.
3. In the left pane, click *Test* to open the Test property page:



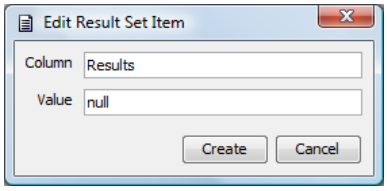
4. Select a scenario, then click the *Edit Scenario* icon to open the Edit Scenario dialog box. The Edit Shared Scenario dialog box displays:



5 Do one of the following:

- Select the *Edit the scenario shared by all tests* radio button to edit the shared scenario definition.
- Select the *Create and edit a private copy* radio button to modify the scenario without affecting all tests that use it.

6 Click the *Edit Result Set Item* icon to open the Edit Result Set Item dialog box:



7 In the *Column* text box, type the name of the database column.

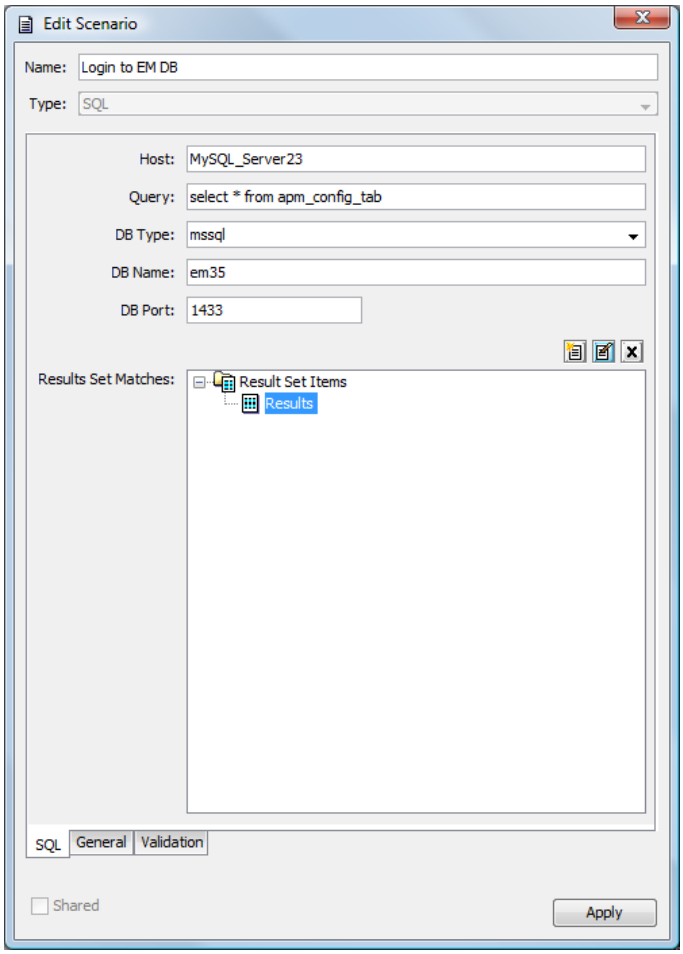
Column is the name of the database column name.

8 In the *Value* text box, type one of the following:

- Type `value` for a result set that matches this value.
- Type `null` for a result set that returns a null value.

9 Click the *Create* button.

The new result set displays in the Edit Scenario dialog box:



10 Click the *Apply* button to update the scenario and close the Edit Scenario dialog box.

5.5.11 Trace Route Scenario Attributes

Table 5-10 describes the trace route scenario attributes.

Table 5-10 *Trace Route Scenario Attributes*

Attribute	Specify
Host	Target host machine name.
Maximum Hops	Maximum number of hops allowed. If the test cannot reach the host in the number of hops specified, the test fails. A suggested value is 30. In a packet-switching network, a data packet hops from one router to another.

5.5.12 Third-Party Scenario Attributes

Developers can write tests that the Experience Manager Monitor's test engine executes, leveraging all the normal consistency of the other test protocols.

A sample third-party test is located in the `tests/thirdpartyexamples` file in the Experience Manager Monitor installation directory.

Creating a Class File

To create a class file:

1. Create a class file that extends the Experience Manager Monitor's methods.
2. Continue with "Creating a Java File to Implement the Test" on page 65.

Creating a Java File to Implement the Test

To create a Java file to implement the test:

1. Create a Java file with a class that imports:

   ```
   com.mcode.agent.test.thirdparty.*;
   ```

2. Verify that the class extends:

   ```
   com.mcode.agent.test.thirdparty.ThirdPartyTest
   ```

3. Verify that the following abstract protected method is included:

   ```
   protected void runTest() throws java.lang.Exception
   {
     //perform your test
   }
   ```

   The parent class gathers the response time.

4. To access the parameter data, add one of the following methods:

   - ```
     /**
         * This method returns the name portion of the parameters
         *
         * @return the names
         */
        public String[] getParameterNames()
     ```
     ```

- ```
  /**
       * This method returns the value portion of the parameters
       *
       * @return the values
       */
      public String[] getParameterValues()
      {
        return values;
      }
  ```

- ```
  /**
       * This method returns a specific value of the parameter based on name
       *
       * @return the value based on a name or NULL if not found
       */
      public String getSpecificParameterValue(String name)
  ```

5 To include additional data in the alarm properties, call the following method:

```
public void addDetail(Detail detail);
//To create a new Detail object
public class Detail
{
 private String name;
 private String value;
 public Detail(String name, String value)
 {
  this.name=name;
  this.value=java.net.URLEncoder.encode(value);
 }
}
```

In the alarm properties, each value pair is represented as a name value pair.

To access the log object, add one of these methods:

- ```
  /**
       * This method logs a warning
       */
      public void logWarning(String msg)
  ```

- ```
  /**
       * This method logs a debug
       */
      public void logDebug(String msg)
  ```

- ```
  /**
       * This method logs a Info
       */
      public void logInfo(String msg)
  ```

- ```
  /**
       * This method logs a error
       */
      public void logError(String msg)
  ```

6 Compile the code using the `thirdpartytest.jar` located in the `/`*`OperationsCenter_ExperienceManager_install_path`*`/tests/thirdpartyexamples` directory.

A `.class` file is created for the Java file.

7 Create a `.jar` file.

8 Add this new `.jar` file to the */OperationsCenter_ExperienceManager_install_path*`/html/bem/monitor` directory.

   This alerts the Experience Manager Monitor to add the file into its classpath.

   It is not necessary to add the `thirdpartytest.jar` because it is already included in the `manager.jar` file.

9 Add this new `.jar` file as an application resource to */OperationsCenter_ExperienceManager_install_path*`/html/bem/monitor.jnlp`.

10 Restart the Experience Manager Monitor to download the new binary file.

Creating a Third-Party Test Scenario

To create a third-party test scenario:

1 In the Create Scenario dialog box, select the *Type* drop-down list, then click *Third Party*.

2 On the *Third Party* page, type the Class Name.

   Provide a fully qualified class in order to call the test. It is not necessary to add the `.class` extension to the class name.

3 Click the *New Parameter* icon to add name/value pairs as required for the third-party test scenario.

4 Click the *Create* button to create the scenario. The new scenario displays under the Shared Scenarios element in the Explorer pane.

5.5.13 Using Date Macros within Scenarios

Define date macros within scenarios to calculate dates based on the current date:

- The CURRENT_ DATE macro defines values in text fields for all scenario types
- The DAY_OF_WEEK function can calculate a static date, such as the first day of the week

Table 5-11 illustrates the various CURRENT_ DATE macro syntax patterns and resulting output.

Table 5-11 *CURRENT_ DATE Macro Syntax Patterns*

Date/Time Pattern	Result
{CURRENT_DATE}	20120629 (default yyyyMMdd)
{CURRENT_DATE(1)[yyyy.MM.dd]}	2012.06.30
{CURRENT_DATE(-1)[EEE, MMM d, yy]}	Tues, Jun 28, 12
{CURRENT_DATE[MM/dd/yy]}	06/28/12

The following syntax is valid for defining a date macro within the field values. `skew` represents the number of days plus or minus the current date:

```
{CURRENT_DATE}
{CURRENT_DATE(skew)}
{CURRENT_DATE(skew)[format]}
{CURRENT_DATE[format]}
```

Integrate date macros with other data to define the value. For example, the file location of an HTTP test might vary every day with locations based on dates:

```
<http file="/articles/{CURRENT_DATE(-1)[MMddyyyy]}/article1.htm">
```

This translates to `file="/articles/06282004/article1.htm"` when the test is executed on June 28, 2004 and to `file="/articles/06292004/article1.htm"` on the following day.

The DAY_OF_WEEK function can calculate a date based on the first day of the current week, which is assumed to be Sunday. The following syntax is valid for defining a date macro within the field values. `Skew` represents the number of days plus the first day of the week:

```
{DAY_OF_WEEK}
{ DAY_OF_WEEK(skew)}
{ DAY_OF_WEEK(skew)[format]}
{ DAY_OF_WEEK[format]}
```

Table 5-12 shows examples of the DAY_OF_WEEK macro syntax patterns and resulting output.

Table 5-12 *{DAY_OF_WEEK} Macro Syntax Patterns*

Date/Time Pattern	Result
{ DAY_OF_WEEK}	20121127 (yyyyMMdd — first Sunday of current week)
{ DAY_OF_WEEK(1)[yyyy.MM.dd]}	2012.11.28 (Monday)
{ DAY_OF_WEEK[MM/dd/yy]}	11/27/12

It is also possible use a date macro to set the current date. Table 5-13 shows some examples.

Table 5-13 *Date/Time Syntax Patterns*

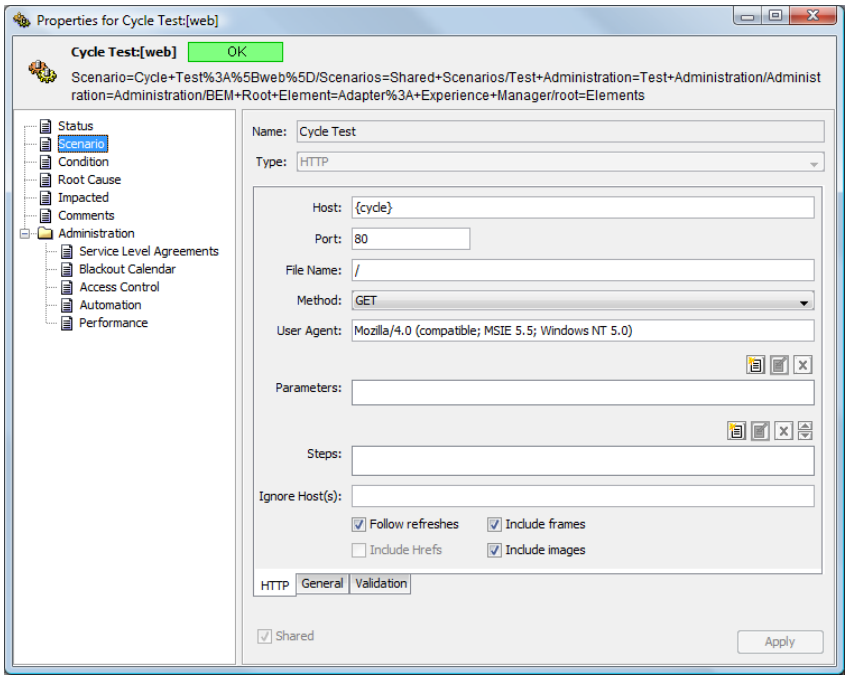
Date/Time	Pattern Result
yyyy.MM.dd G 'at' HH:mm:ss z	2012.07.04 AD at 12:08:56 PDT
EEE, MMM d, “yy	Mon, Jul 4, '12
h:mm a	12:08 PM
hh 'o''clock' a, zzzz	12 o'clock PM, Pacific Daylight Time
K:mm a, z"	0:08 PM, PDT
yyyyy.MMMMM.dd GGG hh:mm aaa	02005.July.04 AD 12:08 PM
EEE, d MMM yyyy HH:mm:ss Z	Mon, 4 Jul 2012 12:08:56 -0700
yyMMddHHmmssZ	120704120856-0700

Another option is to create variables and use the date macro as the value. See the Section 6.11, “Creating Test Variables,” on page 89 section for details.

5.6 Editing Scenarios

To edit test scenarios:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration* > *Shared Scenarios*.
2. Right-click a scenario, then click *Properties* to open the Status property page.
3. In the left pane, click *Scenario* to open the Scenario property page:



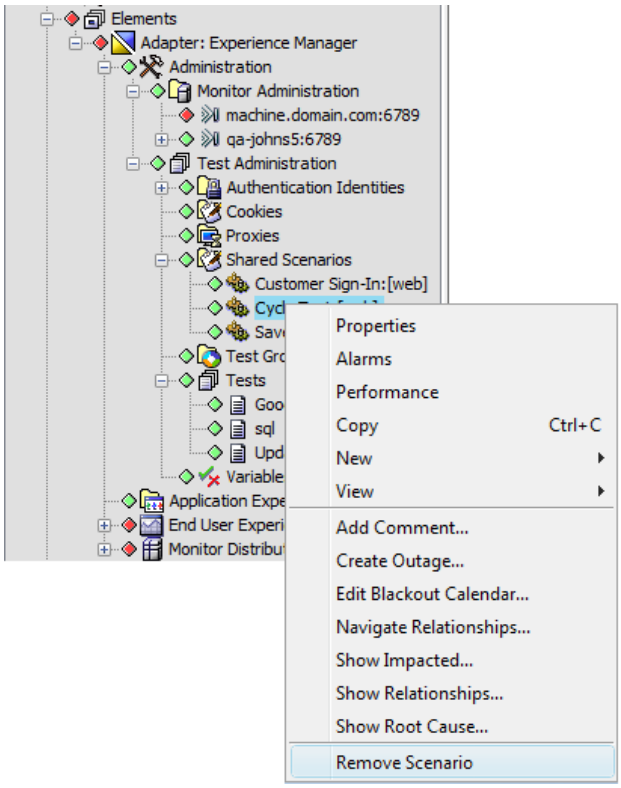
4. Edit the values, then click the *Apply* button to save the changes.

5.7 Deleting Scenarios

To delete test scenarios:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration* > *Shared Scenarios*.
2. Right-click a scenario, then click *Remove Scenario* to open a confirmation dialog box.
3. Click the *Yes* button in the confirmation dialog box.

   If the scenario is shared by two or more tests, a message displays the test names that use the scenario. It is necessary to remove the scenario from these tests, then use the Remove Scenario command.



6 Creating and Deploying Synthetic Tests

Schedule synthetic test transactions to run at specific time intervals and emulate various numbers of users. It is possible to test any combination of time periods and user volumes. Moreover, placing Experience Manager Monitors at different points in the network infrastructure can help determine how the infrastructure affects response time. Use all this data to establish baselines for acceptable performance levels and alert thresholds. Continuous monitoring of synthetic tests can also help identify possible slowdowns.

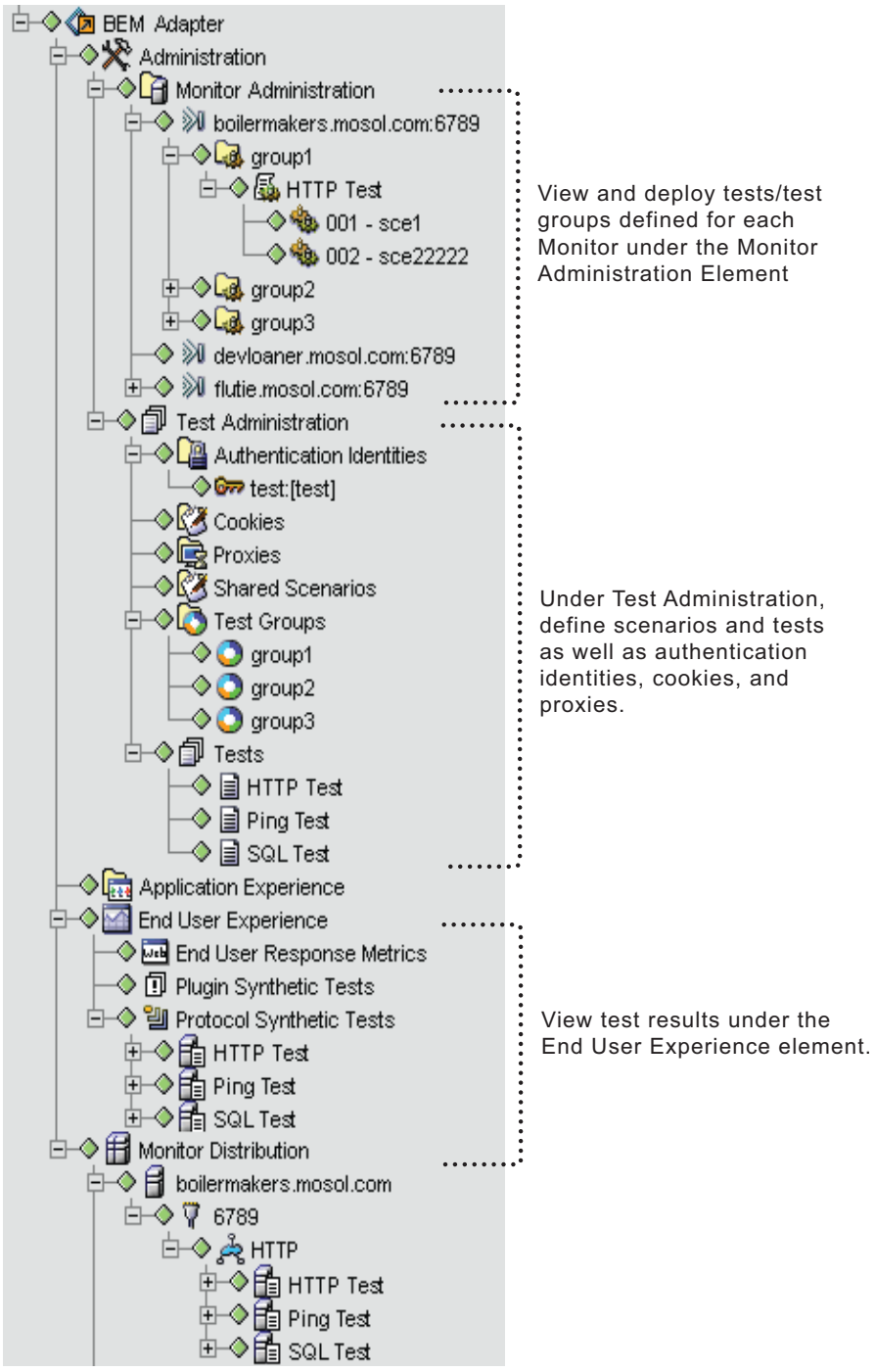
6.1 Overview

Synthetic tests defined for Experience Manager Monitors perform actions that are defined using test scenarios. These scripts can include tests that access and perform actions on multiple Web sites, log into e-mail, access FTP sites or perform more network-centric tasks such as ICMP, traceroute or nslookup transactions.

Use the response times resulting from the synthetic tests as a baseline to measure real-time responses that occur when end users interact with the system.

In the Explorer pane, defined tests display under the Test Administration element. Assign these tests to Experience Manager Monitors listed under the Monitor Administration element. Test results display under the End User Experience element, as shown in Figure 6-1:

Figure 6-1 *Test Results Displaying Under the End User Experience Element*



Tests defined for a Experience Manager Monitor reside on a specific monitor host. Only the specified Experience Manager Monitor can access the tests. Operations Center stores the test configuration. Only the Experience Manager Monitor needs to run to deploy, start, or stop tests.

6.2 Defining Tests

After creating a shared scenario, add it to a test definition.

It is possible to create a scenario on the fly for a test. However, these are saved as private scenarios that apply only to the test definition in which they were created.

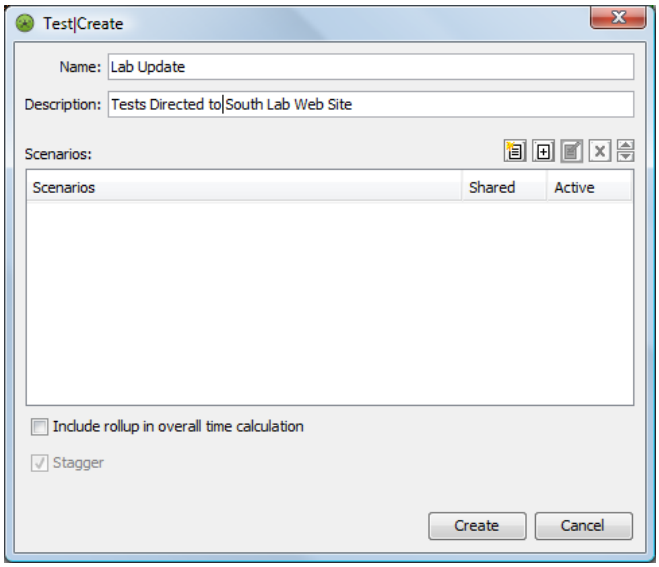
After creating a test, determine if it requires user authentication or accessing a proxy server. See Section 6.4, "Defining Proxy Settings," on page 77 and Section 6.5, "Defining User Authentication Settings," on page 78 for details. It might also be necessary to import a custom SSL certificate to run SSL tests that require validation of a security certificate. See Section 6.6, "Importing a Custom SSL Server for a Certificate Authority (CA)," on page 78.

- Section 6.2.1, "Creating a Test," on page 74
- Section 6.2.2, "Creating a Private Scenario," on page 75

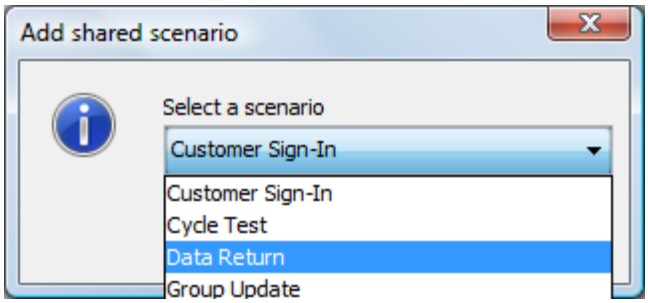
6.2.1 Creating a Test

To create a test:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.
2. Right-click *Tests*, then click *Test* > *Create* to open the Test|Create dialog box.
3. In the *Name* text box, type a test name.
4. In the *Description* text box, type text that distinguishes this test from other tests.



5 Click the *Add Shared Scenarios* icon and select a scenario.



6 Select the *Active* check box to activate the scenario for the test.

7 Select additional scenarios, if necessary, and select the *Active* check box next to each scenario.

8 Because scenarios execute in the order in which they display in the Test definition, you can use the *Move Up* and *Move Down* icons to reorder scenarios.

9 Select the *Include rollup in overall time calculation* check box to calculate overall time as an aggregate element, including all response times for the test.

10 Select the *Stagger* check box to have the test complete runs for all simulated users before waiting the specified period.

11 Click the *Create* button to create the test. The new test displays beneath the Tests element in the Explorer pane.

IMPORTANT: If the Stagger check box is deselected, an overload and ramping of simulated users can occur.

6.2.2 Creating a Private Scenario

Private scenarios are only available and visible in the test definitions in which they were created.

Execute multiple tests together in a test group. See Section 6.3, "Creating Test Groups," on page 76 for more information about test groups.

Some tests require Proxy and User Authentication settings.

A separate action is required to deploy tests and test groups for each Experience Manager Monitor.

To create a private scenario:

1 In the Test | Create dialog box, click the *New Private Scenario* icon to open the Create Scenario dialog box.

2 Create the scenario.

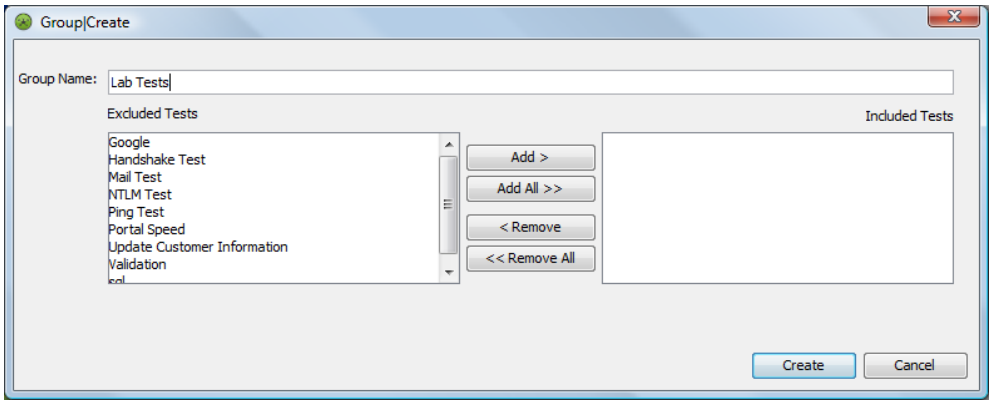
For more information, see Chapter 5, "Creating Test Scenarios," on page 41.

6.3 Creating Test Groups

Test groups consist of multiple tests included in a single definition.

To create a test group:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.
2. Right-click the *Test Groups* element, then click *Group* > *Create* to open the Group|Create dialog box:



3. Type a group name in the *Group Name* text box.
4. To add one or all tests to the group perform one of the following steps:
   - To add a single test to the group, select a test name in the *Excluded Tests* list, then click the *Add* button. The test displays in the *Included Tests* list.
   - To add all tests to the group, click the *Add All* button. The tests display in the *Included Tests* list.
5. To remove one or more tests from the group, perform one of the following steps:
   - Select a test name in the *Included Tests* list, then click the *Remove* button. The test moves from the *Included Tests* list to the *Excluded Tests* list.
   - To remove all selected tests from the group, click the *Remove All* button. The tests move from the *Included Tests* list to the *Excluded Tests* list.
6. Click the *Create* button to save the test group. The new group displays under the *Groups* element.
7. Click the *Close* button to close the Create Group dialog box.

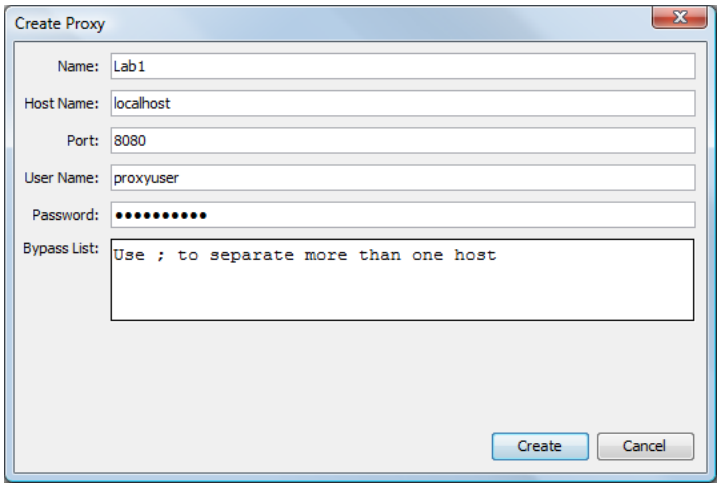
A separate step is required to deploy tests and test groups for a Experience Manager Monitor. See Section 6.9, “Deploying and Managing Tests,” on page 85 for more information.

6.4 Defining Proxy Settings

If a test requires accessing Web sites through proxy servers, create proxy definitions in Experience Manager. During deployment, the tests load all associated proxy definitions.

To create a proxy definition:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.
2. Right-click *Proxies*, then click *Create Proxy* to open the Create Proxy dialog box:



3. Type values for the proxy settings:

   Name: Name for the proxy definition.

   Host Name: The proxy server name.

   Port: The proxy server port number. The default is 8080.

   User Name: Type the user name for the proxy account. If none is required, leave this text box blank.

   Password: Type the password for the proxy account. If no password is required, leave this text box blank.

   Bypass List: Type hostnames that are bypassed during Web scenario tests. Use a semicolon to separate multiple hosts.

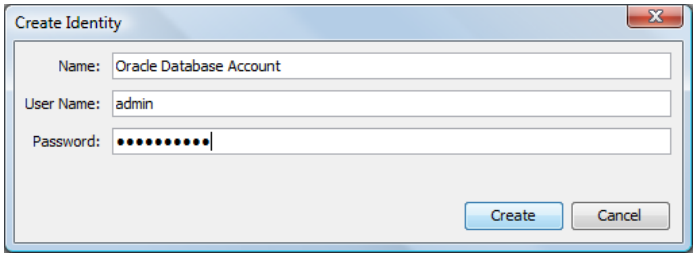
4. Click the *Create* button to save the proxy definition. The new proxy definition displays under the *Proxies* element.

6.5 Defining User Authentication Settings

Define user authentication identities for tests that require them, such as LDAP and tests that require Integrated Windows Authentication (IWA)/Windows NT LAN Manager (NTLM) authentication. During deployment, tests load all associates user authentication identities.

To define user authentication settings:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.
2. Right-click *Authentication Identities*, then click *Create Identity* to open the Create Identity dialog box:



3. In the *Name* text box, type a name for the identity definition.
4. In the *User Name* text box, type the expected user name.

   Enter the *`domain\user name`* for an HTTP(S) scenario to perform authentication with a Web server that requires IWA/NTLM authentication. For example, `NET1\ajones`. When the test executes, Experience Manager detects the IWA/NTLM requirement and performs the authentication.
5. In the *Password* text box, type the expected password.
6. Click the *Create* button to save the identity definition.

   The new identity definition displays under the *Identities* element.

6.6 Importing a Custom SSL Server for a Certificate Authority (CA)

To run SSL tests that require validation of a security certificate, it is necessary to import custom SSL certificates if a Web site or Web application has a custom Secure SSL Server for a Certificate Authority (CA). You must obtain a copy of the required public key file from the Web site and import it into the Experience Manager keystore library.

- Section 6.6.1, "Obtaining a Copy of the Required Public Key File," on page 78
- Section 6.6.2, "Updating the Experience Manager Monitor," on page 79

6.6.1 Obtaining a Copy of the Required Public Key File

To get the public key file:

1. Use Internet Explorer* to navigate to the HTTPS Web site.
2. From the menu, click *File* > *Properties*.
3. Click the *Certificate* button.

4 Click the *Certificate Path* tab.

5 Select the root CA certificate.

6 Click the *Details* tab.

7 Click the *Copy to file* button.

8 Click *Next*.

9 Select *Base-64 encoded X.509* as the format.

10 Click *Next*.

11 Type a file name, click *Next*, then click *Finish*.

6.6.2 Updating the Experience Manager Monitor

The certificate file might be the in X.509 v1, v2, or v3 format. After you have the certificate file, update the Experience Manager Monitor.

To update the Experience Manager Monitor:

1 Stop the Experience Manager monitor process.

2 Open a command line window and navigate to the `/OperationsCenter_ExperienceManager_install_path/jre/bin` directory.

3 Run the following command:

```
./keytool –import –keystore ../lib/security/cacerts –file
<pubkey.filename.withlocation> -alias customerName
```

4 When prompted, type the password for this keystore: `changeit`

5 To verify success, run the following command and confirm that the custom certificate is listed:

```
./keytool –list –v –keystore ./lib/security/cacerts
```

You can also use this command to view a list of all current certificates.

6 Edit the monitor.properties file by setting `tests.https.trustManagerMode=ENABLED`.

7 Restart the monitor and test the Web site access.

6.7 Defining Cookie Information

6.7.1 Understanding Cookies in Experience Manager

Experience Manager has an internal Cookie Manager that can persist cookies received from the server and pass these cookies back to the server upon request.

Define cookie locations in cookie definitions (under the Test Administration element). Associate cookies with specific tests during deployment.

During test deployment, HTTP and HTTPS scenarios attempt to read the specified cookie file from a machine’s hard drive or a specified URL and populate the Cookie Manager.

If the loaded file contains cookies, the Cookie Manager no longer accepts or stores cookies sent from Web servers. If this file cannot be located or contains no data, it uses the default internal Cookie Manager. The cookie file can contain multiple `Set-Cookie:` lines.

The syntax of the `Set-Cookie` HTTP response header is:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure
```

Table 6-1 describes the cookie formats in this HTTP response header.

Table 6-1 *Cookie Format*

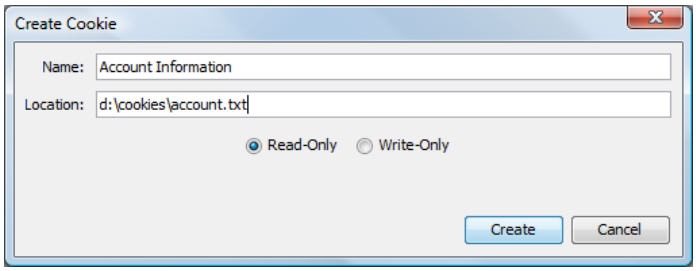
Value	Description
NAME=VALUE	Name of the cookie, using any characters except for a semicolon, comma, or white space. If these exceptional characters must be placed in the name or value, use an encoding method such as URL style %XX. This is the only required attribute for the `Set-Cookie` header.
expires=DATE	A date string that defines the valid lifetime of the cookie. After the expiration date, the cookie is no longer stored or given out. This is an optional attribute. If it is not specified, the cookie expires when the user's session ends. The date string is formatted as Wdy, DD-Mon-YYYY HH:MM:SS GMT This is based on RFC 822 http://ds.internic.net/rfc/rfc822.txt, RFC 850 http://ds.internic.net/rfc/rfc850.txt, RFC 1036 http://www.w3.org/hypertext/WWW/Protocols/rfc1036/rfc1036.html, and RFC 1123 http://ds1.internic.net/rfc/rfc1123.txt, with the variations that the only legal time zone is GMT and the separators between the elements of the date must be dashes.
domain=DOMAIN_NAME	When searching a list for valid cookies, a comparison occurs between the domain attributes of the cookie and the Internet domain name of the host from which the URL is fetched. If there is a tail match, the cookie goes through path matching to determine if it is sent. Tail matching matches the domain attribute against the tail of the fully qualified domain name of the host. For example, a domain attribute of acme.com matches host names anvil.acme.com as well as shipping.crate.acme.com. Only hosts within the specified domain can set a cookie for a domain. Domains must contain at least two or three periods to prevent domains of the form .com, .edu, and va.us. The following seven special top-level domains require only two periods: COM, "EDU, NET, GOV, MIL, and INT. All other domains require at least three periods. The default value of domain is the hostname of the server that generated the cookie response.
path=PATH	The path attribute specifies the subset of URLs in a domain for which the cookie is valid. For a cookie that passes domain matching, a comparison occurs between the pathname component of the URL and the path attribute. If there is a match, the cookie is valid and sent along with the URL request. The path /foo matches /foobar and /foo/bar.html. The path "/" is the most general path. An unspecified path is assumed to be the same path as the document described by the header that contains the cookie.

Value	Description
secure	Marking a cookie secure transmits it only if the communications channel with the host is secure. This means that secure cookies only travel to HTTPS (HTTP over SSL) servers. If the secure attribute is not specified, a cookie is considered safe to send over unsecured channels.

6.7.2 Creating a Cookie Definition

To create a cookie definition:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.
2. Right-click *Cookies*, then click *Create Cookie* to open the Create Cookie dialog box:



3. In the *Name* text box, type a name for the identity definition.
4. In the *Location* text box, type the cookie path and file name or URL.
5. To specify if the cookie is read-only or write-only, perform one of the following steps:
   - Select the *Read-Only* radio button to populate the Cookie Manager with the contents of the specified file prior to running any scenarios.
   - Select the *Write-Only* radio button to store all cookies in the Cookie Manager in the specified file/location after completing all test scenarios.
6. Click the *Create* button to save the identity definition. The new cookie definition displays under the Cookies element.

6.8 Managing Tests

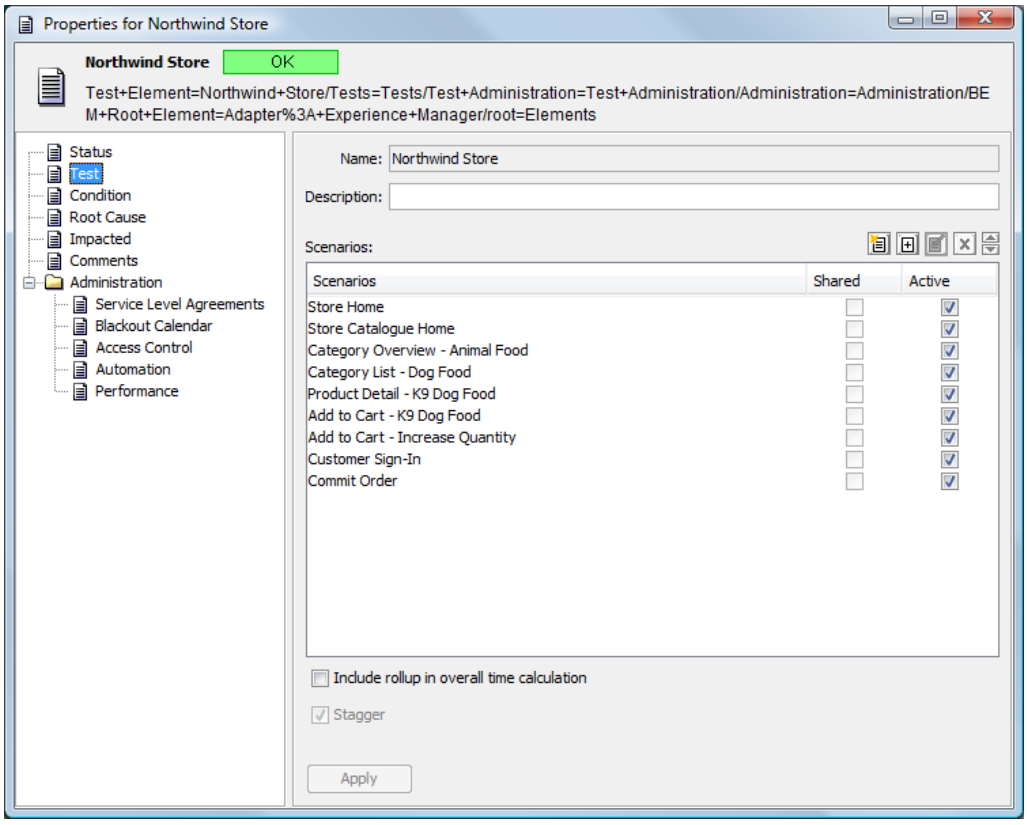
6.8.1 Editing Tests

To edit a test scenario:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration* > *Tests*.
2. Right-click a test, then click *Properties* to open the Status property page.
3. In the left pane, click *Test* to open the Test property page.
4. Use the Scenario icons to add a private scenario or a shared scenario, edit a scenario, or remove a scenario.

Scenario Icon	Function
	Add Shared Scenario
	Add Private Scenario
	Edit Scenario
	Remove Scenario
	Move Scenario Up or Down

5. Click the *Apply* button to save the changes.



6.8.2 Copying Tests

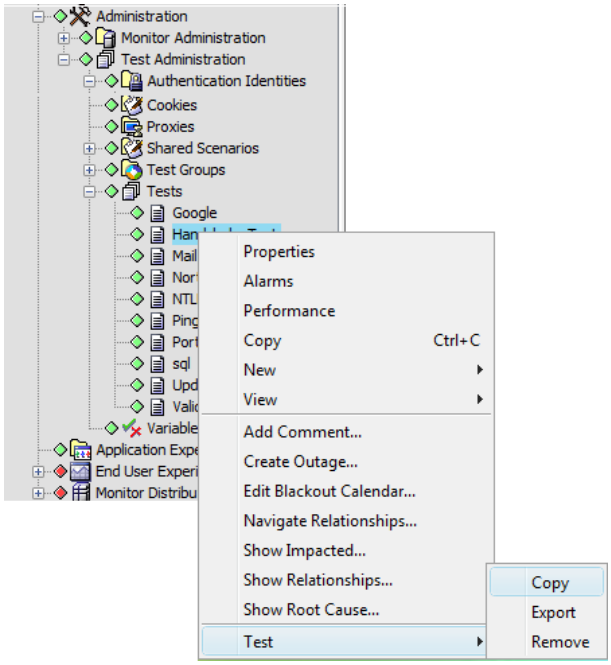
Do the following to copy tests:

- "Creating a Test by Copying an Existing Test" on page 83
- "Editing the Scenarios in a Copied Test" on page 84

Creating a Test by Copying an Existing Test

To create a test by copying an existing test:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration* > *Tests*.
2. Right-click a test, then click *Test* > *Copy*.

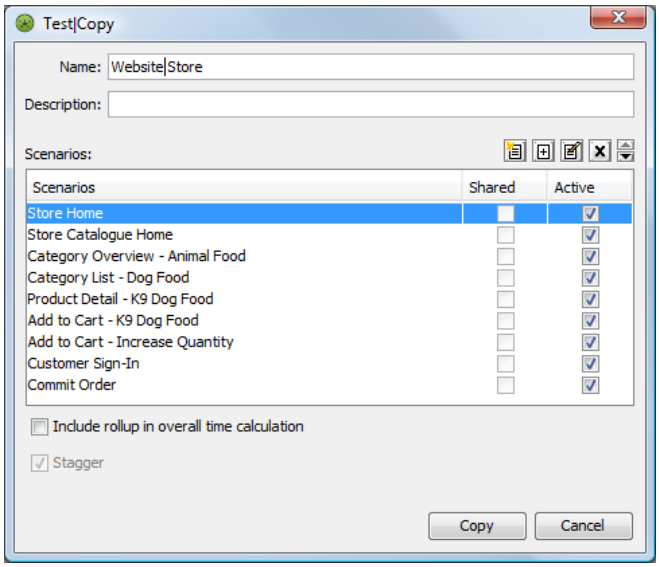


3. In the Test | Copy dialog box, type a new test name.

   It is not possible to edit the scenarios in this dialog box.

4 Click the *Copy* button.

The dialog box closes and the new test name displays in the Explorer pane:



Editing the Scenarios in a Copied Test

To edit the scenarios in a copied test:

1 Right-click the test element in the Explorer pane, then click *Properties* to open the Status property page.

2 In the left pane, click the *Test* tab to open the Test property page.

3 Use the Scenario icons to add a private scenario or a shared scenario, edit a scenario or remove a scenario.

4 Click the *Apply* button to save the changes.

6.8.3 Debugging Tests

Use the Script Debugger to develop and test scripts. See the *Operations Center 5.5 Scripting Guide* for details on its features.

To load the Debugger:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration* > *Tests*.

2 Right-click a test, then click *Test* > *Debugger* to open the Debugger in a new window.

6.8.4 Adding a Test Timeout Parameter

The `test.timeout` parameter can be used to cancel a test that runs longer than a specified amount of time. When a test exceeds the timeout, the test is cancelled and the current scenario is set to CRITICAL. The current scenario is allowed to finish running, but subsequent scenarios are cancelled.

To set the `test.timeout` parameter:

1. In a text editor, open the */OperationsCenter_ExperienceManager_install_path*`/config/monitor.properties` file.
2. Set or edit the `test.timeout` parameter to the number of milliseconds to time out when exceeded by a test. The default is 30 minutes.
3. Save your changes.

6.8.5 Deleting Tests

To delete a test:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration* > *Tests*.
2. Right-click a test, then click *Test* > *Remove* to open a confirmation dialog box.
3. Click the *Yes* button to remove the test from the Explorer pane.

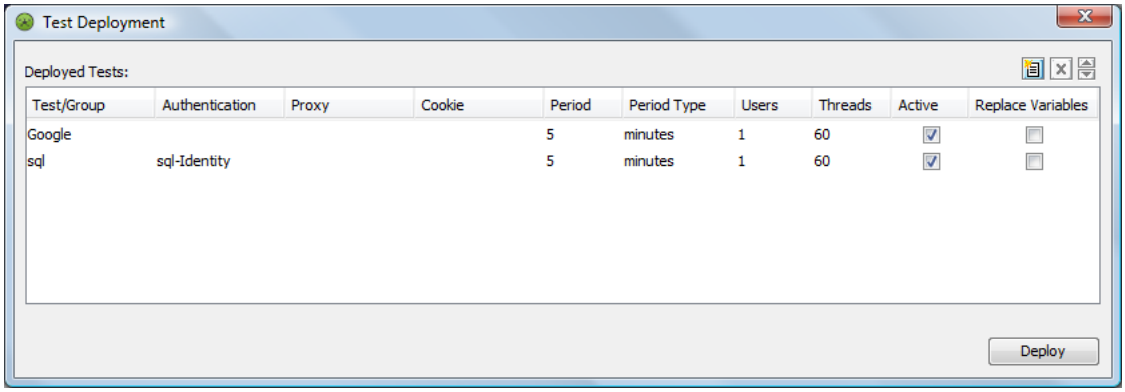
6.9 Deploying and Managing Tests

After defining all scenarios, tests, test groups as well as proxy and user identities, deploy them for individual Experience Manager Monitors.

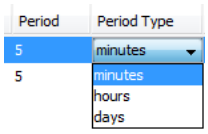
6.9.1 Deploying a Test

To deploy a test:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Monitor Administration*.
2. Right-click a Experience Manager Monitor, then click *Deploy Tests* to open the Tests Deployment dialog box.



3. Click the *New Item* icon to display a new item in the *Deployed Test* list.
4. Click a value in the *Test/Group* column and select a test or test group from the drop-down list.
5. Click the *Period Type* drop-down list and select the frequency of the test.



6. In the *Period* text box, type the time interval for the test.

   If no value is specified, the internal default is 5 minutes.
7. If necessary, specify user authentication accounts, proxies, cookies or threads for the test.

   See Section 6.9.2, “Associating Cookies, Authentication, and Proxy Accounts,” on page 86 for details.
8. In the *Users* text box, type the number of users to simulate in the test.
9. In the *Active* column, select the check box next to the test.

   This activates the test to run at the specified time interval.
10. Click the *Deploy* button.

    All deployed tests display under the associated monitor in the Explorer pane.

6.9.2 Associating Cookies, Authentication, and Proxy Accounts

To associate with a test or group:

1. To associate a cookie file, click a value in the *Cookie* column and select a cookie definition.

   For instructions regarding linking to a cookie file by pathname or URL, see Section 6.7, “Defining Cookie Information,” on page 79.

2 To associate a user authentication account, select a value in the *Authentication* column and select an Identity definition.

   For instructions regarding creating a user authentication account, see Section 6.5, "Defining User Authentication Settings," on page 78.

3 To associate a proxy account, select a value in the *Proxy* column and select a Proxy definition.

   For instructions on creating a proxy account, see Section 6.4, "Defining Proxy Settings," on page 77.

6.9.3 Specifying Threads

To specify the maximum number of spawned threads used for each simulated user in a test:

1 Click a value in the *Threads* column and enter a number in the text box.

   For example, a test with a Users value of 500 and 10 Threads completes all 500 tests, but only 10 (the Threads value) execute at a time. When one test finishes, another appears in the list.

6.9.4 Using Test Variables

Test variables replace test/scenario parameters during test execution. Select one or more check boxes in the *Replace Variable* column next to the tests or groups to make the entire list of variables for replacement available during test execution, If the check box is cleared, the Monitor uses one value and does not substitute the variable. See Section 6.11, "Creating Test Variables," on page 89 for information on creating and using variables.

6.10 Starting and Stopping Synthetic Tests and Scenarios

After deploying synthetic tests with Monitors, start the tests and they continue to run automatically at the specified intervals. Use commands to stop tests or scenarios within a test and restart them.

6.10.1 Starting All Synthetic Tests for a Monitor

To start all synthetic tests for a monitor:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Monitor Administration*.

2 To start all tests, right-click a Monitor, then click *Activate All Tests*. The tests execute at their specified time intervals.

6.10.2 Stopping All Synthetic Tests for a Monitor

To stop all synthetic tests for a monitor:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Monitor Administration*.
2. Right-click a Monitor, then click *Deactivate All Tests*.

6.10.3 Starting or Stopping a Synthetic Test

To start or stop a specific synthetic test:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > Monitor Administration, select a monitor.
2. To start a test, right-click a test and select *Test* > *Activate*. The active test scenarios run at their specified time intervals.
3. To run the test immediately, thereby resetting its time interval, right-click a test and select *Test* > *Run Now*.
4. To stop a test, right-click a test, then click *Test* > *Deactivate*.

6.10.4 Starting a Synthetic Test from the Command Line

To start a specific synthetic test immediately from the command line and reset its time interval:

1. Enter the following command:

```
java -cp [location]/bootstrap.jar com.mcode.agent.bootstrap.Bootstrap
"[location]/monitor.properties" "SYNTH" "[testname]" "RUN"
```

6.10.5 Stopping a Synthetic Test from the Command Line

To stop a synthetic test from the command line:

1. Enter the following command:

```
java -cp [location]/bootstrap.jar com.mcode.agent.bootstrap.Bootstrap
"[location]/monitor.properties" "SYNTH" "[testname]" "STOP"
```

6.10.6 Starting or Stopping a Scenario Within a Test

To start or stop a specific scenario within a test:

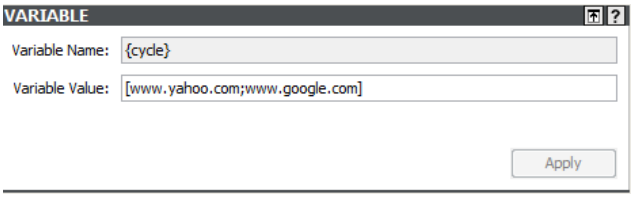
1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Monitor Administration*, then select a monitor and a test.
2. To start a scenario, right-click a scenario, then click *Scenario* > *Activate*. The selected scenario executes at the specified time interval.
3. To stop a scenario, right-click a scenario, then click *Scenario* > *Deactivate*. The selected scenario stops.

6.11 Creating Test Variables

Test variables replace test/scenario parameters during test execution. Use these variables in any open text field for scenarios. Also create array-based variables, if necessary.

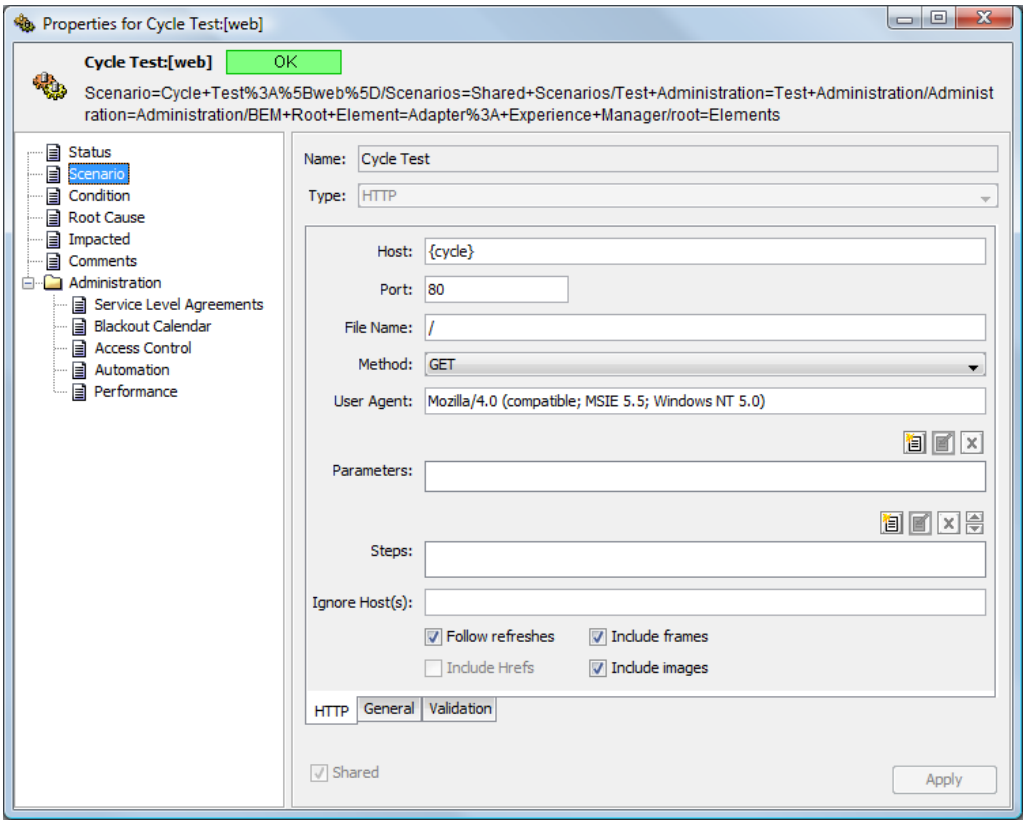
As an example of using variables, assume an array-based variable allows a scenario to cycle every scenario execution. Assume an HTTP test sets the host field to the {cycle} variable. The values are www.yahoo.com;www.google.com.

Figure 6-2 *Variable Element Properties in the Portal View*



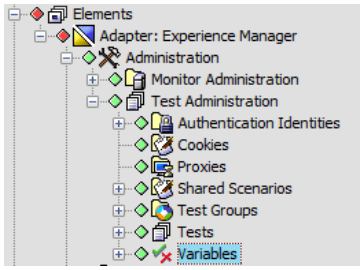
The first scenario execution results in a HTTP test against www.yahoo.com and the second test against www.google.com. It then cycles back to www.yahoo.com and so on. The event in Operations Center is the same, but the host field changes when each array value executes.

Figure 6-3 *Scenario Properties Dialog Box Showing an Array-Based Variable for the Hostname*

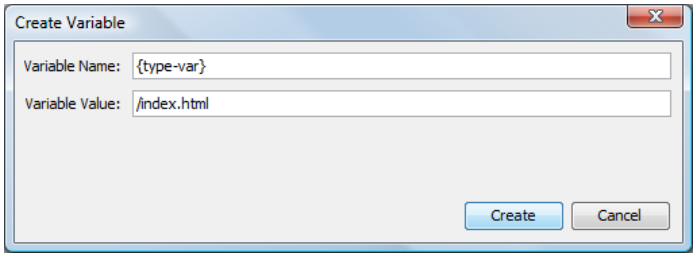


To define a test variable:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > and *Test Administration*.



2 Right-click *Variables*, then click *Create Variable* to open the Create Variable dialog box:



3 In the *Variable Name* text box, type a name for the variable. Enclose the name in brackets: {…}.

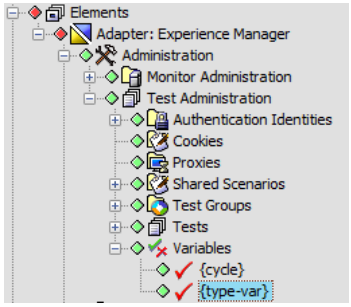
4 In the *Variable Value* text box, type the name of the variable to replace during test execution.

For array-based variables, enclose the values in square brackets: […]. Also, delimit the array values by using a semicolon. Incorrect syntax results in not creating the array and using the text itself as the default.

For example, assume an array-based variable allows a scenario to cycle every scenario execution. Assume an HTTP test has the host field set to the {cycle} variable. The Variable Values are [www.yahoo.com;www.google.com].

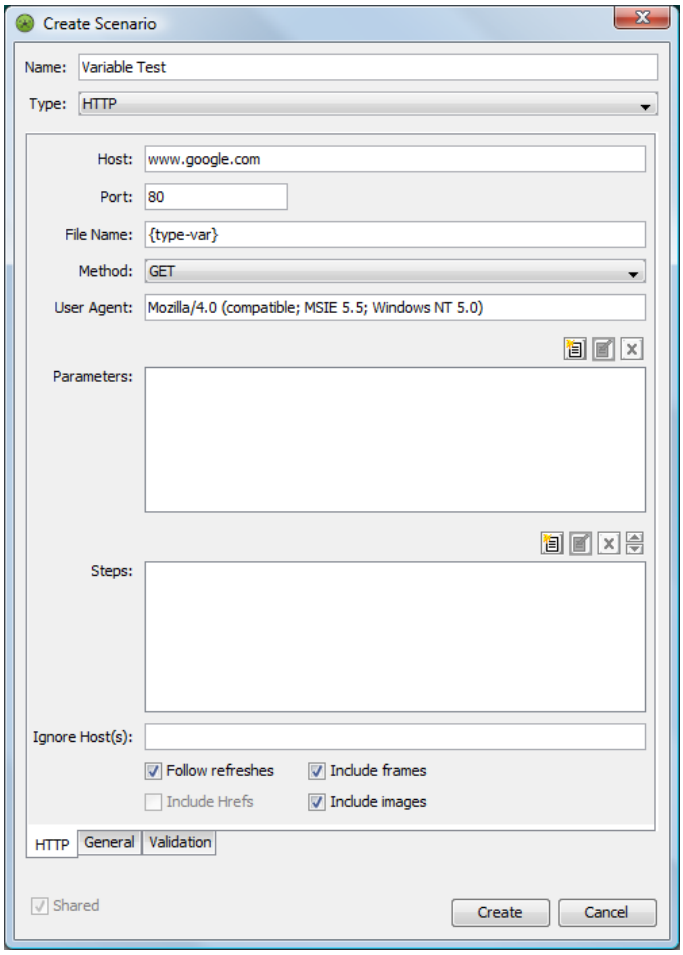
5 Click the *Create* button.

The new variable name displays beneath the *Variables* element in the Explorer pane.



6 Create a scenario using the variable.

Assume a HTTP test uses the `{type-var}` variable. When creating the test scenario, type `{type-var}` in the File Name text box. When the test executes, /index.html is replaced.



7 Define a test using the variable scenario.

See Section 6.2, “Defining Tests,” on page 74 for details.

8 Deploy the test for a monitor.

See Section 6.9, “Deploying and Managing Tests,” on page 85 for details on accessing the Deploy Tests dialog box for a Experience Manager Monitor.

9 In the Deploy Tests dialog box, select the check box in the *Replace Variable* column beside a test or group.

The entire list of variables is made available to the selected text or group.

If the check box is cleared, the Experience Manager Monitor uses the value and does not substitute the variable.

6.12 Exporting and Importing Tests

Export test groups and individual tests and import tests. Export tests and test groups as XML files.

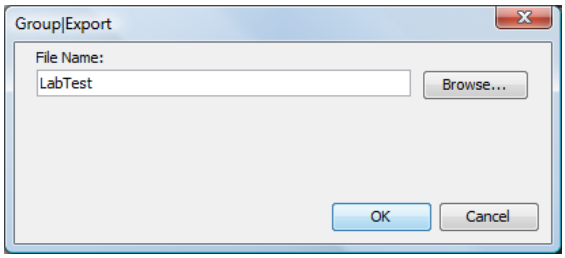
6.12.1 Exporting Test Groups

To export single or multiple test groups, do one of the following:

Exporting a Test Group

To export a specific test group:

1. In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration* > *Test Groups*.
2. Right-click an individual test group element, then click *Group* > *Export* to open the Group Export dialog box.



3. In the *File Name* text box, type an export file name.

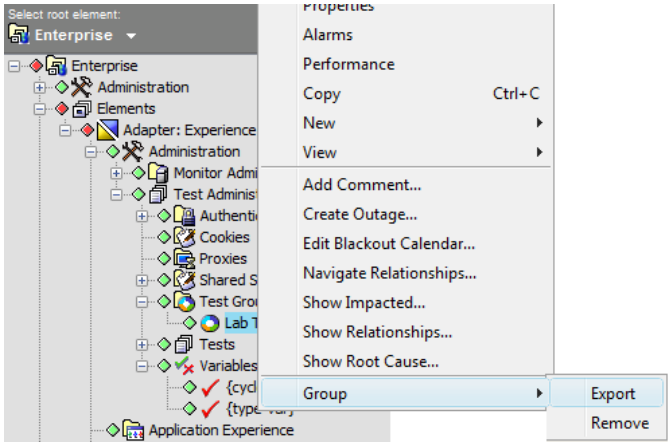
Click the *Browse* button to select an existing XML file.

4 Click the *OK* button to export the test group to the specified file.

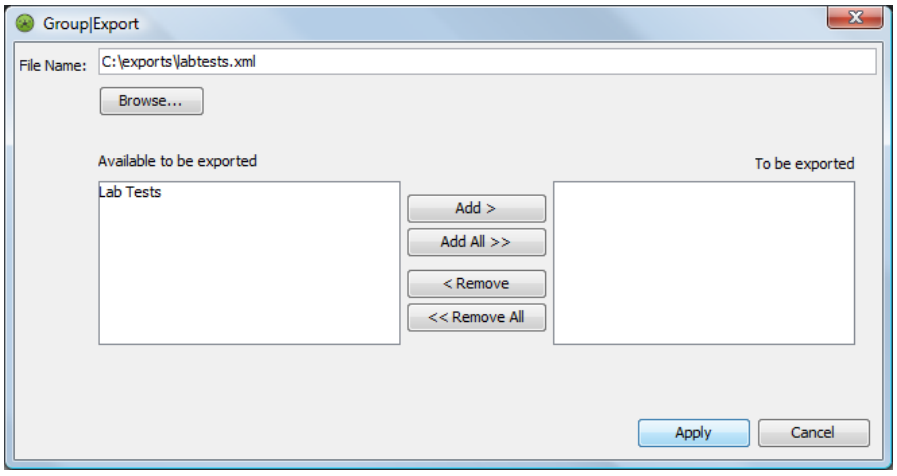
Exporting Multiple Test Groups

To export multiple test groups at a time:

1 In the *Explorer* pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration*.



2 Right-click the *Test Groups* element, then click *Group* > *Export* to open the Group Export dialog box:



3 In the *File Name* text box, type an export file name.

Use the *Browse* button toselect an existing XML export file.

4 In the *Available to be exported* list, select the groups to export, then click the *Add* button.

The selected groups move to the *To be exported* list.

5 Click the *Apply* button to export the groups to the specified file.

6.12.2 Exporting Tests

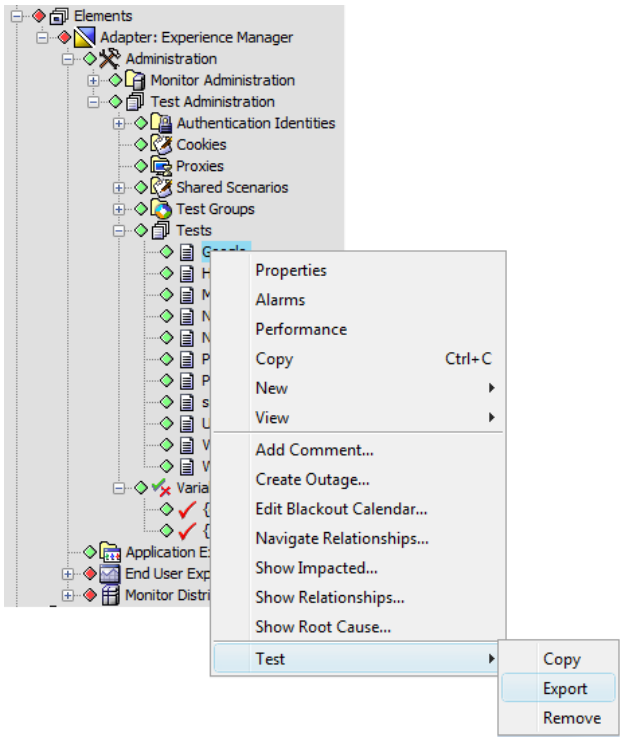
To export single or multiple tests, do one of the following:

- "Exporting a Test" on page 94
- "Exporting Multiple Tests" on page 95

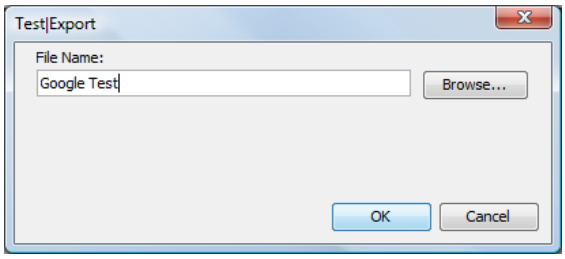
Exporting a Test

To export a single test:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration* > *Tests*.



2 Right-click a test, then click *Test* > *Export* to display the Test Export dialog box:



3 In the *File Name* text box, type an export file name.

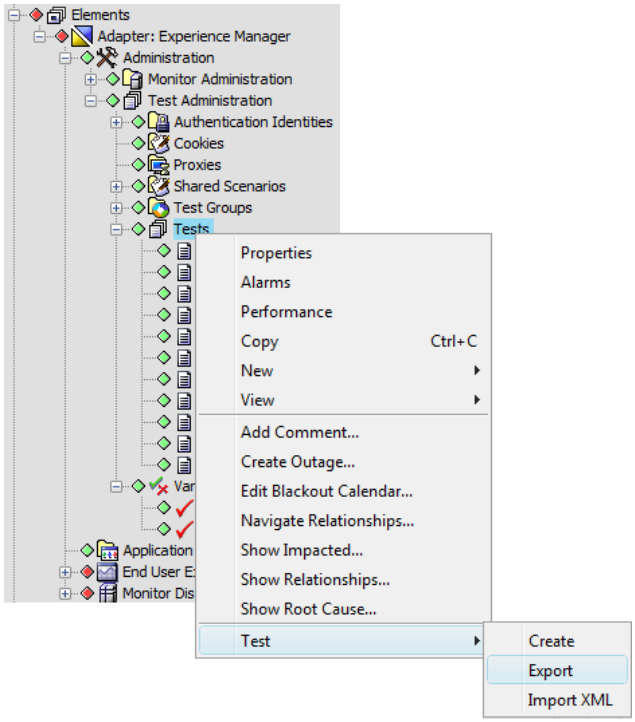
Use the *Browse* button to select an existing XML export file.

4 Click the *OK* button to export the test to the specified file.

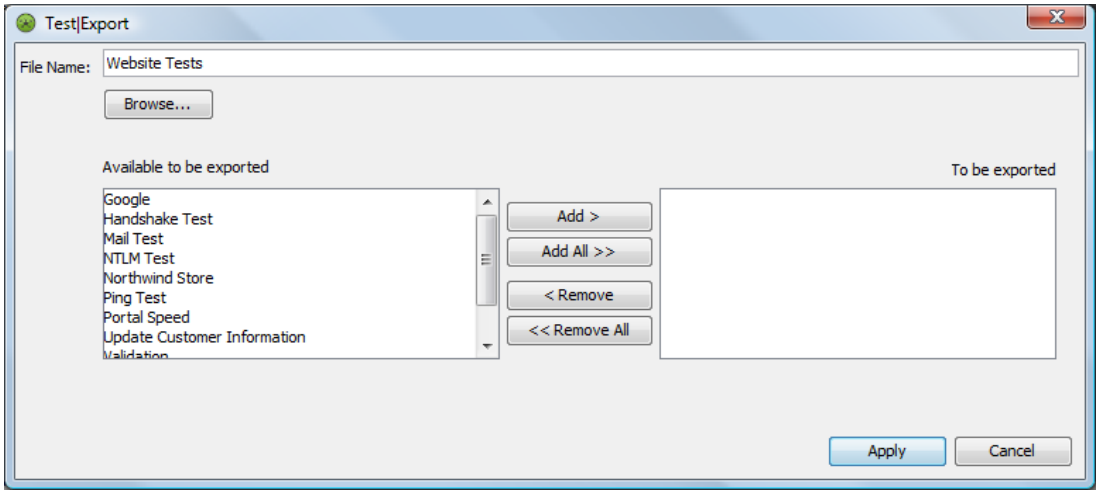
Exporting Multiple Tests

To export multiple tests:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *Administration* > *Test Administration*.



2 Right-click the *Tests* element, then click *Tests* > *Export* to display the Test Export dialog box:



3 In the *File Name* text box, type an export file name.

Use the *Browse* button to select an existing XML export file.

4 In the *Available to be* exported list, select the tests to export, then click the *Add* button.

The selected tests move to the To be exported list.

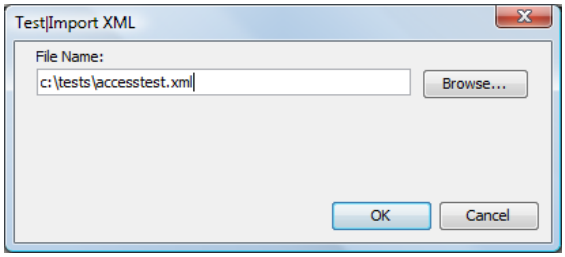
5 Click the *Apply* button to export the selected tests to the specified file.

6.12.3 Importing Tests

To import a test configuration XML file:

1 In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration*.

2 Right-click the Tests element, then click *Test* > *Import XML* to display the Test Import dialog box:



3 Use the *Browse* button to locate the XML file, then click the *OK* button.

The imported tests display under the *Tests* element.

7 Creating Synthetic Tests Using the HTTP Recorder

The HTTP Recorder can capture, verify, and replay user interactions, allowing an organization to view availability and ensure that business processes perform optimally upon deployment and remain reliable.

Use the HTTP Recorder to assist with the creation of complex, Web-based synthetic testing scenarios, such as those described in the previous sections.

HTTP and HTTPS sessions are automatically captured in an XML script file, which can be then be used to generate all or a portion of a synthetic test. The XML file automatically saves information about the steps performed and the data entered.

After importing the HTTP Recorder file into the Experience Manager adapter, it creates a test and automatically generates private scenarios for the captured test scenarios.

7.1 Configuring the HTTP Recorder

There are two optional configurations that can be made to the HTTP Recorder to streamline tests and improve efficiency:

- Restrict specific HTTP/HTTPS transactions for a host from a recording when you do not wish to capture information about those transactions. For example, in the case of a page running Google Adwards, you most likely would not want to capture data regarding transactions for the ads.
- If the recorder experiencing difficulty connecting to an SSL site, specify host(s) requiring SSL for secure communications so the the recorder connects using SSL 3.0.

To configure the HTTP Recorder:

1 Open the */OperationsCenter_ExperienceManager_install_path*/config/recorder.properties file in a text editor.

2 Set the following properties as required:

Configuration Property	Specify...
Recorder.IgnoreHost	A comma-delimited list of host names to exclude. Partial host names can be specified using wild cards. For example, `test*` excludes all host names that begin with test.
Recorder.SSLHosts	A comma-delimited list of host names that do not support TLS 1.0. When the recorder establishes a secure connection to a host, it uses TLS 1.0 and switches to SSL 3.0 if TLS doesn't work. If a host is defined here, the recorder will not attempt to connect initially with TLS 1.0.

3 Save the file.

7.2 Selecting Proxy Server Settings for the Recorder

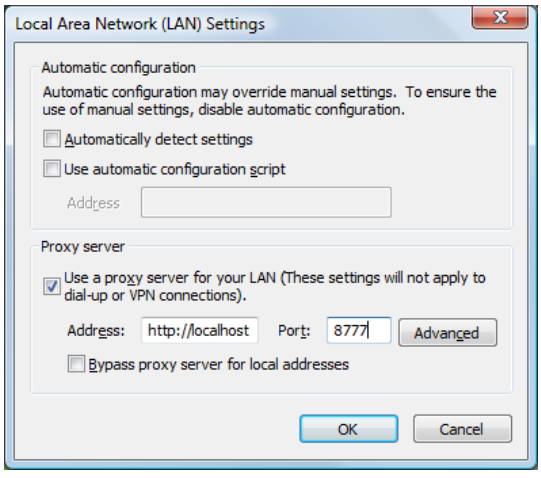
In the Web browser, set the proxy server settings for the HTTP Recorder.

To set up the HTTP Recorder using Internet Explorer:

1 From the Internet Explorer *Tools* menu, click *Internet Options* to open the Internet Options dialog box.

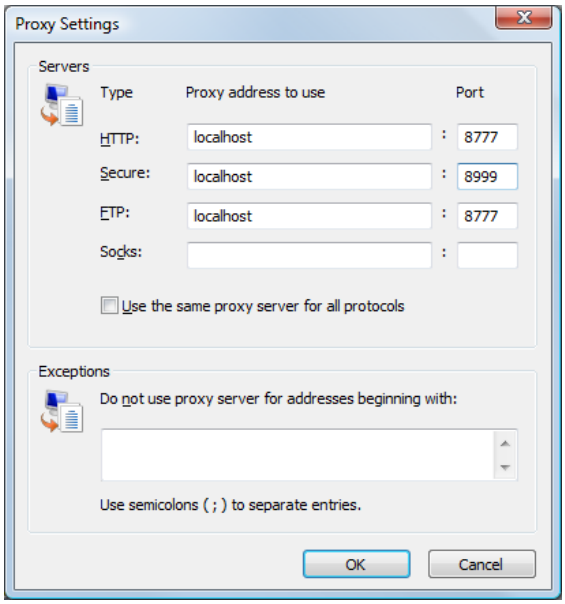
2 Click the *Connections* tab to display it.

3 Click the *LAN Settings* button to open the LAN Settings dialog box.



4 In the *Proxy Server* section, select the *Use a proxy server for your LAN* check box.

5 Click the *Advanced* button to open the Proxy Settings dialog box:



6 Specify the HTTP and Secure Server addresses and ports.

The default port for the HTTP server is 8777 and for the Secure Server is 8999.

7 Click the *OK* button.

7.3 Recording Steps for the HTTP(S) Test

To capture a test by using the HTTP Recorder:

1 Click the Windows *Start* button, then click *Programs* > *Experience Manager* > *Start HTTP Recorder* to start the Recorder.

2 Perform the steps to capture by browsing a Web application.

3 To stop the Recorder, click the Windows *Start* toolbar button, then click *Programs* > *Experience Manager* > *Stop HTTP Recorder*.

The test saves as an XML file in the `/OperationsCenter_ExperienceManager_install_path/tests/recorder` directory.

4 Change the Web browser proxy settings back to normal by deselecting the *Use a proxy server for your LAN* check box in the LAN Settings dialog box.

7.4 Editing Tests

The Experience Manager Monitor automatically generates test names. Consider changing to the name value for the tests and scenarios to something more meaningful because the tests imported into Operations Center use these names.

To edit a test:

1. Open any text or HTML editor.
2. Open the new test XML file saved in the */OperationsCenter_ExperienceManager_install_path*/tests/recorder directory.

   In the file, note that a section of <scenario> code exists for every Web page visited during the test:

```
<?xml version="1.0" encoding="UTF-8"?>
<tests>
    <!-- TEST [Yahoo]-->
    <test name="Yahoo" status="RUN" period="10" period_type="MIN" users="1" thread_limit="60" include_rollup="true" >
        <!-- SCENARIO [Web query] -->
        <scenario name="Web+query" type="shared" status="RUN" retries="0" continue_on_failure="true" rollup="true">
          <http
              follow_all_url_refresh="true"
              include_all_hrefs="false"
              meth="GET"
              useragent="Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"
              include_all_images="true"
              file="%2F"
              host="www.yahoo.com"
              include_all_frames="true"
              port="80"
          >
          </http>
        </scenario>
    </test>
</tests>
```

   The HTTP Recorder automatically configures a name value for the <scenario> tag.

   Consider changing this to something more descriptive. For example, in this case, the test name is Google Search, so change <scenario name="/search"> to <scenario name="Google Search">.

   IMPORTANT: Scenario and test names longer than 250 characters cause the test fails.

3. From the *File* menu, click *Save* to update the XML file.

7.5 Importing Tests

To import an XML test file into Operations Center:

1. In the Explorer pane, expand the *Elements* root element > *Experience Manager Adapter* > *Administration* > *Test Administration*.
2. Right-click *Tests*, then click *Import XML* to open the Test|Import dialog box.
3. Click the *Browse* button to select the generated test XML file from the */OperationsCenter_ExperienceManager_install_path*/test/recorder directory.
4. Click the *OK* button to import the XML file.

   The new test displays under *Tests* in the Explorer pane.

8 Tracking the User Experience on Web Applications

Use Experience Manager to monitor a variety of Web application properties, such as page uploads, downloads and render times. No modification of the browser environment or agents running on end users' machines is required, so users are not disrupted while they work with Web-based applications. The alarm information includes response time and data about the browser environment used to access the page.

The Experience Manager measures HTML page loading only, and does not track AJAX or scripting calls. It can measure response times for the Operations Center console's Web page (used for console login), but does not work with the Dashboard.

Use the procedures in the following sections to add minor code references to the monitored Web pages, or to provide server-side instrumentation and response time tracking:

8.1 Understanding User Experience Alarms

A JavaScript instrumentation file assists Experience Manager in capturing various information about the Web application's response, and are subsequently surfaced in alarms that report each response. For information on generating a JavaScript instrumentation file and instrumenting a Web application, see Section 8.2, "Configuring Web Application to Measure Response Times," on page 102.

Table 8-1 lists alarm information captured from Web applications.

Table 8-1 *Data captured when monitoring User Experience*

Alarm Property	Definition
responseTime	Total response time
downloadTime	Time to download and render the HTML Web page
uploadTransTime	Time required for the Web server to process the request
urlReferrer	Web page that referred the current URL
urlDocTitle	Title of the Web page
urlLocation	URL of the current Web page
brAppName	Browser name

Alarm Property	Definition
brAppCodeName	Browser code name (such as Mozilla*)
brAppVersion	Browser version
brClientIP	IP address of the end user's browser
brClientHost	Hostname of end user's browser
httpClientIP	NAT or proxy server IP address (if used, defaults to browser's address)
httpClientHost	Hostname of the NAT or proxy server host (if used, defaults to browser's hostname)
brAppPlatform	Operating system on which the browser is running
appScreenHeight	Browser's screen height
appScreenWidth	Browser's screen width
appAvailHeight	Browser's available height
appAvailWidth	Browser's available width
appJavaEnabled	Indicates if Java is enabled or disabled

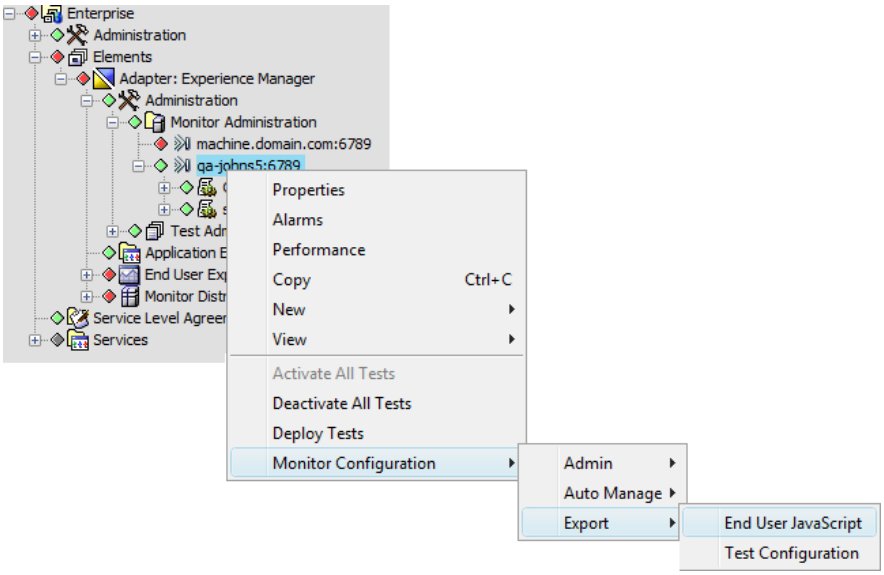
8.2 Configuring Web Application to Measure Response Times

Monitored Web pages must include specific code for each Experience Manager Monitor that tracks them. First you must export an instrumentation file, and then add a minor code snippet to each page you wish to monitor.

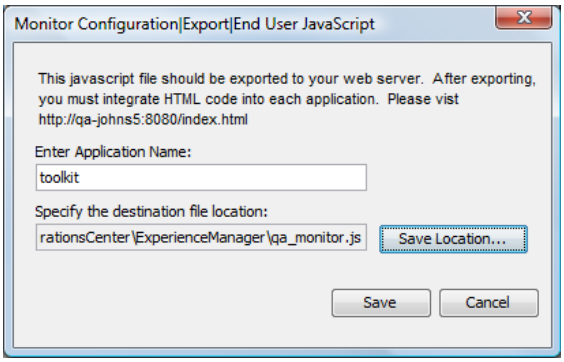
To instrument a Web Application to monitor Web page response times:

1 Do the following to generate a JavaScript instrumentation file for each Experience Manager monitor being use to collect data for the Web application:

1a In the Explorer pane, right-click the desired monitor, and select *Monitor Configuration* > *Export* > *End User Javascript*.



1b Type an application name for the Web application in the *Enter Application Name* field.



This name is used for the generated element (when data is available) in the Operations Center hierarchy under *End User Experience* > *End User Response Metrics*.

1c (Optional) Click the *Save Location* button to specify a destination file that includes a destination address and file name for the exported JavaScript code.

Perform this step only if users access the instrumented Web page via a proxy server or run a NAT server. If using proxy or NAT servers, the reported client IP might reflect the server's IP address, not necessarily that of the end user's browser.

1d (Optional) Customize the exported JavaScript file.

For information on customizing the file, see Section 8.3, "Customizing the Instrumentation Script," on page 104.

2 Using a Web browser, go to the following URL to verify the server is running and Operations Center can receive data:

`http://monitor-hostname:port/index.html`

Where, *monitor-hostname* is the name of the device where the monitor is running, and *port* is the port defined for the monitor's Web server.

3 Cut and paste the following sample code (with the appropriate modifications) into the target HTML page or template between the `<head>` and `</head>` tags:

```
<script type="text/javascript" src="JavaScript_File"></script>
<IFRAME src="about:blank" name="experiencemanager" width="0" height="0"
align="top" frameborder="0" scrolling="no"></IFRAME>
```

Where, *JavaScript_File* is the name and location of the exported file from Step 1 on page 102.

- Embed this code in each Web-based application that Experience Manager tracks. The location of the embedded code is called an instrumentation point.
- The number of files in which to embed the code varies. Using Header templates can reduce the number of instrumentation points across a Web-based application.
- Look for a consistent Header or other HTML utility text that exists on every Web page. There can be only one instrumentation point for an entire application.
- This code can also be embedded by using frames. Locate the outermost frame for the best results.

8.3 Customizing the Instrumentation Script

The End-User Experience component of Experience Manager uses the `urlLocation` alarm property to generate element structures under the monitor's hierarchy. Each element represents an individual page that is accessed by a user or a group of users.

By default, the `urlLocation` property contains the fully qualified URL, excluding any Post or Get parameters that are passed with the URL. By excluding '?' characters and long parameter strings, represented elements in the hierarchy can be easier to read and understand.

Configure the JavaScript file for the unique requirements of your application to:

- Capture specific parameters to differentiate users who access programmable pages that output data based on the parameters passed:

  protocol`://`*host*`:`*port*`/`*file_structure*`?and=`*parameters*

  `http://escrow.mosol.com:80/getProperties?dname=abc&document=true`

- Pass a unique session ID in the parameter list to identify every request a user makes as unique, when a Web-based application uses the same dynamic URL to generate various types of content.

  For example, an application uses the following to create new records in a system:

  `http://escrow.mosol.com:80/doAction?action=create&foo=bar`

  And uses a different URL to update records in the same system:

  `http://escrow.mosol.com:80/doAction?action=update&foo=bar`

To customize a JavaScript instrumentation file:

1. After generating the JavaScript file from (*Monitor Configuration* > *Export* > *End User Javascript*), open the JavaScript file in a text editor.

   For more information, see Step 1 on page 102.

2. Modify one or more of the following parameters to determine the values captured in the Operations Center alarm's `urlLocation` property:

   - **includeParamsInUrl:** Specify the parameter information to capture from the `document.location.href` value.

     Valid settings are (must be lowercase):

     - **none:** The `urlLocation` value won't include any parameter information.
     - **all:** The `urlLocation` value includes all parameters.
     - **list:** The `urlLocation` value includes (or excludes) only the parameters listed as determined by the `includeExcludeParamList`) setting.

     Defaults to `none`; no parameter information is captured.

   - **segmentedParamList:** Specify specific parameters to capture or exclude when `includeParamsInUrl="list"`. This list must be lowercase and comma delimited. Do not use any ending or trailing commas.

     Defaults to empty. No parameter information is included when `includeExcludeParamList="include"`. In contrast, all parameters are included if `includeExcludeParamList="exclude"`.

   - **includeExcludeParamList** Indicate if the parameters listed (specified in `includeExcludeParamList`) are captured or excluded. Valid values are `include` or `exclude`. Defaults to empty.

   For example, if we set:

```
includeParamsInUrl="list"
segmentedParamList="paramName1,paramName2,paramName3"
includeExcludeParamList="include"
```

where *paramName* instances are the names of parameters as found in the URL.

With the above settings, the instrumentation file parses all dynamic page executions (where the URL is appended with parameters), and records only the *paramName1, paramName2,* and *paramName3* parameters in the Operations Center alarm's `urlLocation` property.

The resulting `urlLocation` value based on this example is displayed as:

```
http://escrow/.mosol.com/action
(paramName=value)(paramName1=value)(paramName3=value)
```

Alternatively, we can set `includeExcludeParamList="exclude"` to capture all parameters except than those as defined in `segmentedParamList`.

9 Reviewing Synthetic Test Results

Each performed test generates a test record in the form of a Operations Center alarm. Each performance of the test updates the alarm. Obtain valuable information by reviewing the events and viewing the test record properties.

9.1 Reviewing Test Alarms and Performance Data

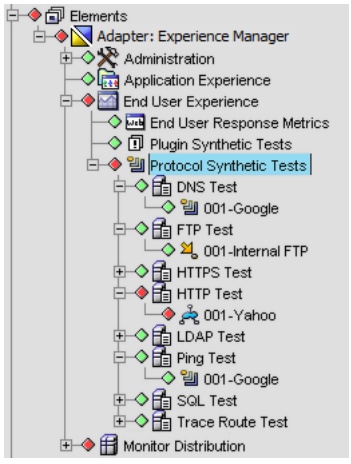
You gain valuable information from reviewing test events and test record properties. Critical alarms flag problem tests and displays details on the alarm property pages. Test events display as alarms in the Alarms view in the Operations Center console.

All test elements display in the Explorer pane, under the End User Experience element and the associated test type.

9.1.1 Viewing Test Results

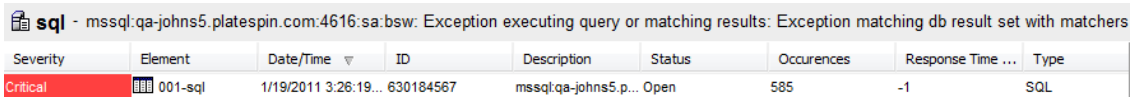
To view test results:

1 In the Explorer pane, expand the *Elements* root element > Experience Manager Adapter > *End User Experience* > *Protocol Synthetic Test*, then select a hostname.



2 Select a test name.

3 In the View pane, click the *Alarms* tab and the Alarms view is updated:



The severity and response times for each test display in the Alarms view. The Aggregate element displays the combined response time for all the tests. It also has a class of ROLLUP.

For more information on using the Alarms view, see *Filtering and Managing Alarms* in the *Operations Center 5.5 User Guide*.

4 To view the alarm properties for a synthetic test, right-click an alarm in the Alarms view, then click *Properties* to open the Status property page.

The named property pages vary depending on the test type.

Test Type	Property Page	Information Provided
All Synthetic Tests	Core Properties	The event type (synAlarm is synthetic alarm), the affected element, and other key information about the test performed, including actual, average, low and high response times.
	Synthetic Testing Metrics	The scenario name, synthetic test type, and target and test details.
E-Mail	Core Properties	Subdivides the response time into send and receive time.
HTTP		When there is a content match failure, custom alarm tabs display raw source code for the HTML response. These tabs vary depending upon the HTML response.
	HTTP Assets	A list of all images downloaded from the Web page and the amount of time taken to download. If a specific asset fails to download, it is noted and the event has a severity of MINOR.
	Certificate	If applicable, displays various certificate information including Issuer DN, Subject DN, Version, Algorithm Name, Expiration Date, and Serial Number.
SQL	Core Properties	Login time and query time.
Trace Route	Hops	A list of all hops made to access the specified host.

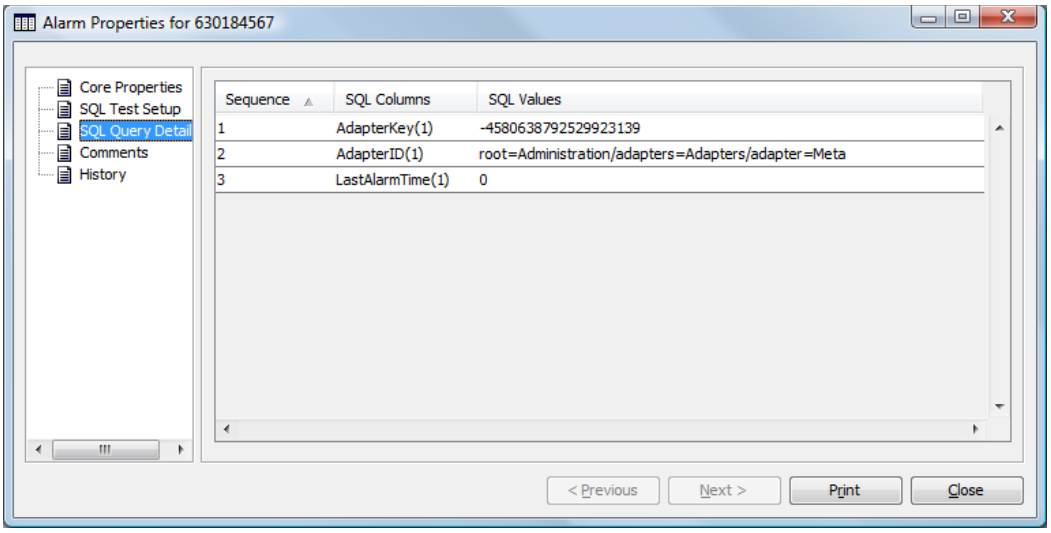
Understanding Alarm Status

When a HTTP/HTTPS scenario fails to retrieve an image, the event does not send a CRITICAL event. It continues processing and identifies the failed image in the alarm's *Assets* tab. The sent event has a MINOR severity in Operations Center.

In the alarm Core Properties property page, the Description text area includes the message `One or more of the assets had errors, see asset list`. The Application Name text box displays a message similar to `test image failure lab1.acme.com`.

The HTTP Assets tab displays “failed” in the *Response Times* column, next to the affected image:

Figure 9-1 *Response Times Column Displaying the Failed Message*



9.1.2 Charting Data in the Performance View

It is useful to graph the data received for some of the Experience Manager Monitor’s tests. Graphical information can effectively present statistics for response time over a period of time.

In Operations Center, use the Performance view to chart the following metrics available for a synthetic test:

- **Connection Time:** The elapsed time required to establish a connection (overlaps DNS time).
- **DNS Time:** The elapsed time required to complete the name resolution.
- **Download Time:** The elapsed time between starting the (HTTP) response and fully reading the response.
- **Response Times (Avg, High, and Low):** The elapsed time required to complete a full download.
- **SSL Handshake Time:** The elapsed time required to establish a SSL handshake.
- **Transaction Time:** The elapsed time between starting and responding to the start.

To compare response times over a time period, use the Performance view charting features. The line charts on the following page show response times for three different Web pages every 10 minutes over a one-hour period.

For additional information about using the Performance view, see *Charting Performance Data* in the *Operations Center 5.5 User Guide*.

To chart test data:

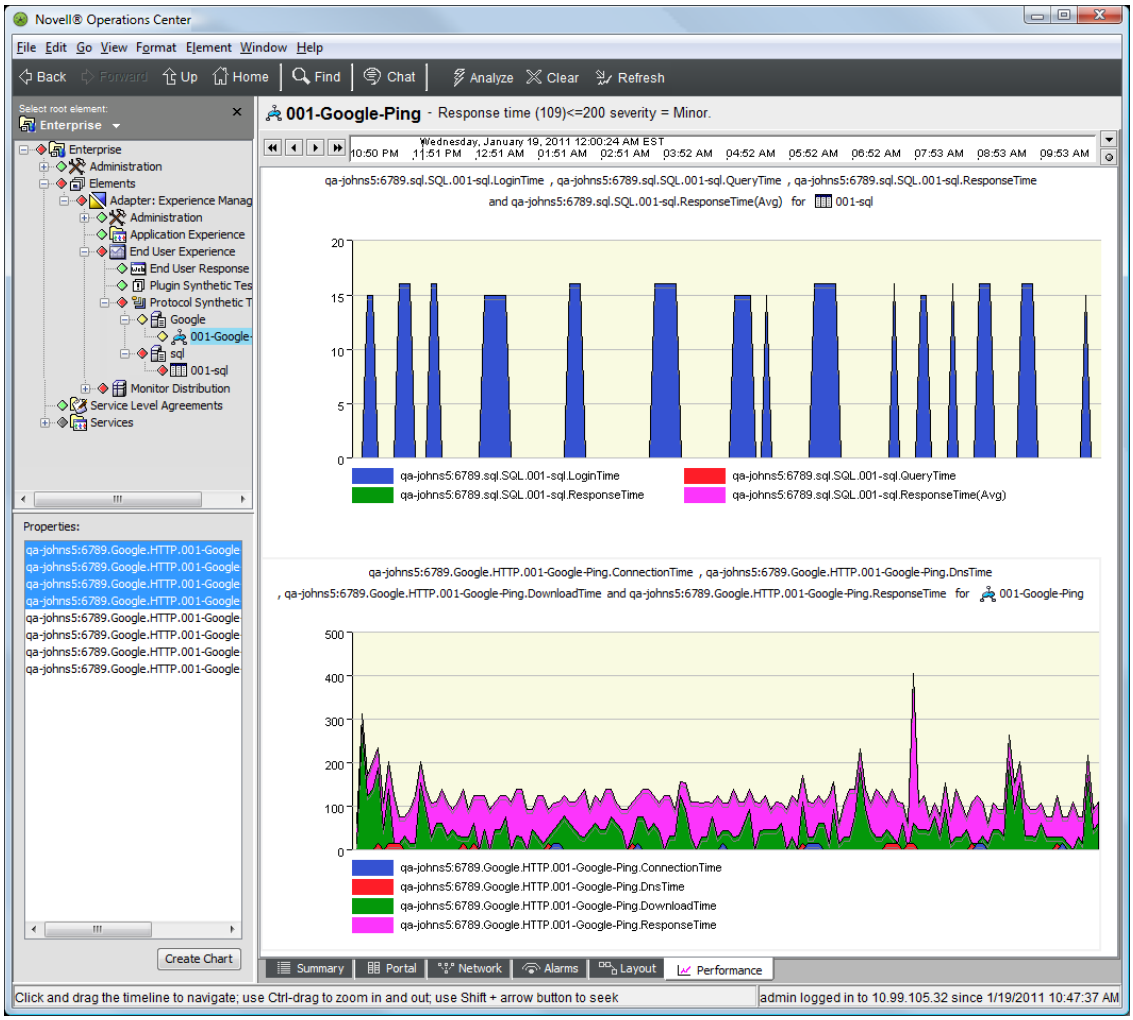
1. In the Operations Center console, click the *Performance* tab.

   The Performance view updates.
2. In the Explorer pane, expand *Elements* > Experience Manager Adapter > *End User Experience* > *Synthetic Testing Metrics*.
3. Select a test/scenario element.

   Associated chart properties display in the Properties panel.

4 Select a property, then click the *Create Chart* button.

A chart displays in the Performance view:



9.2 Performing Operations on Alarms

The following sections cover useful alarm operations for Experience Manager test alarms and describe how these vary from other integrations:

- Section 9.2.1, “Acknowledging Alarms,” on page 110
- Section 9.2.2, “Resetting Alarm History,” on page 111
- Section 9.2.3, “Rerunning a Test from an Alarm,” on page 111
- Section 9.2.4, “Deleting Test Elements after Closing Alarms,” on page 111

9.2.1 Acknowledging Alarms

When you acknowledge a test alarm, the alarm severity changes to *OK* and remains at *OK* regardless of the state of occurrences for the alarm. If the alarm is not acknowledged, the test alarm changes from *OK* to the current state.

9.2.2 Resetting Alarm History

Use the reset operation when you need to reset alarm counts and averages. For example, when introducing a new version of a Web server and the alarm history no longer applies.

To reset an alarm's response time values and occurrences, from the Alarms view, right-click an alarm, then click *Reset*.

Response time and occurrence values are set to 0.

9.2.3 Rerunning a Test from an Alarm

To rerun a test from an alarm, from the Alarms view, right-click a synthetic test alarm, then click *Run Test*.

The test associated with the alarm is run.

9.2.4 Deleting Test Elements after Closing Alarms

Experience Manager leverages Managed Object Definition Language (MODL) to generate elements for a received alarm. MODL consists of a DTD (Data Type Definition) that follows rules for structuring an XML document.

To automatically remove test elements after closing or deleting all their associated alarms:

1. Open the `database/examples /bemHierarchy.xml` file in any text editor.
2. Change `hold=yes` to `hold=no` for the furthest leaf nodes.
3. Move the modified `bemHierarchy.xml` file to the database directory so that Operations Center does not overwrite the changes.

10 Algorithms and Storing Experience Manager Properties

Experience Manager uses Managed Object Definition Language (MODL) to create a meaningful element hierarchy for the interpretation of metrics. MODL consists of a DTD (Data Type Definition) that follows rules for structuring an XML document.

You can manage element conditions in two ways:

- Use the threshold definitions for each synthetic test definition
- Use the algorithm's subsystem to calculate the state of MODL-generated objects based on response time values received from Experience Manager alarms

Use one or both of these mechanisms to promote the state in the Service Object Model:

10.1 Specifying Algorithms to Calculate Element Conditions

10.1.1 Understanding Algorithms

When Operations Center receives Experience Manager alarms with response times or other metric values, it promotes these properties to leaf objects (lowest level elements) and associates them via MODL. You can modify this association by using the MODL syntax in the `/database/examples/bemHierarchy.xml` file. For more information about how to generate elements and properties through MODL, see the *Operations Center 5.5 Adapter and Integration Guide*.

The ElementPropertyBand algorithm can discover these new values and determine the state of the elements based on banded thresholds. You can manually edit the `algorithms.xml` file to create algorithms using ElementPropertyBand or any of the other algorithms provided by Operations Center. Table 10-1 lists the default algorithms used for configuring Experience Manager elements.

Table 10-1 *Useful Algorithms for Experience Manager Elements*

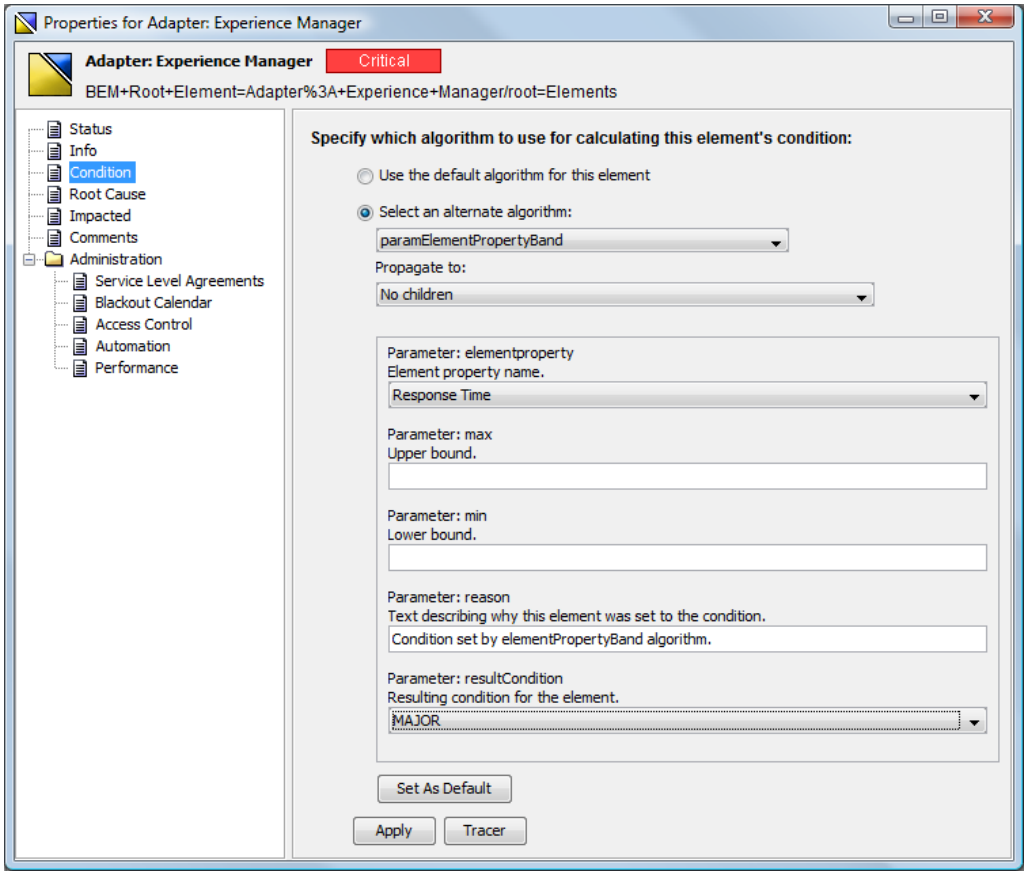
Algorithm	Description
bemEndUserBand	Default algorithm for any Experience Manager elements created as a result of metrics gathered through Web applications.

Algorithm	Description
bemSyntheticBand	Default algorithm for any Experience Manager elements created as a result of metrics gathered from synthetic tests.
paramBemSyntheticBand	Default algorithm that sets high and low bands for all alarm severity values. Appropriate to set for any Experience Manager element.
paramElementPropertyBand	Default algorithm that sets a single banded parameter based on the minimum and maximum values of an element property. This algorithm is similar to paramBemSyntheticBand, but sets only one threshold. Use this algorithm for testing purposes.

10.1.2 Setting the Thresholds for Experience Manager Elements

To set thresholds for Experience Manager elements:

1. In the *Explorer* pane, expand *Elements* > Experience Manager Adapter.
2. Right-click the Experience Manager Adapter element, then click *Properties* to open the Status property page.
3. In the left pane, click *Condition* to open the Condition property page.
4. Select the *Select an alternate algorithm* radio button.
5. Select the *Select an alternative algorithm* drop-down list, then select *paramElementPropertyBand* to use a single banded parameter to calculate condition.



6 Click the *Element* property name drop-down list, then select a property.

The illustration above selects the Response Time property.

7 Type the threshold values to define the maximum and minimum response time thresholds for the condition in the *Upper bound* and *Lower bound* text boxes.

Response times display in milliseconds.

8 Click the *Parameter: resultCondition* drop-down list, then select the condition to modify in order to include the newly defined minimum and maximum thresholds.

The illustration above selects MAJOR.

9 Click the *Apply* button to set the thresholds for the Experience Manager element:

10.2 Storing Experience Manager Properties in the Service Level Manager

10.2.1 Understanding Experience Manager Properties

The Service Level Manager (SLM) measures service levels and agreement health and compliance both in real time and historically. This section is relevant for users who have purchased and installed the Service Level Manager, which is an add-on to Operations Center. SLM measures service levels, and agreement health and compliance both in real time and historically. The SLM terms used in this section, such as profiles and expressions, assume that the reader is familiar with the product.

For more information about storing alarm history, see *Capturing Alarm and Performance History* in the *Operations Center 5.5 Server Configuration Guide*. For information about SLM and capturing Experience Manager metrics, see the *Operations Center 5.5 Service Level Agreement Guide*.

Out of the box, Experience Manager provides persistence of scalar performance information for response time and memory information (depending on the Experience Manager alarm type). It is also possible to store and analyze other information used in Experience Manager that surfaces either as alarm properties or promoted element properties (via MODL).

Because Experience Manager is a MODL-based integration, all data values enter Operations Center as alarms or through the alarms' associated properties. Based on these properties, users can create a hierarchy of elements which are the outcome of "break-outs" in the `bemHierarchy.xml` file.

Use this natural hierarchy to create Data Warehouse profiles that match these elements or hierarchy of elements. Also create profile expressions that pull alarm properties directly from the alarms associated with these elements and gather property values over a scheduled period of time.

Use the `/OperationsCenter_install_path/database/examples/bemHierarchy.xml` file to persist any Operations Center property in the Service Analyzer for further analysis and reporting.

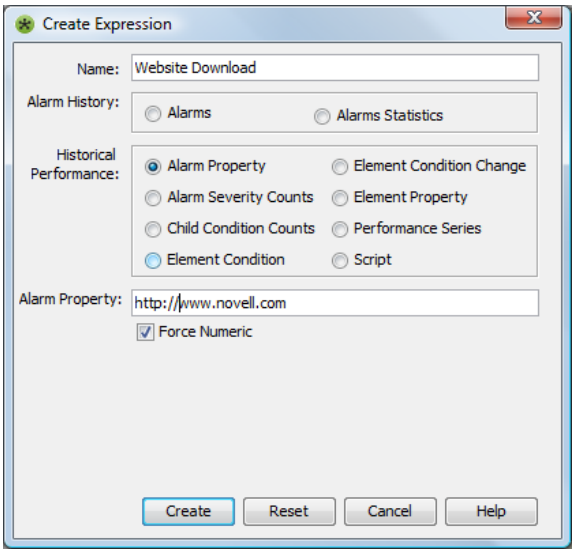
Use the SLM to set service level objectives for these property values to evaluate their impact on the state of the services that they support.

The following are general directions. For details on creating profiles and alarm expressions, see *Capturing Alarm and Performance History* in the *Operations Center 5.5 Server Configuration Guide*.

10.2.2 Storing Experience Manager Properties Using BSA

To store Experience Manager properties using BSA:

1. Identify the Experience Manager metrics to store as historical performance metrics.
2. If necessary, create a new profile with expressions to capture the desired metrics.
   - **2a** n the *Explorer* pane, expand *Administration* > *Data Warehouse* > *Profiles*. Right-click *Profiles* and select *Create Profile* to create a new profile.
   - **2b** Right-click the profile element and select *Create Expression* to create a new expresion for the profile.

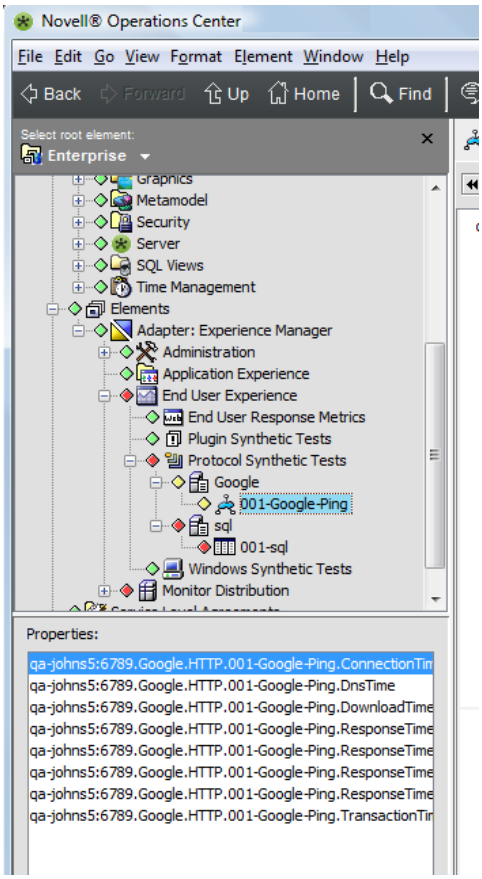


   - **2c** Right-click the profile element and select *Start Profile* to start the profile.

     For information about profiles, expressions, and matches. see *Capturing Alarm and Performance History* in the *Operations Center 5.5 Server Configuration Guide*.
3. In the *Explorer* pane, expand *Elements*, right-click the Experience Manager Adapter element, then click *Properties* to open the Status property page.
4. Click to view the various property pages and identify the metrics to store.
5. In the left panel, click *Performance*.

6 Select the desired profile.

The properties available for charting display in the Performance view as metric data accumulates.



For additional information about setting up profiles and gathering SLM data, see the *Operations Center 5.5 Service Level Agreement Guide*.

11 Secure Communications and Password Encryption

Operations Center can optionally use Secure Socket Layer (SSL) technology to encrypt event traffic between the Operations Center Experience Manager adapters and Experience Manager Monitors.

Operations Center can also use various encryption standards to encode passwords.

11.1 Enabling Secure Communications to/from the Experience Manager Monitor

Operations Center can use SSL technology to encrypt event traffic between the Operations Center Experience Manager adapters and Experience Manager Monitors.

The values for the Experience Manager Monitor's and the adapter's Secure Communications property must be the same or errors occur.

To encrypt communication to or from the Experience Manager Monitor:

1. Modify the Experience Manager Monitor's `/OperationsCenter_ExperienceManager_install_path/config/monitor.properties` file to include the following:

   ```
   Monitor.SecureCommunications=true
   ```

2. Modify the Operations Center Experience Manager adapter properties to set the Secure Communications property to true.

   See Section 3.2, "Creating a Experience Manager Adapter," on page 24 for information about setting adapter properties.

11.2 Encryption Mechanisms for Passwords

Operations Center provides two methods of data encryption using industry-standard encryption algorithms.

- **Password-based Encryption (PBE):** Defaults to using the DES encryption algorithm for generating a cryptographic key used to seed the encryption algorithm.
- **File-based Encryption:** Defaults to AES encryption to obtain a secure keyblock from a persistent file.

Configure both of the provided algorithm types with system settings provided to the virtual machine. Place these configuration values in the system properties of the virtual machine using the encryption libraries.

- Section 11.2.1, "Password-Based Encryption (PBE)," on page 120
- Section 11.2.2, "FILE-based Encryption," on page 120

11.2.1 Password-Based Encryption (PBE)

For the PBE-based encryption algorithm, use the following customization parameters. Add the parameters to both the */OperationsCenter_install_path*/config/formula.custom.properties and */OperationsCenter_ExperienceManager_install_path*/config/monitor.properties files. Table 11-1 lists the parameters for PBE.

Table 11-1 *Parameters for PBE*

Parameter	Description
System.Cipher.PBE.CipherName	Algorithm name (default: PBEWithMD5AndDES)
System.Cipher.PBE.KeyFactoryName	Default key factory name (default: PBEWithMD5AndDES)
System.Cipher.PBE.Passphrase	Password (default: fixed)
System.Cipher.PBE.Salt	Salt (default: fixed)
System.Cipher.PBE.IterationCount	Count (default: 17)
System.Cipher.PBE.ProviderName	the name of the JCE provider (default: JRE supplied provider)

The following is an example of system property settings used to enable a system-wide usage of the PBE algorithm:

```
System.Cipher.DefaultCipher=PBE
System.Cipher.PBE.Passphrase=crackme!
```

11.2.2 FILE-based Encryption

Table 11-2 lists the customization parameters to be used for the file-based encryption algorithm.

Table 11-2 *Parameters for File Based Encryption*

Parameter	Description
System.Cipher.FILE.CipherName	Name of the algorithm. Defaults to AES.
System.Cipher.FILE.KeyFile	Location of keyfile (URL or file).
System.Cipher.FILE.AbortOnMissing	Exits VM if keyfile is missing.
System.Cipher.FILE.ProviderName	Name of the JCE provider. Defaults to the JRE supplied provider.

This cipher requires the KeyFile parameter and contains the generated keyblock used to seed the algorithm.

- "Generating a Keyblock" on page 121
- "Using Longer Length Keyblocks" on page 121
- "Using an Encryption Library" on page 122

Generating a Keyblock

To generate a keyblock, use the provided utility:

```
mosjava com.mosol.util.security.crypt.DefaultFileBasedKeyCipherWrapper generate
keyfile [bits: 128,192 or 256]
```

For example, type the following:

```
mosjava com.mosol.util.security.crypt.DefaultFileBasedKeyCipherWrapper generate /
OperationsCenter_install_path/128aes.keyblock 128
```

The following example shows system property settings used to enable a system-wide usage of the PBE algorithm:

```
System.Cipher.DefaultCipher=FILE   System.Cipher.FILE.KeyFile=/
OperationsCenter_install_path/128aes.keyblock
```

A requirement is transmitted the KeyFile to the client software in a secure way if used for client-based authentication.

Using Longer Length Keyblocks

It is possible to use a keyblock length longer than the default 128-bit setting (for example, 192 or 256 bit AES encryption keyblock length).

To use a keyblock length longer than the default 128-bit setting:

1. Download and install the unlimited strength JCE policy files from Sun, which is subject to export restrictions. These files are at: http://java.sun.com/j2se/1.4.2/download.html#docs (http://java.sun.com/j2se/1.4.2/download.html#docs).

   Perform this step for each virtual machine, including the Operations Center server, Experience Manager remote Monitor, Operations Center console (operations client), etc.
2. Generate the keyblock using the utility provided above. Supply either 192 or 256 as the keyblock length.

Using an Encryption Library

To use the encryption library for only one of the Operations Center subsystems, such as Experience Manager:

1. Set a configuration value in the */OperationsCenter_install_path*/config/formula.custom.properties file that directs the configuration type or subsystem to use a particular mode.

   For example, the following statements direct the Experience Manager integration to use the FILE-based algorithm for encrypting data. To configure a Experience Manager Monitor to handle encryption in the same way, place the same values in the Experience Manager monitor.properties file.

   ```
   System.Cipher.Configured.BEM=FILE
   System.Cipher.FILE.KeyFile=/OperationsCenter_install_path/keyblock.aes
   System.Cipher.FILE.AbortOnMissing=true
   ```

12 Scripting Layer

The Experience Manager scripting layer is a feature of the Experience Manager monitor, and used to extend and enhance several aspects of the monitor's operation. This document provides reference information necessary to implement scripts, and describes those monitor features that support scripting and how they can leverage the scripting functionality.

12.1 The High-Volume Metrics Framework

High-Volume Metrics (HVM) and End-User Metrics (EUM) are frameworks in the Experience Manager product that monitor the performance of Web sites as experienced by their users. Both frameworks collect performance data by embedding an instrumentation script in the pages of the Web sites to be monitored (the target Web sites). The instrumentation script is a customized JavaScript file that is provided with the product. As users browse through the target Web site, the instrumentation script executes in the user's browser, measuring the performance of the target site and reporting this information to a Experience Manager monitor.

The End-User Metrics framework is designed to track individual hits by individual users on the target Web site. As performance data is received from the end user browsers, the monitor forwards this information to any connected Operation Center servers in the form of an alarm. There is typically a one-to-one correspondence between user page hits and Operations Center alarms that warehouse the resulting metrics. Each hit, on each page, from each end user system is reported as a Operations Center alarm. (The adapter hierarchy file is used to organize these alarms into Operations Center elements.) Subsequent hits from the same system on the same page updates that alarm, making it possible to track the history from the perspective of a particular IP address and/or a particular Web page. The EUM framework is quite simple to set up and is appropriate for use on intranet sites with no more than a few thousand users.

The High-Volume Metrics framework implements more scalable and flexible approach designed for Internet or intranet sites with an unlimited number of users. Instead of feeding the performance data for individual hits on the target Web site, data collectors are used to collect customized information about activity on the target site. The collected data are periodically reported to scripted handlers that use it to create and populate a custom *Operations Center* element hierarchy to capture collected metrics. Depending on several variables, including complexity of the HVM algorithms and the hardware where it is running, a single monitor can process up to 20,000 hits from end user browsers per minute. For environments with a higher flow rate, multiple monitors can be configured in a cluster to collaborate in collecting end user data to feed a common set of HVM algorithms.

While a sample HVM configuration is provided with the Experience Manager installation, initial configuration and customization of the HVM framework is much more involved than is set up for EUM.

12.1.1 Instrumentation Script

The instrumentation script produces the seed data that drives the HVM framework. It is included in each Web page of a monitored Web site and executes inside the browser of each end user that browses the site. As end users navigate from page to page in the monitored Web site, this script calculates the time required to navigate from one page to the next. After each page is loaded, the script sends a message to a Experience Manager monitor that includes information about what page was loaded and how long it took. The time reported by this script includes both time to retrieve the page components from the Web site, as well as the time to render those components in the end user browser.

How the Instrumentation Script Works

The instrumentation script calculates performance data by capturing three time points:

- **Before-unload:** The `onbeforeunload` event is a standard event that the browser fires just before a new page is requested. For example, right after the user clicks a hyperlink, but before the browser sends a request to the new URL. The time this event fires is stored in a cookie so that it is visible to the script when the next page loads.
- **Page-initialization:** This time point is captured immediately after the headers for the new page are loaded, but before the document body and its components are loaded and rendered.
- **Load-complete:** The `onload` event is a standard event that is fired after the browser has completely loaded and rendered a page.

As a rule, the response time reported in HVM is the difference between the “before-unload” and the “load-complete” event. The exception to this rule is when the user enters the site—either for the first time, or after navigating away. In this case, only the period between “page-initialization” and “load-complete” is used.

Browser security restricts visibility of navigation history, so this special case is detected by setting a maximum threshold for the interval between “before-unload” and “page-initialization”. (This interval is called “upload time” and the threshold limiting its use, called `uploadTimeLimit`, is set in the preprocessor.)

When “load-complete” is detected, the browser sends the URL parameters listed in Table 12-1 to a Experience Manager Monitor.

Table 12-1 *URL parameters sent to the Experience Manager Monitor by the browser.*

Field Name	Parameter Name	Description
Application Name	“an”	Name of the application being monitored. This value is configured in the instrumentation script. By using different values, different Web sites or applications within a Web site can be easily distinguished.
URL	“rl”	URL of the loaded page as retrieved from the browser’s “document” object.
Title	“ti”	Title of the page as retrieved from the browser’s “document” object.
Upload Time	“ut”	Number of milliseconds between “before-unload” and “page-initialization.”
Download Time	“dt”	Number of milliseconds between “page-initialization” and “load complete.”
User Agent	“ua”	Describes the user’s browser and version. Retrieved from the browser’s “navigator” object.

Upon receiving this request from the browser, the Experience Manager monitor interrogates the TCP/IP connection from the browser to retrieve the IP address of the client that sent the information and include that with the received data. Note, this IP address is not useful if the monitor is sitting behind a Virtual IP server.

How to Use the Instrumentation Script

A template of an HVM instrumentation script is included with the Experience Manager installation in the *`OperationsCenter_ExperienceManager_install_path`*`/hvm-template/` directory. Before loading the file onto a Web site you wish to monitor, you must set configuration parameters located at the top of the file. Typically, the only variables that might need to be changed are the following:

- **appName:** Set this variable to identify the application being monitored.
- **posttLoc:** Set this variable to the URL of the Experience Manager monitor Web server. By default, this is:

  http://*monitor_machine_name*:8080

After you have set these variables, save the updated file and load it onto the Web server that is to be monitored. Update the `<HEAD>` section of each page to include an empty IFRAME called `experiencemanager` and a reference to the JavaScript file. The following is an example from the `index.html` file on your Monitor:

```
<!-- START Experience Manager Instrumentation Hooks -->
<script type="text/javascript" src="/HVReporting.js"></script>
<IFRAME src="about:blank" name="experiencemanager" width="0" height="0"
align="top" frameborder="0" scrolling="no"></IFRAME>
<!-- END Experience Manager Instrumentation Hooks -->
```

12.1.2 Monitor Components

The Experience Manager Monitor is the core of the HVM framework. It is responsible for receiving and analyzing the raw performance data from thousands of browsers and generating meaningful metrics information and delivering those metrics to Operations Center.

All HVM components on the monitor are implemented with user-defined "HVM scripts" that define how data is collected and how collected data is reported to Operations Center. The scripting layer defines the "metrics" object that exposes the services and events needed by HVM scripts to perform these tasks.

The remainder of this section gives an overview of these components to help you understand how they fit together. Examples in the Script Reference section of this document, as well as the DemoMetrics configuration included in the Experience Manager Monitor installation, provide deeper insight.

- "Data Collection" on page 126
- "Report Processing" on page 128
- "Configuration" on page 128

Data Collection

When the monitor is ready to start processing HVM data, it fires the "metrics.onload" event. During its handling of this event, an HVM script should define a preprocessor object (see "metrics.createPreprocessor") and one or more data-collector objects (see "metrics.creatSummaryCollector"), then deploy these objects using the "metrics.deploy" function. The configuration of these deployed objects determines how data is collected from the stream of events received from the instrumentation script running in end user browsers.

The preprocessor is composed of FieldFilter, FieldXForms (transformation rules), and rules for handling the received upload time. Using these value, the preprocessor scrubs each event record, normalizing data and filtering out any unwanted records. The monitor then uses the `uploadTimeLimit` value in the preprocessor to calculate the total response time for the event record.

Each event record that is not filtered out by the preprocessor is then evaluated be each deployed data collector. Data collectors collect information about the records received. In its smallest form, a data collector must define a name, an intervalDuration, and a function to which it delivers reports at the end of each interval. For example:

```
var collector = metrics.createSummaryCollector();
collector.name='example1';
collector.intervalDuration=60000;
collector.onreport=myReportHandler1;
```

As written, this example summary data collector calls myReportHandler1 once a minute (60000ms) and pass it a summary report, including the count and response time information (average, maximum and minimum values) for each event record received during that minute.

However, data collectors include several optional properties that allow for much richer data. For example, adding the following line to the example collector causes the collector to generate a separate report for each unique page title encountered during the reporting period:

```
collector.groupingFieldNames=[metrics.constants.FN_PAGE_TITLE];
```

Now, instead of generating a single report summarizing all the hits on an instrumented Web site, the data collector generates a separate report for each unique page title encountered during the interval. However, we might be using the HVM framework to collect information on more than one Web site, and those sites could have overlapping names. In that case, the groupingFieldNames property could be extended to include application name:

```
collector.groupingFieldNames=[metrics.constants.FN_APPLICATION_NAME,
                              metrics.constants.FN_PAGE_TITLE];
```

Now, assuming the instrumentation scripts on the two Web sites are configured with different application name values, the reports for those Web sites are properly segregated. Each unique combination of application name a page title produces a separate report.

In the examples above, groupingFieldNames were used to track metrics based on unique values sent from the instrumentation script. In some cases, the data from the instrumentation script does not provide the desired granularity. An example of this is a use case where data was tracked by browser type. The instrumentation script does not send browser type as one of the standard fields. It is however, possible to derive browser type from the user-agent field that is sent by the instrumentation script. To accomplish this, a set of FieldXForms in the preprocessor is needed to derive the browser type.

In the following snippet, FieldXform objects are created that identify Microsoft Internet Explorer and Mozilla Firefox browser types, the a collector is created to track hits by each of these types:

```
var iex = metrics.createFieldXform('IEX');
iex.fieldName=metrics.constants.FN_USER_AGENT;
iex.matchExpression='.*MSIE[^;]*.*';
iex.replaceExpression='Microsoft Internet Explorer';
iex.destFieldName='BrowserType';

var ffx = metrics.createFieldXform('FFX');
iex.fieldName=metrics.constants.FN_USER_AGENT;
iex.matchExpression='.*Firefox\S*.*';
iex.replaceExpression='Mozilla Firefox';
iex.destFieldName='BrowserType';

var preprocessor = metrics.createPreprocessor(3000, [iex,ffx],null);


var collector = metrics.createSummaryCollector();
collector.name='BrowseTypeCollector';
collector.intervalDuration=60000;
collector.onreport=myReportHandler2;


collector.groupingFieldNames=['BrowserType'];
metrics.start(preprocessor,[collector];
```

The HVM framework provides a broad, flexible set of tools to account for a variety of different Web site architectures. In the DemoMetrics example, provided with the Experience Manager Monitor installation, reports are generated by page, but in this case, Page Title is not assumed to be a valid way of distinguishing the page that was hit. In that case, the URL is passed through a FieldXform to determine the page that was hit, then a collector groups its reports based on those derived values.

Report Processing

At the end of each defined interval, a data collector calls their report handler functions and reports any activity it collected during that interval. If the data collector has grouping field names defined, then it delivers one report for each unique set of field values encountered during the interval.

The report handler extracts information from received reports and determines how the gathered information should be represented in the *Operations Center* element tree. Handlers can be very straightforward and simply map the reports directly into elements. For example:

```
function myReportHandler1(reports)
{
   for(var i=0; i < reports.length; i++)
   {
      var elem = metrics.createMetricsElement(reports[i], true);
      metrics.sendMetricsElement(elem);
   }
}
```

With very little script, this example captures virtually all of the data captured in the reports to Operations Center element. The element names reflect the name of the collector, as well as any grouping field values used to generate the report. The element properties reflect the current values collected, as well as details of the collector configuration, and the element is updated with series data containing the hit count and average, maximum, and minimum response times.

On the other hand, report handlers can be much more sophisticated. They can create and update any number of elements. It can add or update its own custom properties on those element, and it can store whatever series data is required with the element. There is no restriction as to how many elements can be created by a report handler, or how many report handlers can write to a given element. The only restriction is that for a given element, you might not write to the same series data name and time point more than once. So, for instance, in the report handler for the BrowserTypeCollector referenced above, we could just capture the percentage of hits on each browser and save that information as property and/or series data on an element that is also updated by another collector.

The report handler in DemoMetrics.js is much more sophisticated. It uses a number of page-specific configuration parameters to implement custom business rules. In that case, the condition of an element is driven by the percentage of hit within a predefined range to determine whether the page is performing at an appropriate level.

Configuration

An HVM implementation requires its own properties file and at least one script (`.js`) file. The name of the properties file is set in the "Metrics.properties" property in the `monitor.properties` file. For example, the default installation is configured as follows:

```
Metrics.properties=DemoMetrics.properties
```

The referenced properties file must identify the main script file of the HVM implementation and any configuration properties required by that script. It is recommended that HVM scripts not use the HVMetrics. prefix for its configuration properties. In a distributed data collection environment, it must also specify the HVM group properties (see Section 12.1.4, "Grouping Monitors for distributed HVM data collection," on page 131).

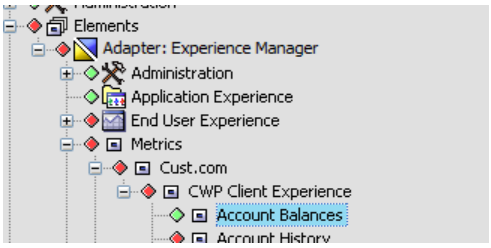
Errors in either the properties file or the HVM script file causes the monitor to halt.

12.1.3 Operations Center Data Repository

The Experience Manager Monitor delivers metrics element data to any Experience Manager adapter managing the monitor. (It also caches data for unreachable adapters per the events.log feature.) Upon receiving element data from a monitor, the adapter adds or updates an element with a hierarchy matching the MetricsElement.elementKeys under the *Metrics* branch of the adapter's element hierarchy.

The highlighted element in the hierarchy shown in Figure 12-1 was created with a MetricsElement with element keys `['Cust.com','CWP Client Experience','Account Balances']`.

Figure 12-1 *Adapter Hierarchy*

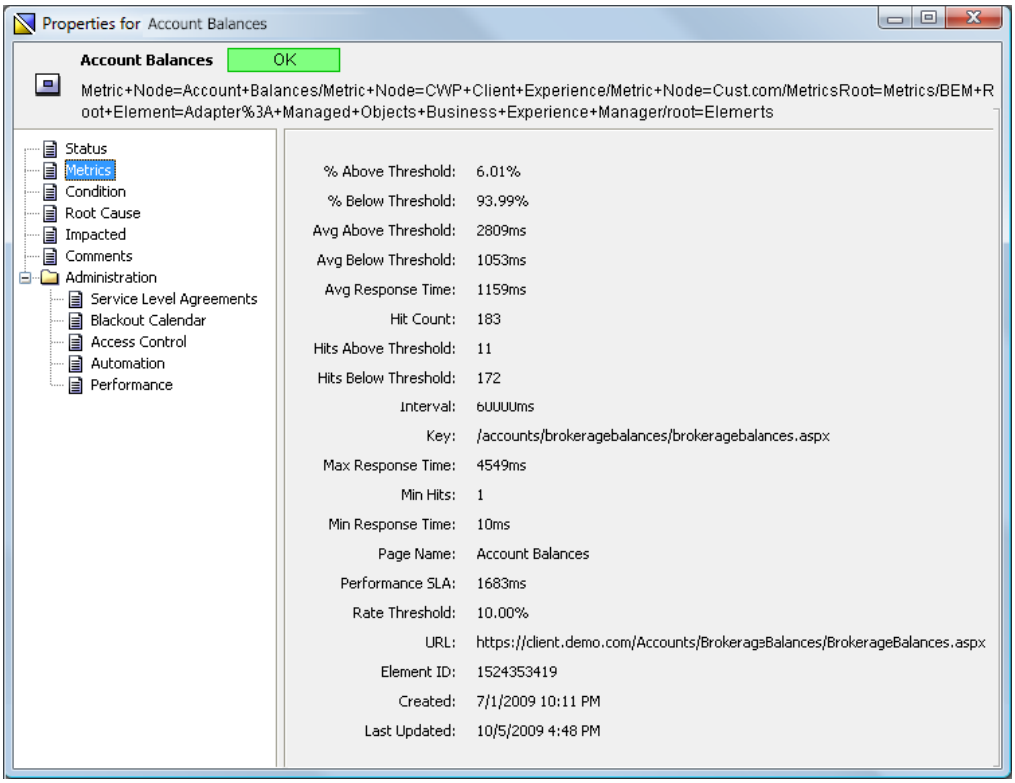


It is not necessary to explicitly create parent elements in the hierarchy.

Metrics Elements are purely a reflection of the data written to them from the HVM script. Adapter properties and operations on these elements are provided to allow clean up of unwanted data. Otherwise the element content is entirely driven by the HVM script.

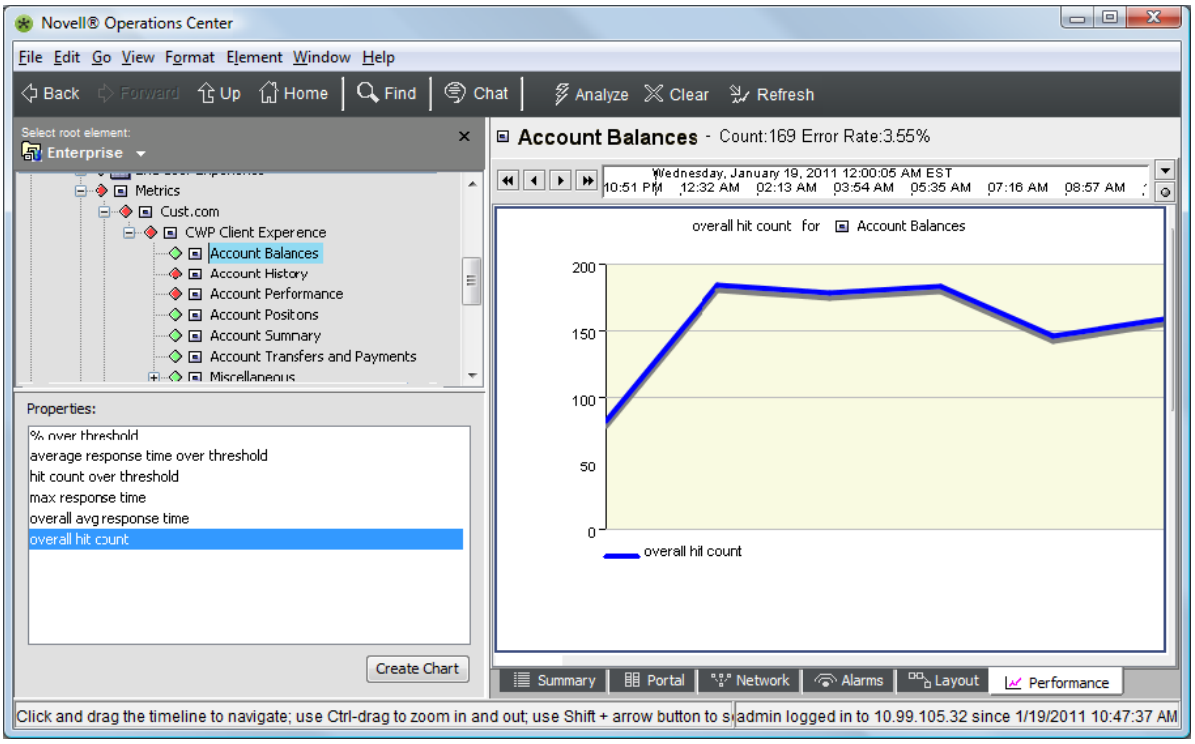
Element properties defined by the HVM Script are exposed on the properties page for the element:

Figure 12-2 *Properties Page: Account Balances*



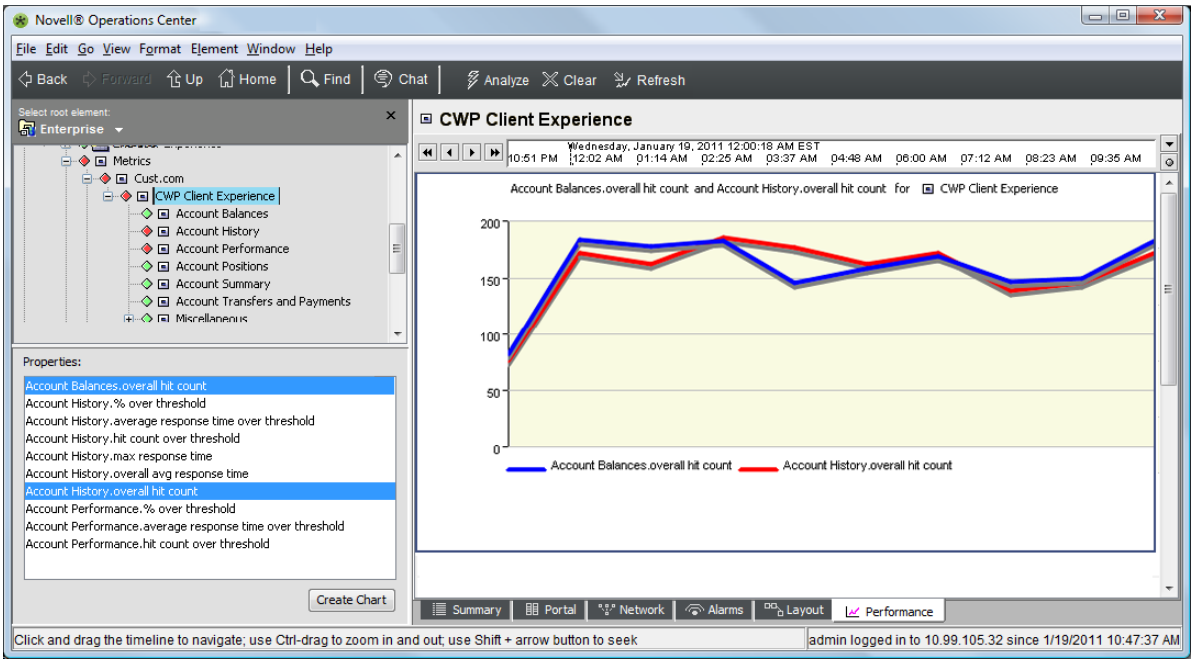
Series data attached to the element is exposed in the *Performance* view:

Figure 12-3 *Performance Tab*



Performance data of children is exposed up the element hierarchy, making it easier to compare data across elements:

Figure 12-4 *Performance Data*



12.1.4 Grouping Monitors for distributed HVM data collection

When monitoring a Web site that has very high volume (greater than 20k hits/minute) it is possible to configure multiple Experience Manager monitors to act in concert to collect and process event records from end user systems. In this configuration, all monitors in the group typically sit behind a load balancer that distributes incoming event records among the available monitors. The load balancing functionality, virtual IP address, and so on are external to the Experience Manager product.

When acting as a group, one monitor assumes the role of group controller and all others act as slaves to the controller. The controller has the responsibility of distributing information about deployed preprocessor and data collector objects to the members of the group, collecting reports from the group members at the end of each reporting interval and merging the reports into a single set of reports representing the data collected by all members of the group. Slave monitors in an HVM group typically keep their Web servers shut off until they receive deployment instructions from the controller monitor (see `Metrics.manageWebServer` in `monitor.properties`).

Setting up a group of monitors is driven from the HVM properties file and requires little or no change to the HVM script. The controller acts very much like a monitor collecting HVM data in the stand alone mode. When it is ready to start the HVM feature, it calls the metrics.onload event handler in its local environment, and waits for the handler to deploy a preprocessor and a set of data collectors. At that point the controller updates the local environment and sends copies of the configured preprocessor and data collectors to all other group members. Each group member then begins to collect and process event records from end user browsers using this shared deployment. At the end of each reporting interval, the controller collects reports from the local environment, as well as all group members. The data in these reports is then merged into a single set of reports representing the data collected by the entire group. These reports are then passed to the `onreport` handler for the data collector just as they are in a stand alone environment.

It is vital that all members of an HVM group work from the same group configuration. Group configuration properties are designed such that the same properties file can be used, without modification by all members of the group. (Either copies of the same properties file, or a common file accessed on a network share.)

A complete set of HVM group properties are included at the bottom of the included `DemoMetrics.properties` file. The most important properties are described here:

- **HVMetrics.group.controllers:** Ordered list of monitors that could act as controller in a group. Monitors are specified in *monitor_name*:*monitor_port* format (such as `devtower18.mosol.com:6789`) Members of the list are separated by commas. The first entry in the list is always the group controller, unless that monitor is not running. Any later members only assume control of the group if they time out, waiting for control by a higher precedence controller.
- **HVMetrics.controller.protomotionTimeout:** Specifies the amount of time (in ms) that a backup controller waits for connectivity with a higher-precedence controller before assume the active controller role. If multiple backup controllers are specified, then this timeout must expire for each of the higher-precedence controllers before activating.
- **HVMetrics.group.members:** Comma-separated list of all monitors in this group (in the same format described for controllers).

12.1.5 Testing Tools

The `/eventPump` directory in the Experience Manager monitor installation contains a test tool that can be used to generate simulated event records and send them to a Experience Manager monitor. The event pump tool is driven by a properties file that identifies the monitor to which events should

be sent, the frequency with which they should be sent, and the range of values that should be used in generating events. To use the `eventPump` tool, configure a properties file with the desired information using the provided sample as a template and entire the following command:

```
java -jar eventpump.jar properties_filename
```

The `eventPump` tool is not dependent on the monitor installation structure. You can copy the JAR and properties files to any system with an appropriate JRE and you can run as many concurrent instances as you wish.

12.2 Managing Synthetic Tests with Scripts

The Experience Manager Monitor scripting layer includes a testEngine object that allows user-defined scripts to monitor and influence the operation of the monitor's synthetic test engine. The testEngine object includes methods that allow scripts to retrieve information about deployed tests and to change the run status of any deployed test or test scenario. It also defines fields (JavaScript properties) in which a script can define handlers to monitor changes in the test configuration, and the results of test and scenario execution. For a complete description of the testEngine object, see Section A.1.3, "The "testEngine" object," on page 148.

As an example, suppose we had a test called `'WebTest'` that monitored a Web site and another test called `'Diagnostics'` that diagnosed potential causes in the event `'WebTest'` had errors. We only want `'Diagnostics'` to run if `'WebTest'` fails. To implement this logic, we should create a Javascript function that implements the basic logic by processing a SyntheticTestResults object. For information about the SyntheticTestResults object, see Section A.2.18, "SyntheticTestResults," on page 177).

```
function myHandler(testResult)
{
   if(testResult.test.name.equals('WebTest'))
   {
      if(testResult.getFailCount() > 0)
      {
          testEngine.activateTest('Diagnostics');
      }
   }
}
```

Then we should designate myHandler as the listener for the OnTestComplete event:

```
testEngine.onTestComplete=myHandler;
```

When this script is loaded as the test management script, the `'Diagnostics'` test is started after the first failure of `'WebTest'`. However, the example is not complete. As it is scripted, `'Diagnostics'` runs continually after it is activated. What we really want is for it to run only once after a failure of `'Web Test'`. To fix this, we will extend `'myHandler'` to also look for `testResults` from our `'Diagnostics'` test:

```
testEngine.onTestComplete=myHandler;
function myHandler(testResult)
{
   if(testResult.test.name.equals('WebTest'))
   {
      if(testResult.getFailCount() > 0)
      {
          testEngine.activateTest('Diagnostics');
      }
   }
   else if(testResult.test.name.equals('Diagnostics'))
   {
      testEngine.deactivateTest('Diagnostics');
   }
}
```

To deploy a test management script, both the script file and an associated properties file are required. The properties file specifies the name of the script file, as well as any configuration parameters needed by the script. The example script from above does not use any configuration parameters, but it certainly is more flexible if the names of the tests `'Web Test'` and `'Diagnostics'` could be changed without editing the script itself. To do this, we would change the hard-coded with variables that are loaded with values set in the test management properties file:

```
var monitoredTestName=testEngine.properties.get('example.monitorTestName',null);
var diagnosticTestName=
    testEngine.properties.get('example.diagnosticTestName',null);

if(monitoredTestName != null && diagnosticTestName != null)
{
   testEngine.onTestComplete=myHandler;
}

function myHandler(testResult)
{
   if(testResult.test.name.equals(monitoredTestName))
   {
      if(testResult.getFailCount() > 0)
      {
          testEngine.activateTest(diagnosticTestName);
      }
   }
   else if(testResult.test.name.equals(diagnosticTestName))
   {
      testEngine.deactivateTest(diagnosticTestName);
   }
}
```

The name of the test to monitor and the diagnostics test are loaded from properties `example.monitorTestName` and `example.diagnosticTestName` respectively. In addition, we added logic so that if either of these properties is not set, the event handler is not configured.

To deploy this management script, we should save the script to a file called `example.js` in the monitor's scripts directory, then we should create the properties file `example.properties` in the `/config` directory. Following is the content of `example.properties`:

```
tests.management.script=example.js
example.monitorTestName=WebTest
example.diagnosticTestName=Diagnostics
```

Finally, we must tell the monitor that we want to use this test management configuration. To do that, update `monitor.properties` as follows:

```
tests.management.properties=example.properties
```

You must then restart the monitor for these changes to take effect.

That covers the basics of how to manage synthetic tests with scripts. The example we used is fairly simple, but covers the core concepts. A slightly more sophisticated example is included with the monitor install called `'FailOverDemo'`. In that example, we extend the basic concepts shown here by allowing multiple tests to be monitored, each with a unique configuration. While the script and its properties are more complex, the sophistication comes from more advanced Javascript. It uses the same basic hooks shown here.

12.3 Invoking Scripts through the Experience Manager-Monitor Web Server (WebOps)

The Experience Manager-Monitor scripting layer allows certain scripted functions to be invoked through the monitor's Web server through a feature called Web-Invocable Scripted Operations (WebOps). This feature allows custom functionality to be accessed by remote systems or custom Web pages hosted on the monitor's Web server. In the default configuration, only diagnostic WebOps are enabled:

- **threadDump:** Returns a dump of all threads in the Experience Manager-Monitor's process space.
- **heapDump:** Dumps the current Experience Manager-Monitor heap data to a file and returns the file path to the caller.

These and other sample WebOps are included in the `WebOps.js` file in the Monitor's "scripts" directory. WebOps can be invoked by pointing a browser to /Plugins/invoke.pl?op=*WebOps_name* on the monitor's Web server. For example, to assuming a monitor with the default configuration is running on the local system and the Web server is listening on port 8080, pointing a browser to http://localhost:8080/Plugins/invoke.pl?op=threadDump returns a thread dump for the running monitor.

Custom WebOps might be coded to perform virtually any action available to the monitor scripting layer. Web server requests to /Plugins/invoke.pl are routed to the WebOps framework. Upon receiving a request, the framework searches for a valid WebOps specified by the `op` parameter. If found, all of the parameters in the request are bound into a java.util.Map object and passed as an argument to the operation. Anything returned by the operation is automatically returned to the caller as an HTML string. Refer to the included `WebOps.js` file for examples.

Two settings in `monitor.properties` are used to manage WebOps functionality:

- **Monitor.webserver.ops.script:** Specifies the name of the script file containing any defined WebOps. Only JavaScript functions that are first loaded with this file are eligible to be invoked through the WebOps framework. If this property is left blank, WebOps functionality is disabled.
- **Monitor.webserver.ops.functions:** Specifies the names of Javascript functions that can be invoked as WebOps. If this property is left blank, any function loaded in the specified WebOps script can be invoked.

A Experience Manager Script Reference

Experience Manager Script refers to a JavaScript implementation with a customized context that allows access to various components and services of the Experience Manager monitor. Experience Manager Script is used to implement High-Volume Metrics (HVM), Web-invokeable operations (WebOps), and synthetic test management scripts.

A.1 Predefined Objects

These objects are predefined in the Experience Manager scripting environment.

A.1.1 The “bem” object

The `bem` object provides access to various services common to all Experience Manager monitor script-based applications. This object is predefined and always available in all contexts.

bem.load()

Loads a script source file into the current context. The fileName argument can be the name of a script file in the Monitor’s scripts directory, the path to the script file relative to the scripts directory, or an absolute file path. If the file cannot be found, or cannot be parsed by the script interpreter, an exception is thrown.

Valid Signatures

void bem.load(String *file_name)*	Loads the specified script file.

Returns

None

bem.logDebug

Generate a log message at DEBUG level. All scripting logs are recorded in the "Scripting" log category. Messages can be allocated to a subcategory of "Scripting" using the optional category argument.

For example `bem.logDebug('testCategory', 'test message')` writes `test message` to the logger at DEBUG level, with the category `Scripting.testCategory`.

Valid Signatures

void bem.logDebug(String *message_text*)	Generates a log message in the default category.
void bem.logDebug(String *category_name*,String *message_text*)	Generates a log message in the specified category.

Returns

None

bem.logError

Generate a log message at ERROR level. All scripting logs are recorded in the "Scripting" log category. Messages can be allocated to a subcategory of "Scripting" using the optional category argument.

For example `bem.logError('testCategory', 'test message')` writes `test message` to the logger at ERROR level, with the category `Scripting.testCategory`.

Valid Signatures

void bem.logError(String *message_text*)	Generate a log message in the default category.
void bem.logError(String *category_name*,String *message_text*)	Generate a log message in the specified category.

Returns

None

bem.logInfo

Generate a log message at INFO level. All scripting logs are recorded in the "Scripting" log category. Messages can be allocated to a subcategory of "Scripting" using the optional category argument.

For example `bem.logInfo('testCategory', 'test message')` writes `test message` to the logger at INFO level, with the category `Scripting.testCategory`.

Valid Signatures

void bem.logInfo(String *message_text*)	Generate a log message in the default category.
void bem.logInfo(String *category_name*,String *message_text*)	Generate a log message in the specified category.

Returns

None

bem.logWarning

Generate a log message at WARN level. All scripting logs are recorded in the "Scripting" log category. Messages can be allocated to a subcategory of "Scripting" using the optional category argument.

For example `bem.logWarning('testCategory', 'test message')` writes `test message` to the logger at WARN level, with the category `Scripting.testCategory`.

Valid Signatures

void bem.logWarning(String *message_text*)	Generate a log message in the default category.
void bem.logWarning(String *category_name*,String *message_text*)	Generate a log message in the specified category.

Returns

None

bem.nodeId

The bem.nodeId property is read-only and identifies the local monitor to the scripting layer in the format, *`monitor_name`*`:`*`monitor_listener_port`*

For example, `qasun2.mosol.com:6789`

A.1.2 The “metrics” object

The “metrics” object provides access to various services specific to the High-Volume-Metrics application. This object is predefined and always available in the HVMetrics.script context.

- “metrics.constants” on page 138
- “metrics.createFieldFilter()” on page 140

metrics.constants

metrics.constants is a collection of global read-only values used by HVM scripts when interacting with various parts of the API. The collection as a whole is normally not referenced. Members of this collection fall into the categories described in the following sections:

metrics.constants Case Rules

Case rule constants are assigned to the caseRule property of a FieldXform. These rules are typically used for data normalization, especially for data collector grouping fields. For example,

```
var xform = metrics.createXform('Normalize URL');
xform.fieldName=metrics.constants.FN_URL;
xform.caseRule=metrics.constants.CASE_TO_LOWER;
```

Values

`metrics.constants.CASE_TO_LOWER`	Convert field content to lower case.
`metrics.constants.CASE_TO_UPPER`	Convert field content to upper case.

metrics.constants Field Names

Field names are used to identify standard fields in an HVEvent object. They are used in a variety of objects, including FieldFilters, FieldXForms, and DataCollectors to specify the event fields on which they operate. For example,

```
var ipFilter = metrics.createFieldFilter('ipFilter',
               metrics.constants.FN_CLIENT_IP, '192.168.1.*');
var appsCollector = metrics.createSummaryCollector();
appsCollector.exclusionFilters=[ipFilter]; //exclude the local subnet
```

Values

`metrics.constants.FN_APPLICATION_NAME`	Identifies the field containing the application name value sent by the instrumentation script that sent the event.
`metrics.constants.FN_APPLICATION_NAME`	Identifies the field containing the application name value sent by the instrumentation script that sent the event.
`metrics.constants.FN_CLIENT_IP`	Identifies the field containing the IP address of the system that delivered the event to the monitor.
`metrics.contstants.FN_DOWN_LOAD_TIME`	Identifies the field containing the time between page initialization and load complete.
`metrics.constants.FN_EVENT_TS`	Identifies the field containing the time the event was received by the monitor.
`metrics.constants.FN_PAGE_TITLE`	Identifies the field containing the page title of the page that generated the event.
`metrics.constants.FN_RESPONSE_TIME`	Identifies the field containing the response time for this event.
`metrics.constants.FN_UPLOAD_TIME`	Identifies the field containing the time before-unload and page initialization for the event.
`metrics.constants.FN_URL`	Identifies the field containing the URL of the page that generated the event.
`metrics.constants.FN_USER_AGENT`	Identifies the field containing the navigator.userAgent value as reported by the browser that reported this event.

metrics.constants Severities

Severity constants are used when specifying the condition of an alarm or element. For example,

```
var alarm = metrics.createTecAlarm();
alarm.severity= metrics.constants.SEV_CRITICAL;
var elem = metrics.createMetricsElement() ;
elem.condition=metrics.constants.SEV_MINOR;
```

Values

`metrics.constants.SEV_CRITICAL`

`metrics.constants.SEV_INFO`

`metrics.constants.SEV_MAJOR`

`metrics.constants.SEV_MINOR`

`metrics.constants.SEV_OK`

```
metrics.constants.SEV_UNKNOWN
```

metrics.constants Stale-Report Handling Options

Stale-report handling option constants are used to specify how to process stale report data from distributed data collectors during aggregation. These values are assigned to a data collector's staleReportOption property in a distributed collection environment. Refer to SummaryDataCollector.staleReportOption or SamplingDataCollector.staleReportOption for more information.

```
var appsCollector = metrics.createSummaryCollector();
appsCollector.staleReportOption= metrics.constants.SRO_PASS;
```

Values

`metrics.constants.SRO_FILTER`	Specifies that stale reports should be filtered out of aggregate reports.
`metrics.constants.SRO_INCLUDE`	Specifies that stale reports should be rolled into the current interval of aggregate reports.
`metrics.constants.SRO_PASS`	Specifies that stale reports should be passed through to the report handler for processing.

metrics.createFieldFilter()

Creates a new FieldFilter object. FieldFilter objects are included in Preprocessor objects to exclude an HVEvent from processing. They are also used with DataCollectors to determine what HVEvents should be reported on. For example,

```
var ipFilter = metrics.createFieldFilter('ipFilter',
               metrics.constants.FN_CLIENT_IP, '192.168.1.*');

var appsCollector = metrics.createSummaryCollector();
appsCollector.exclusionFilters=[ipFilter]; //exclude the local subnet
```

Valid Signatures

```
FieldFilter metrics.createFieldFilter()

FieldFilter metrics.createFieldFilter(String filterName, String fieldName, String
expression)
```

Returns

A FieldFilter object.

metrics.createFieldXform()

Creates a new FieldXform object. FieldXform objects are included in Preprocessor objects to normalize data in HVEvent objects. For more information and an example, see Section A.2.2, "FieldXform," on page 155.

Valid Signatures

```
FieldXform metrics.createFieldXform()

FieldXform metrics.createFieldXform(String name)

FieldXform metrics.createFieldXform (String name, String fieldName, String
matchExression, String replaceExpression)

FieldXform metrics.createFieldXform (String name, String fieldName, String
matchExression, String replaceExpression, String destFieldName, CaseRule rule)
```

Returns

A FieldXform object.

metrics.createMetricsElement()

Creates a new MetricsElement object. These objects are used to create and update *Operations Center* elements that capture data about your metrics data.

The createMetricsElement method supports two specialized forms, one that takes a SummaryReport, and the other a SamplingReport. These prepopulate MetricsElement.elementKeys with the collector name, followed by the grouping values. If the mapValues argument is set to True, this method also populates the seriesData arrays. In the case of SummaryReports, each of the SummaryStats value in SummaryReport.overall, as well as any populated SummaryReport.rangeStats values.

The example for MetricsElement shows this mapping done in JavaScript. For SummaryReports, each of the captured hits produces a SeriesData entry, with any duplicate time stamps being scrubbed out. For an example, see Section A.2.5, “MetricsElement,” on page 159.

Valid Signatures

```
MetricsElement metrics.createMetricsElement()

MetricsElement metrics.createMetricsElement(String elementKeys)

MetricsElement metrics.createMetricsElement(SummaryReport report, Boolean
mapValues)

MetricsElement metrics.createMetricsElement(SamplingReport report, Boolean
mapValues)
```

Returns

A MetricsElement object.

metrics.createPreprocessor()

Creates a new Preprocessor object. Preprocessor objects are responsible for normalizing and filtering HVEvent objects before they are processed by data collectors. For more information, see Section A.2.7, “Preprocessor,” on page 163.

Valid Signatures

```
Preprocessor metrics.createPreprocessor()

Preprocessor metrics.createPreprocessor(integer uploadTimeLimit)
```

```
Preprocessor metrics.createPreprocessor(integer uploadTimeLimit, FieldXform[]
xforms, FieldFilters[] filters)
```

Returns

A Preprocessor object.

metrics.createResponseTimeRange()

Creates a new ResponsetimeRange object. These objects are used to refine data collectors to track hits where the response time falls within the specified range values. For an example, see Section A.2.9, "ResponseTimeRange," on page 165.

Valid Signatures

```
ResponsetimeRange metrics.createResponsetimeRange()
```

```
ResponsetimeRange metrics.createResponsetimeRange(String name, integer lowTime,
integer highTime)
```

Returns

A ResponsetimeRange object.

metrics.createSeriesData()

Creates a new SeriesData object. These objects are used to track statistical information over time. For more information and an example, see Section A.2.10, "SeriesData," on page 165.

Valid Signatures

```
SeriesData metrics.createSeriesData()
```

```
SeriesData metrics.createSeriesData(String name, java.util.Date, Number value)
```

```
SeriesData metrics.createSeriesData(String name, java.util.Date, Number value,
integer expiryDays)
```

Returns

A SeriesData object.

metrics.createSummaryCollector()

Creates a new SummaryDataCollector object. These objects are used to analyze the HVEvent stream and periodically produce summary reports describing the content of the event stream. The various configuration options on the data collector determine precisely what information to extract and the frequency it is reported. For more information and an example, see Section A.2.11, "SummaryDataCollector," on page 167.

Valid Signatures

```
SummaryDataCollector metrics.createSummaryCollector()

SummaryDataCollector metrics.createSummaryCollector(String name, integer
intervalDuration, Function reportHandler)

SummaryDataCollector metrics.createSummaryCollector(String name, integer
intervalDuration, Function reportHandler, String[] groupingFieldNames,
FieldFilter[] inclusionFilters, FieldFilter[] exclusionFilters,
ResponseTimeRange[] ranges, String staleReportOption)
```

Returns

A SummaryDataCollector object.

metrics.createTecAlarm()

Creates a new MetricAlarm object for delivery to a TEC subscriber. The content of these alarms is largely driven be the TEC receiver. There are two convenience versions that automatically mines relevant information from the provided report object as follows:

```
metricAlarm.timestamp = report.stopTime;
metricAlarm.id = report.key.hashCode();
metricAlarm.collectorName = xiReport.collector.name;
```

In addition, the appName property is populated from an HVEvent in the event stream.

For more information and an example, see Section A.2.4, “MetricAlarm,” on page 158.

Valid Signatures

```
MetricAlarm metrics.creatTecAlarm()

MetricAlarm metrics.creatTecAlarm(SamplingReport report, String severity, String
description)

MetricAlarm metrics.creatTecAlarm(SummaryReport report, String severity, String
description)
```

Returns

A MetricAlarm object.

metrics.getAllMembers()

Retrieve the list of members defined in the current group. The list includes the local instance. All return data is read-only.

```
var members = metrics.getAllMembers();
for(var i=0; i < members.length; i++)
{
   var elem = metrics.createMetricsElement();
   elem.elementKeys = ['groupStatus', members[i].id];
   elem.properties.put('Is Controller', !members[i].isRemote);
   elem.properties.put('Is Connected', members[i].isConnected());
   elem.condition=metrics.constants.SEV_MINOR;
   elem.isPropertySetComplete=false;
   metrics.sendMetricsElement(elem);
}
```

Valid Signatures

```
MetricsMember[] metrics.getAllMembers()
```

Returns

The list of members defined in the current group as MetricsMembers objects.

```
var statusElem = metrics.createMetricsElement();
statusElem.elementKeys = ['groupStatus'];
statusElem.properties.put('currentController', metrics.getLocalMember().id);
statusElem.condition=metrics.constants.SEV_OK;
statusElem.isPropertySetComplete=false;
metrics.sendMetricsElement(statusElem);
```

metrics.getLocalMember()

Retrieve information about the local metrics group member. All return data is read-only.

Valid Signatures

```
MetricsMember metrics.getLocalMembers()
```

Returns

The MetricsMembers object representing the local Monitor.

metrics.onload

The metrics.onload property is used to specify the event handler to be called when HVM is starting up. The specified handler function is responsible for deploying the HVM configuration. In a single-monitor environment, this handler is normally only called during monitor startup. In a grouped-monitor environment, it can be called several times in the life of a monitor instance as the controller changes.

```
metrics.onload=doStart;

function doStart()
{
   var preproc metrics.createPreprocessor(
                    metrics.properties.get('Cust.uploadThreshold',2500));
   var apps = metrics.createSummaryCollector('Applications',60000,handleReports);
   apps.groupingFieldNames=[metrics.constants.FN_APPLICATION_NAME];

   metrics.start(preproc, apps);
}
function handleReports(reports)
{
   // process reports here
}
```

Handler Signature

```
void onload()
```

metrics.onMemberConnect

The metrics.onMemberConnect property is used to specify the event handler to be called when a connectivity is established with another member in the group. The handler function can be used to track report status of an HVM group configuration. In a single monitor environment, this handler is not called. In a group monitor environment, it is called when the initial connection is established and each time it is re-established after a connectivity failure.

```
metrics.onMemberConnect=handleConnect;

function handleConnect(memberData)
{
   var statusElem = metrics.createMetricsElement();
   statusElem.elementKeys = ['groupStatus', memberData.id];
   statusElem.condition=metrics.constants.SEV_OK;
   statusElem.message='Member is active';
   statusElem.isPropertySetComplete=false;
   metrics.sendMetricsElement(statusElem);
}
```

Handler Signature

```
void onMemberConnect(GroupMember memberData)
```

metrics.onMemberDisconnect

The metrics. onMemberDisconnect property is used to specify the event handler to be called when connectivity to another monitor in the group is lost. The handler function can be used to track report status of an HVM group configuration. In a single monitor environment, this handler is not called. In a group monitor environment, it is called when the connectivity is lost and cannot be immediately re-established.

```
metrics.onMemberDisconnect=handleDisconnect;

function handleDisconnect(memberData)
{
   var statusElem = metrics.createMetricsElement();
   statusElem.elementKeys = ['groupStatus', memberData.id];
   statusElem.condition=metrics.constants.SEV_UNKNOWN;
   statusElem.message='Member unreachable';
   statusElem.isPropertySetComplete=false;
   metrics.sendMetricsElement(statusElem);
}
```

Handler Signature

```
void onMemberDisconnect(GroupMember memberData)
```

metrics.onMemberRefused

The metrics.onMemberRefused property is used to specify the event handler to be called when another member in the group refused control by the local instance. The handler function can be used to track report status of an HVM group configuration. In a single monitor environment, this handler is not called. In a group monitor environment, it is called when a member refuses to be controlled by this instance. This indicates that the remote instance either does not recognize the local instance as valid controller, or it is already controlled by a higher precedence controller.

```
metrics.onMemberRefused=handleRefused;

function handleRefused(memberData)
{
   var statusElem = metrics.createMetricsElement();
   statusElem.elementKeys = ['groupStatus', memberData.id];
   statusElem.condition=metrics.constants.SEV_CRITICAL;
   statusElem.message='Member denied connection. Check configuration';
   statusElem.isPropertySetComplete=false;
   metrics.sendMetricsElement(statusElem);
   // might want to send an alarm here too...
}
```

Handler Signature

```
void onMemberRefused(GroupMember memberData)
```

metrics.onunload

The metrics.onunload property is used to specify the event handler to be called when HVM is shutting down. The handler function can be used to track report status of an HVM configuration. In a single monitor environment, this handler is only called when the monitor is shutting down. In a grouped-monitor environment, it can be called several times in the life of a monitor instance as the controller changes.

```
metrics.onunload=stopping;

function stopping()
{
   var statusElem = metrics.createMetricsElement();
   statusElem.elementKeys = ['groupStatus',
                             metrics.properties.getMyID().toString()];
   statusElem.condition=metrics.constants.SEV_UNKNOWN;
   statusElem.message='Discontinued group control';
   statusElem.isPropertySetComplete=false;
   metrics.sendMetricsElement(statusElem);
}
```

Handler Signature:

```
void onunload()
```

metrics.properties

The metrics.properties object provides read access to values set in the metrics properties files. The current metrics properties file is specified in the "metrics.properties" property of the `monitor.properties` file. While this file does contain certain properties that are interpreted by the monitor (such as HVMetrics.script), the primary purpose of the metrics properties file is to configure the scripts that drive the HVM implementation. As such, the content of a particular metrics properties file is specific to the implementation. The examples used here are based on the `DemoMetrics.properies` file that ships with the product.

metrics.properties.get()

Retrieve the value of a specified property. The various forms of the get method allows the script to quickly access a property value in the same form as the provided default value. When the value is retrieved from the properties file, it is coerced to the target type in the java layer, not the JavaScript layer. This means that if your properties file contains "myProp.count=14", then metrics.properties.get("myProp.count", 5) returns 14, but if "myProp.count=", then 5 is returned. (A JavaScript interpretation of "myProp.count=" returns 0.)

```
var elementName=metrics.properties.get('Cust.topElementName','demo');
var uploadThreshod=metrics.properties.get('Cust.uploadThreshold',2000);
var errorPercent=metrics.properties.get('Cust.errorPercentThreshold', 0.10);
var sendDisconnectAlerts=metrics.properties.get(
   'Cust.group.sendMonitorConnAlerts',true);
```

Valid Signatures

```
String metrics.properties.get(String propName, String defaultValue)

Integer metrics.properties.get(String propName, integer defaultValue)

Float metrics.properties.get(String propName, float defaultValue)

Boolean metrics.properties.get(String propName, Boolean defaultValue)
```

Returns

The value of the specified property from the current metrics properties file. The returned type always matches the type of the provided defaultValue argument. If the specified property is not defined or cannot be interpreted as the specified type, the default value is returned.

metrics.sendMetricAlarm()

Send MetricAlarm objects to designated recipients. The recipients of the alarm are designated at alarm creation. Currently only TEC recipients are support using the metrics.createTecAlarm() functions. If any included elements are invalid (MetricAlarm.isValid()), then an exception is thrown. Otherwise, alarms are delivered to the publishing component.

Errors in delivery are handled like other event data sent to the recipients (retries, caching, and so on). No notification is sent back to the script.

For more information and an example, see Section A.2.4, “MetricAlarm,” on page 158.

Valid Signatures

```
metrics.sendMetricAlarm (MetricAlarm alarm)

metrics. sendMetricAlarm (MetricAlarm[] alarms)
```

Returns

N/A

metrics.sendMetricsElement()

Send MetricElement objects to Operations Center. If any included elements are invalid (MetricsElement.isValid()), then an exception is thrown. Otherwise, elements are delivered to the publishing component. Errors in delivery are handled like other event data sent to Operations Center (retries, caching, and so on). No notification is sent back to the script.

For an example, see Section A.2.5, “MetricsElement,” on page 159.

Valid Signatures

```
metrics.sendMetricsElement (MetricsElement element)

metrics.sendMetricsElement (MetricsElement[] elements)
```

Returns

N/A

metrics.start()

Deploys a new HVM configuration and initiates collection. A successful start requires a valid Preprocessor object and a list of valid SummmaryCollector and/or SamplingCollector objects. At least one collector must be in the list. If any arguments are invalid, this method throws a ScriptingException. Otherwise, the objects are deployed to the local system and, if necessary, the Web server is started (`If Monitor.properties/Metrics.manageWebServer=true`). If in a distributed configuration, the monitor also connects to the group members and starts them, as well. (Remote deployment occurs on a separate thread of execution so distribution might not have been completed when this method returns. You can track progress using the metrics.onMemberConnected and related event handlers.)

```
metrics.onload=doStart;

function doStart()
{
   var preproc =
metrics.createPreprocessor(metrics.properties.get('Cust.uploadThreshold',2500));
   var apps = metrics.createSummaryCollector('Applications',60000,handleReports);
   apps.groupingFieldNames=[metrics.constants.FN_APPLICATION_NAME];
   var pages = metrics.createSummaryCollector('Pages',60000,handleReports);
   pages.groupingFieldNames=[metrics.constants.FN_PAGE_TITLE];

   metrics.start(preproc, [apps,pages]);
}
function handleReports(reports)
{
   // process reports here
}
```

Valid Signatures

```
void metrics.start(Preprocessor preprocessor, DataCollector collector)

void metrics.start(Preprocessor preprocessor, DataCollector[] collectors)
```

A.1.3 The “testEngine” object

The “testEngine” object provides limited access to the Experience Manager Monitor’s synthetic test engine, allowing scripts to track changes to the deployment, and start and stop tests and/or scenarios. This object is predefined and always available in all scripting contexts.

testEngine.activateScenario()

Activates a test scenario in the Experience Manager Monitor test engine. The test scenario must be part of the SyntheticDeployment and must currently be deactivated. This method is functionally equivalent to the "Activate Scenario" operation available under *Monitor Administration* in the *Operations Center* client.

```
/*
 * Simple web-invocable operation to activate a specified test scenario.
 * Activate the first scenario in test 'foo',with:
 * http[s]://<BemWebServer>/Plugins/
invoke.pl?testName=foo&seq=1&op=activateScenario
 */
function activateScenario(parmMap)
{
   var testName=parmMap.get('testName');
   var seqNum=parmMap.get('seq');
   return testEngine.activateScenario(testName,seqNum);
}
```

This operation has no impact on the status of the SyntheticTest object.

Valid Signatures

```
String testEngine.activateScenario(String testName, Number sequenceNumber)
```

Returns

OK if the operation is successful; otherwise, a string describing the error.

testEngine.activateTest()

Activates a test in the Experience Manager Monitor test engine. The test must be part of the SyntheticDeployment and must currently be deactivated. If successful, the test is scheduled to run immediately and continues to run on its scheduled interval until deactivated or removed. This method is functionally equivalent to the "Activate Test" operation available under *Monitor Administration* in the *Operations Center* console.

```
/*
 * Simple web-invocable operation to activate a specified test.
 * Activate test 'foo' with:
 * http[s]://<BemWebServer>/Plugins/invoke.pl?testName=foo&op=activateTest
 */
function activateTest(parmMap)
{
   var testName=parmMap.get('testName');
   return testEngine.activateTest(testName);
}
```

Valid Signatures

```
String testEngine.activateTest(String testName)
```

Returns

OK if the operation is successful; otherwise, a string describing the error.

testEngine.deactivateScenario()

Deactivates a test scenario in the Experience Manager Monitor test engine. The test scenario must be part of the SyntheticDeployment and must currently be activated. This method is functionally equivalent to the "Deactivate Scenario" operation available under *Monitor Administration* in the *Operations Center* console.

```
/*
 * Simple web-invocable operation to deactivate a specified test scenario.
 * Deactivate the first scenario in test 'foo',with:
 * http[s]://<BemWebServer>/Plugins/
invoke.pl?testName=foo&seq=1&op=deactivateScenario
 */
function deactivateScenario(parmMap)
{
   var testName=parmMap.get('testName');
   var seqNum=parmMap.get('seq');
   return testEngine.deactivateScenario(testName,seqNum);
}
```

This operation has no impact on the status of the SyntheticTest object.

Valid Signatures

```
String testEngine.deactivateScenario(String testName, Number sequenceNumber)
```

Returns

OK if the operation is successful; otherwise, a string describing the error.

testEngine.deactivateTest()

Deactivates a test in the Experience Manager Monitor test engine. The test must be part of the SyntheticDeployment and must currently be activated. If successful, the test is no longer scheduled to run. If it is currently running, it is allowed to complete, but is not rescheduled. This method is functionally equivalent to the "Deactivate Test" operation available under *Monitor Administration* in the *Operations Center* console.

```
/*
 * Simple web-invocable operation to deactivate a specified test.
 * Deactivate test 'foo' with:
 * http[s]://<BemWebServer>/Plugins/invoke.pl?testName=foo&op=deactivateTest
 */
function deactivateTest(parmMap)
{
   var testName=parmMap.get('testName');
   return testEngine.deactivateTest(testName);
}
```

Valid Signatures

```
String testEngine.deactivateTest(String testName)
```

Returns

OK if the operation is successful; otherwise, a string describing the error.

testEngine.constants

testEngine.constants is a collection of global read-only values used by Test Management and WebOps scripts. The collection as a whole is normally not referenced. Members of this collection fall into the categories described in the following sections:

testEngine.constants Scenario Types

Scenario type constants are used to distinguish different types of SyntheticScenario objects. The provided values correspond to the types of scenarios that can be created in Experience Manager adapter test administration.

Values

`testEngine.constants.SCENARIO_TYPE_HTTP`

`testEngine.constants.SCENARIO_TYPE_HTTPS`

`testEngine.constants.SCENARIO_TYPE_SQL`

`testEngine.constants.SCENARIO_TYPE_THIRD_PARTY`

`testEngine.constants.SCENARIO_TYPE_NATIVE_WINDOWS`

`testEngine.constants.SCENARIO_TYPE_MAIL`

`testEngine.constants.SCENARIO_TYPE_FTP`

`testEngine.constants.SCENARIO_TYPE_PING`

`testEngine.constants.SCENARIO_TYPE_TRACERT`

`testEngine.constants.SCENARIO_TYPE_DHCP`

`testEngine.constants.SCENARIO_TYPE_LDAP`

testEngine.constants Severities

Test engine severity constants are used to identify the severity of a SyntheticAlarm produced by the testEngine.

For instructions, see Section A.2.18, “SyntheticTestResults,” on page 177.

Values

`testEngine.constants.SEV_CRITICAL`

`testEngine.constants.SEV_INFO`

`testEngine.constants.SEV_MAJOR`

`testEngine.constants.SEV_MINOR`

`testEngine.constants.SEV_OK`

`testEngine.constants.SEV_UNKNOWN`

testEngine.getDeployment()

Retrieves the current deployment of synthetic tests. The returned object is a snapshot of the monitor's deployment. It does not change if the deployment changes.

For instructions, see Section A.2.15, "SyntheticDeployment," on page 172.

Valid Signatures

```
SyntheticDeployment testEngine.getDeployment()
```

Returns

A SyntheticDeployment object representing the current monitor deployment.

testEngine.onDeploymentChanged

The testEngine.onDeploymentChanged property is used to specify the event handler to be called when the tests deployed to the test engine change in any way. The handler function can be used to reinitialize rules that handle fail-over conditions, audit changes to deployment, and so on.

The following example is really just a skeleton to show how to capture this event. Implementations of fail-over tracking and other synthetic test management functions can be fairly complex and distracts from the purpose of this example.

```
testEngine.onDeploymentChanged = handleNewDeployment;

function handleNewDeployment(originator,deployment)
{
   // filter out changes that originate locally...
   if(!(originator == bem.nodeId))
   {
      bem.logDebug('Processing deployment change from: ' + originator);
      var tests = deployment.getTests();
      // ... update rules/states based on changes in deployment
   }
}
```

Handler Signature

```
void onDeploymentChanged(String originator, SyntheticDeployment deployment)
```

testEngine.onScenarioComplete

The testEngine.onScenarioComplete property is used to specify the event handler to be called when the test engine completes execution of a test scenario. The handler function is typically used as part of a Synthetic Test Management implementation involving some form of conditional execution logic.

For an examples, see Section A.2.18, "SyntheticTestResults," on page 177.

Handler Signature

```
void onScenarioComplete(SyntheticAlarm alarm)
```

testEngine.onTestComplete

The testEngine.onTestComplete property is used to specify the event handler to be called when the test engine completes execution of a test. The handler function is typically used as part of a Synthetic Test Management implementation involving some form of conditional execution logic.

For an example, see Section A.2.18, "SyntheticTestResults," on page 177.

Handler Signature

```
void onTestComplete(SyntheticTestResults testResults)
```

testEngine.properties

The testEngine.properties object provides read access to values set in the test management properties files. The current test management properties file is specified in the tests.management.properties property of the `monitor.properties` file. While this file does contain certain properties that are interpreted by the monitor (for example, tests.management.script), the primary purpose of the properties file is to configure test management scripts. As such, the content of a particular properties file is specific to the implementation.

testEngine.properties.get()

Retrieve the value of a specified property. The various forms of the get method allows the script to quickly access a property value in the same form as the provided default value. When the value is retrieved from the properties file, it is coerced to the target type in the java layer, not the JavaScript layer. This means that if your properties file contains `"myProp.count=14"`, then `metrics.properties.get("myProp.count", 5)` returns 14, but if `"myProp.count="`, then 5 is returned. (A JavaScript interpretation of `"myProp.count="` returns 0.)

Valid Signatures

```
String testEngine.properties.get(String property_name, String property_value)

Integer testEngine.properties.get(String property_name, integer property_value)

Float testEngine.properties.get(String property_name, Float property_value

Boolean testEngine.properties.get(String property_name, Boolean property_value
```

Returns

The value of the specified property from the current properties file. The returned type always matches the type of the provided defaultValue argument. If the specified property is not defined or cannot be interpreted as the specified type, the default value is returned.

A.2 Referenced Types

Types described in this section form the substance of the bridge between the internal Experience Manager-Monitor environment and the scripts used to control various aspects of monitor behavior. Each of these types is either created buy a call to a build-in object, or passed to an event handler defined in script.

- Section A.2.1, "FieldFilter," on page 154
- Section A.2.2, "FieldXform," on page 155

A.2.1 FieldFilter

A FieldFilter object is used to include or exclude an HVEvent based on the value of one of its fields. All fields in this object are required. In addition, the *regex* field must contain a valid regular expression as described here:

http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html. (http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html)

The fieldName value is normally one of standard field names described in metrics.constants Field Names. However, custom field names created with a FieldXform can be used. HVEvents not containing a value in the field specified by fieldName never match the filter.

```
var appsCollector = metrics.createSummaryCollector();

var testFilter = metrics.createFieldFilter();
testFilter.name = 'testFilter';
testFilter.fieldName = metrics.constants.FN_APPLICATION_NAME;
testFilter.regex = 'TEST.*';
if(testFilter.isValid())
{
   appsCollector.exclusionFilters=[testFilter]; //exclude test events
}
else
{
  bem.logWarning('Test filter not valid:' + testFilter.toString());
}
```

Properties

Property Name	Value Type	Description
`name`	`String`	(Required) Name of the filter.
`fieldName`	`String`	(Required) Name of the field the filter operates on.

Methods

`Boolean isValid()`	Returns True if all required fields are populated and the expression fields contains a valid regular expressions.If False is returned, the scripting log contains a description of the failure.
`String toString()`	Returns a string representation of this event.

A.2.2 FieldXform

A FieldXform object is used to normalize data values in an HVEvent object. The name, fieldName, matchExpression, and replaceExpression fields in this object are required. In addition, the matchExpression field must contain a valid regular expressions as described here:

http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html (http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html).

The replaceExpression can contain both literal data, as well as references to captured substrings from the matchExpression. Capture references are referenced using a dollar sign ($) followed by the index of the desired capture field, starting with index 1 to indicate the first capture field. A literal dollar sign must be preceded with a backslash to indicate that it is not a capture reference. The fieldName value is normally one of standard field names described in metrics.constants Field Names. However, custom field names created with a previous FieldXform can be used. HVEvents not containing a value in the field specified by fieldName never match.

When an HVEvent is applied to a FieldXForm, the FieldXForm compares the value of the specified field with its matchExpression. If the value does not match the expression, then no changes are made to the HVEvent. The the value does match, then the replaceExpression is used to transform the value, and if specified, the caseRule is used to normalize the case of the transformed value. The resulting value is then save to the destination field, or if not specified, back to the field from which the source value was taken.

The following example for FieldXform is from the `DemoMetrics.js` script included in the Experience Manager installation. It is designed to identify the same page across multiple hosts, while suppressing any parameters passed to the page.

```
var ulx= metrics.createXform('URLKey Xform');
ulx.fieldName=metrics.constants.FN_URL;
ulx.matchExpression='http[s]?://[^/]+([^#\?]+).*';
ulx.replaceExpression='$1';
ulx.destFieldName='urlKey';
ulx.caseRule=metrics.constants.CASE_TO_LOWER;
```

It evaluates the URL field in the HVEvent against the match expression:

http[s]?://[^/]+([^#\?]+).*

Table A-1 lists the regular expressions, and describes how the configured matchExpression is interpreted.

Table A-1 *matchExpression Interpretations*

Match Characters	Interpretation
http	Matches only the literal string `http`.
[s]?	Matches 0 or one “s” in this position.
://	Matches only the literal string “://” in this position.
[^/]+	Matches one or more of any character up to but not including the / character.
([^#\?]+)	Matches one or more of any character up to but not including # or ? characters. The parentheses mark this section as a capture field that is referenced in replaceExpression.
.*	Matches any number of trailing characters of any type.

Based on this match expression, the xform saves the value of the first capture field ($1), converts it to lower case, then saves it to a custom HVEvent field called `urlKey`. Table A-2 shows sample URL values and the resulting `urlKey` values produced by this xform:

Table A-2 *Sample URL and resulting urlKey values for the xform of the matchExpression example*

URL	UrlKey
http://www.netiq.com/products/experience-manager/	/products/experience-manager/
http://localhost:8080/Plugins/invoke.pl?op=threadDump	/plugins/invoke.pl
http://us.mc1102.mail.yahoo.com/mc/welcome?.gx=1&.tm=1250689315&.rand=3hsbmdsrkkmg6	/mc/welcome

This approach obviously does not work for all Web sites. You should come up with your own schemes for normalizing data into useful values.

- “Properties” on page 156
- “Methods” on page 157

Properties

Property Name	Value Type	Description
`name`	`String`	(Required) Name of the Xform.
`fieldName`	`String`	(Required) Name of the field to be read by the Xform.
`matchExpression`	`String`	(Required) Regular expression describing field values that should be transformed.
`replaceExpression`	`String`	(Required) A regular expression describing how to transform values that match the matchExpression.

Property Name	Value Type	Description
destFieldName	String	Name of the field where transformed data should be written.
caseRule	Object	(Rule for changing the case of matching fields.
generalProperties	java.util.Map	Other properties for the alarm.

Methods

Boolean isValid()	Returns True if all required fields are populated and the expression fields contains a valid regular expressions.If False is returned, the scripting log contains a description of the failure.
String toString()	Returns a string representation of this event.

A.2.3 HVEvent

An HVEvent object represents a single hit by a user on a single Web page. All properties in this object are read-only. Fields in the event have undergone any transformations implemented in the preprocessor. The get() method can be used to access any custom fields produced by xforms.

Under normal circumstances, the vast majority of events received by a monitor are discarded after processing in order to conserve system resources. However, sampling data collectors capture and report a percentage of these events as HVEvent objects. Summary reports also include a reference to a single HVEvent object, provided for reference purposes.

```
// Calculate the median response time from sampled events
function calculateMedian(events)
{
    var median =0;
    if(events.length > 1)
    {
       var values = new Array();
       for(var i=0; i< events.length; i++)
       {
          values.push(events[i].responseTime);
       }
       values.sort(sortOrder);
       var mid = Math.floor(values.length/2);
       if((values.length % 2) == 0)
       {
         median = (values[mid-1]+values[mid])/2;
       }
       else
       {
```

```
            median = values[mid];
         }
      }
      else if(events.length == 1)
      {
         median = events[0].responseTime;
      }
      return median;
}
// ensures sorting by numeric order instead of the default alpha sort
function sortOrder(a,b)
{
   return a-b;
}
```


- "Properties" on page 158
- "Methods" on page 158

Properties

Property Name	Value Type	Description
appName	String	The appName parameter from the instrumentation script that generated this event.
clientIP	String	The IP address of the system that delivered this event to the Monitor.
downloadTime	integer	The reported interval between initialization of the current page and final rendering (in milliseconds).
timestamp	java.util.Date	The time when this event was received by the Monitor.
responseTime	integer	The downloadTime+uploadTime, or downloadTime if uploadTime is out of range. This is the value used in SummaryStats calculations.
title	String	The document title of the page that produced this event.
uploadTime	String	The reported interval between unload of previous page and current page initialization.
URL	String	The URL of the page that produced this event.
userAgent	String	The userAgent value reported by the browser that sent this event.

Methods

String get(String fieldName)	Retrieves the value of a named field.
String toString()	Returns a string representation of this event.

A.2.4 MetricAlarm

A MetricAlarm object represents an alert to be sent to a remote system. This is a general purpose mechanism and the necessary content of the alarm is based on the needs of the system receiving the alarm. (HVM does not currently support Operations Center alarms.)

```
metrics.onMemberRefused=handleRefused;

function handleRefused(memberData)
{
   var alarm = metrics.createTecAlarm();
   alarm.id= memberData.id.hashCode();
   alarm.appName='HVM Group Status';
   alarm.timestamp = new java.util.Date();
   alarm.severity = metrics.constants.SEV_CRITICAL;
   alarm.description= memberData.id + ' refused control by ' +
metrics.getLocalMember().id;
   metrics.sendAlarm(alarm);
}
```


- "Properties" on page 159
- "Methods" on page 159

Properties

Property Name	Value Type	Description
`appName`	`String`	(Required) Name of the application to which this alert applies.
`collectorName`	`String`	Name of the collector this alert is associated with.
`timestamp`	`java.util.Date`	(Required) Time when this alert occurred.
`severity`	`String`	(Required) Severity of the alarm (see mtrics.constants Severities).
`description`	`String`	Description of the alarm.
`id`	`String`	(Required) Identifier for this alert.
`generalProperties`	`java.util.Map`	Other properties for the alarm.

Methods

`Boolean isValid()`	Returns True if object contains all required data and all provided data is valid. If False is returned, the logs include details of the error.
`String toString()`	Returns a string representation of this event.

A.2.5 MetricsElement

A MetricsElement object represents an element on the Operations Center server under the *Elements/Experience Manager_adapter/Metrics/* branch. When a MetricsElement object is received by the Experience Manager adapter on the Operations Center server, the MetricsElement data is used to create or update the Operations Center Element. The elementKeys property specifies the relative path from *Elements/Experience Manager_adapter/metrics/*. So, if you specify `elementKeys ['Page Stats', 'Home Page']`, then the MetricsElement object creates or updates *Elements/Experience Manager_adapter/metrics/Page Stats/Home Page*.

It is not necessary to create "Page Stats" prior to creating "Home Page"; if a nonexistent element is referenced in the element hierarchy, it is created automatically. The condition and message properties are used to update the corresponding fields on the Operations Center element.

The seriesData property is used to capture metric data that you wish to record over time. When you select the *Performance* tab for the *Operations Center* element you have created, any series data types you have populated should show in the *Performance Properties* panel of the display.

The properties property is used to populate the metrics properties page (right-click *Properties* > *Metrics*) for the element. After a metrics property has been added to an element, it remains there until the element has been removed, or the property is deleted. To delete a property, you can set its value to null (for example, `'elem.properties.put('somePropertyName', null);'`). Alternatively, if you set isPropertySetComplete to True, then any properties not included in the MetricsElement object is removed.

```
 var collector = metrics.createSummaryCollector('Page Stats', 60000,
processPageStats);
 collector.groupingFieldNames=[metrics.constants.FN_PAGE_TITLE];
 var targetRange = metrics.createResponseTimeRange('target', 0, 1500);
 var acceptRange = metrics.createResponseTimeRange('accept', 1500, 4000);
 var problemRange = metrics.createResponseTimeRange('problem', 4000, 60000);

 collector.ranges=[targetRange,acceptRange,problemRange];

 // ... add other collector props and deploy

 /*
  * Process reports generated by the Page Stats collector
  */
 function processPageStats(reports)
 {
    for(var i=0; i < reports.length;i++)
    {
       var report = reports[i];
       // create element and automatically map stats into series data.
       var elem = metrics.createMetricsElement()
       elem.elementKeys=[report.key.collectorName, report.key.groupValues[0]];
       assignConditonAndMessage(elem, report);
       elem.seriesData = createSeriesEntries(report);
       elem.properties.put('Reference URL', report.lastEvent.URL);
       elem.properties.put('Page Title', report.lastEvent.title);
       elem.properties.put('Hit Count', report.overall.count);
       // mine other properties ..
       metrics.sendMetricsElement(elem);
    }
 }


 /*
  * calculate element condition based on relative hit count
  */
 function assignConditonAndMessage(elem,report)
 {
    var targetHits = getRangeHitCount(report,'target');
    var acceptHits = getRangeHitCount(report,'accept');
    var problemHits = getRangeHitCount(report,'problem');

    if(targetHits > acceptHits + problemHits)
    {
       elem.condition=metrics.constants.SEV_OK;
       elem.message = 'Performance target achieved';
    }
    else if (targetHits  > problemHits || acceptHits > problemHits)
    {
       elem.condition = metrics.constants.SEV_MINOR;
       elem.message = 'Performance acceptable';
    }
    else if (targetHits + acceptHits > problemHits)
    {
       elem.condition = metrics.constants.SEV_MAJOR;
       elem.message = 'Performance degraded';
```

```
    }
    else
    {
       elem.condition = metrics.constants.SEV_CRITICAL;
       elem.message = 'Performance violation';
    }
    elem.message += ': T=' + targetHits + ' A='+ acceptHits + ' P=' + problemHits;
}

/*
 * Return hit count if range has data, or 0 if it does not
 */
function getRangeHitCount(report, rangeName)
{
   var hitCount = 0;
   var rangeStats = report.rangeStats.get(rangeName);
   if(rangeStats != null)
   {
      hitCount =  rangeStats.count;
   }
   return hitCount;
}

/*
 * Create Series entries for this report
 */
function createSeriesEntries(report)
{
   var seriesData = new Array();

   var avg=metrics.createSeriesData();
   avg.name = 'Overall:Avg';
   avg.timestamp = report.stopTime;
   avg.value = report.overall.average;
   seriesData.push(avg);

   seriesData.push(metrics.createSeriesData('Overall:Max',
                   report.stopTime,report.overall.high));
   seriesData.push(metrics.createSeriesData('Overall:Min',
                   report.stopTime,report.overall.low));
   seriesData.push(metrics.createSeriesData('Overall:Count',
                   report.stopTime,report.overall.count));

   var iterator = report.rangeStats.entrySet().iterator();
   while(iterator.hasNext())
   {
       var mapEntry = iterator.next();
       var rangeName = mapEntry.getKey();
       var rangeStats = mapEntry.getValue();
       seriesData.push(metrics.createSeriesData(rangeName +':Avg',
                       report.stopTime,rangeStats.average));
       seriesData.push(metrics.createSeriesData(rangeName +':Max',
                       report.stopTime,rangeStats.high));
       seriesData.push(metrics.createSeriesData(rangeName +':Min',
                       report.stopTime,rangeStats.low));
       seriesData.push(metrics.createSeriesData(rangeName +':Count',
                        report.stopTime,rangeStats.count));
   }
   return seriesData;
}
```

- "Properties" on page 162
- "Methods" on page 162

Properties

Property Name	Value Type	Description
`elementKeys`	`String`	(Required) The elementKeys array holds the element's hierarchy relative to *Elements/Experience Manager_adapter/metrics/*.
`condition`	`String`	The condition (OK, MAJOR, CRITICAL, and so on) to assign to this element. If set, the value must one of the constants described in `metrics.constants Severities`.
`message`	`String`	The title bar message for this element that contains a short description.
`seriesData`	`SeriesData[]`	Array of time points to record with this element.
`properties`	`java.util.Map`	Property name/value pairs to be exposed on the element's *Properties* page.
`isPropertySetComplete`	`Boolean`	True/false indicator that impacts how properties are applied to a preexisting element.

Methods

`Boolean isValid()`	Returns True if object contains all required data and all provided data is valid. If False is returned, the logs include details of the error.
`String toString()`	Returns a string representation of this event.

A.2.6 MetricsMember

A MetricsMember object refers to a Experience Manager monitor that is a member of an HVM group. Instances of this object are returned by the metrics.getLocalMember() and metrics.getAllMembers and are passed as arguments to metrics.onMemberConnect, metrics.onMemberDisconnect and metrics.onMemberRefused event-handlers.

```
var members = metrics.getAllMembers();
for(var i=0; i < members.length; i++)
{
   var elem = metrics.createMetricsElement();
   elem.elementKeys = ['groupStatus', members[i].id];
   elem.properties.put('Is Controller', !members[i].isRemote);
   elem.properties.put('Is Connected' , members[i].isConnected());
   elem.condition=metrics.constants.SEV_MINOR;
   elem.isPropertySetComplete=false;
   metrics.sendMetricsElement(elem);
}
```


- "Properties" on page 163
- "Methods" on page 163

Properties

Property Name	Value Type	Description
`id`	`String`	*(Read-only)* MetricsMember.id is a read-only property that uniquely identifies a group member. The format of this identifier matches the format of the member in the HVMetrics.group.members property.
`isRemote`	`Boolean`	*(Read-only)* MetricsMember.isRemote is a read-only property that indicates whether the referenced member is the local system or a remote monitor.

Methods

`Boolean isConnected()`	MetricsMember.isConnected() indicates whether connectivity between the local controller and this member is established.

A.2.7 Preprocessor

Preprocessor objects are responsible for normalizing and filtering HVEvent objects before they are processed by data collectors. A deployed preprocessor is the first to handle event data coming in from the browser. First, it populates the response time field by adding the provided upload time and download time values. However, if the provided upload time is greater than the specified uploadTimeLimit, the upload time value is ignored and the response time will equal the download time.

After the deployed preprocessor populates the response time, it performs data transformations as specified by any included xforms. If multiple xforms are specified, they are applied in the sequence that they appear in the array. Following that, filters are applied. Any event matching a preprocessor filter is excluded from further processing.

```
// filter out local subnet to prevent skew.
 var subnetFilter= metrics.createFieldFilter(
     'ipFilter',
     metrics.constants.FN_CLIENT_IP,
     '192.168.1.*');
 var ieXform = metrics.createXform(
     'IE xform',
     metrics.constants.FN_USER_AGENT,
     '(.*)(MSIE[^;]*)(.*)',
     '$2',
     'browserVersion',
     metrics.constants.CASE_TO_LOWER);
 var ffXform = metrics.createXform(
     'Firefox xform',
     metrics.constants.FN_USER_AGENT,
     '(.*)(Firefox\\S*)(.*)',
     '$2',
     'browserVersion',
     metrics.constants.CASE_TO_LOWER);
 var preproc = metrics.createPreprocessor();
 preproc.uploadTimeLimit=2000;
 preproc.xforms=[ieXform,ffXform];
 preproc.filters=[subnetFilter];
```


- "Properties" on page 164
- "Methods" on page 164

Properties

Property Name	Value Type	Description
`uploadTimeLimit`	`integer`	*(Recommended)* Maximum uploadTime value (ms) that is included in response time calculation.
`FieldXform[]`	`xforms`	Transformers to be applied to the incoming fields. Transformations are performed in the provided order and are done after response time calculation.
`FieldFilter`	`filter`	Filters to describe any events that should be ignored. An event that matches any filter in the preprocessor is ignored.

Methods

`String toString()`	Returns a string description of the preprocessor.
`Boolean isValid()`	Returns True if all included xforms and filters are valid.

A.2.8 ReportKey

A ReportKey object is used to identify a SummaryReport or SamplingReport produced by a data collector. The report key ties the associated report to a particular collector and a particular set of grouping values within that collector. Refer to the associated report type for more information.

- "Properties" on page 164
- "Methods" on page 164

Properties

Property Name	Value Type	Description
`collectorName`	`String`	*(Read-only)* Name of the data collector that produced the report.
`type`	`String`	The type of the data collector (Sampling or Summary).
`groupValues`	`String[]`	Values of the grouping fields that provide the scope of this report.

Methods

`String toString()`	Returns a string representation of this object.

A.2.9 ResponseTimeRange

ResponseTimeRange objects are used to refine data collectors to track hits where the response time falls within the specified range values. Each defined response time range on a dta collector reports a discreet count, average/high/low response time of hits that fell within the range.

The following example demonstrates how one might derive series data from a report. The implementation shown here mirrors the default series data created when default series data mapping is enabled using the metrics.createMetricsElement() method.

```
var targetRange = metrics.createResponseTimeRange('target', 0, 1500);
var acceptRange = metrics.createResponseTimeRange('accept', 1500, 4000);
var problemRange = metrics.createResponseTimeRange('problem', 4000, 60000);

var appsCollector = metrics.createSummaryCollector();
appsCollector.ranges=[targetRange,acceptRange,problemRange];
```


- "Properties" on page 165
- "Methods" on page 165

Properties

Property Name	Value Type	Description
`name`	`String`	*(Required)* Name of the range. This value tracks through to the report.
`lowTime`	`integer`	*(Required)* Minimum response time to include in this ranges report (inclusive).
`highTime`	`integer`	*(Required)* Maximum response time to include in this ranges report (exclusive).

Methods

`String toString()`	Returns a string description of the ResponseTimeRange.
`Boolean isValid()`	Returns True if all values are populated and lowTime <= highTime.

A.2.10 SeriesData

A SeriesData object is used to capture a data point for historical tracking. These objects must be associated MetricsElement to be sent to Operations Center. The Operations Center server does not process updates to the same name and time stamp value on the same element. If the value should be kept for a longer of shorter period than the default retention period set in the Experience Manager Adapter properties use expiryDays to override.

```
function createSeriesEntries(report)
{
   var seriesData = new Array();

   var avg=metrics.createSeriesData();
   avg.name = 'Overall:Avg';
   avg.timestamp = report.stopTime;
   avg.value = report.overall.average;
   seriesData.push(avg);

   seriesData.push(metrics.createSeriesData('Overall:Max',
                   report.stopTime,report.overall.high));
   seriesData.push(metrics.createSeriesData('Overall:Min',
                   report.stopTime,report.overall.low));
   seriesData.push(metrics.createSeriesData('Overall:Count',
                   report.stopTime,report.overall.count));

   var iterator = report.rangeStats.entrySet().iterator();
   while(iterator.hasNext())
   {
       var mapEntry = iterator.next();
       var rangeName = mapEntry.getKey();
       var rangeStats = mapEntry.getValue();
       seriesData.push(metrics.createSeriesData(rangeName +':Avg',
                       report.stopTime,rangeStats.average));
       seriesData.push(metrics.createSeriesData(rangeName +':Max',
                       report.stopTime,rangeStats.high));
       seriesData.push(metrics.createSeriesData(rangeName +':Min',
                       report.stopTime,rangeStats.low));
       seriesData.push(metrics.createSeriesData(rangeName +':Count',
                        report.stopTime,rangeStats.count));
   }
   return seriesData;
}
```


- "Properties" on page 166
- "Methods" on page 166

Properties

Property Name	Value Type	Description
`name`	`String`	*(Required)* Name of the data item.
`timestamp`	`java.util.Date`	*(Required)* Time stamp for the data item.
`Number`	`value`	*(Required)* Value of the data item.
`expiryDays`	`integer`	Number of days this item should be kept in the active database for reporting.

Methods

`String toString()`	Returns a string description of the object.
`Boolean isValid()`	Returns True if all required fields are populated. If False is returned, the scripting log contains a description of the failure.

A.2.11 SummaryDataCollector

SummaryDataCollector objects are used to analyze the HVEvent stream and periodically produce summary reports describing the content of the event stream. These objects are generally created using one of the metrics.createSummaryCollector() methods and deployed in the implementation of an metrics.onload event-handler.

If the collector is deployed, every HVEvent that gets past the preprocessor is analyzed by it. First, the event is checked against inclusionFilters and exclusionFilters. If any inclusionFilters are defined, then an event must match at least one of them, or it is excluded. A match to an exclusionFilter also causes the event to be excluded. If an event passes the filters, the event is included in the next set of interval reports generated by this collector.

Collectors generate reports at the frequency specified by the intervalDuration property and delivered to the function specified by the onreport property. Separate reports are generated for each unique set of grouping field values and include all recorded activity for all HVEvent objects with the same set of grouping field values. After reports are delivered to the handler, the collector forgets that information, so if a given set of grouping field values has no activity in the next interval, no report is generated for it.

In a distributed HVM environment, identical collectors are deployed to all members of the HVM group. At the end of each interval, reports are collected from all HVM members and their content is consolidated into a single set of reports. For example, if a data collector specified page title as the sole grouping field and at the end of the interval HVMGroupMember1 reported the following statistics for the page called `'User Login'`:

count= 293
average=1372
high=3599
low=141

and HVMGroupMember2 reported the following for the same page:

count= 311
average=1821
high=3801
low=221

Then the report delivered to the onreport event handler can contain the following:

report.key.groupValues=['User Login']
report.overall.count=604
report.overall.average=1603
report.overall.high=3801
report.overall.low=221
report.reportingMonitors=['HVMGroupMember1:6789', 'HVMGroupMember2:6789']

This consolidation of reports with the same key and interval values is automatic in a distributed HVM configuration. On occasion, a remote monitor can also deliver a report from a previous interval, because some transient condition prevented it from reporting when that interval's data was processed (see Distributed High-Volume-Metrics Data Collection). In this situation, the stale report data (data from an earlier interval) can be rolled into the current interval's data, reported to the event handler as a separate report for the old interval, or filtered out entirely. The value of the staleReportOptions property determines the behavior. Appropriate values are described in `metrics.constants Stale Report Options`.

```
var appFilter = metrics.createFieldFilter('Banking Filter',
          metrics.constants.FN_APPLICATION_NAME,
          'BANKING');
var devSystemFilter = metrics.createFieldFilter('Dev System Filter',
          metrics.constants.FN_URL,
          'http://DEV.*');

var targetRange = metrics.createResponseTimeRange('target', 0, 1500);
var acceptRange = metrics.createResponseTimeRange('accept', 1500, 4000);
var problemRange = metrics.createResponseTimeRange('problem', 4000, 60000);
var collector = metrics.createSummaryCollector();
collector.name= 'Banking';
collector.intervalDuration=60000;
collector.onreport=processBanking;
collector.groupingFieldNames=[metrics.constants.FN_PAGE_TITLE];
collector.ranges=[targetRange,acceptRange,problemRange];
collector.inclusionFilters=[appFilter];
collector.exclusionFilters=[devSystemFilter];
collector.staleReportOption=metrics.constants.SRO_PASS;

// deploy with preprocessor and other collectors.

function processBanking(reports)
{
  // do Banking application processing
}
```


- "Properties" on page 168
- "Methods" on page 169

Properties

Property Name	Value Type	Description
`name`	`String`	*(Required)* Name of the collector.
`intervalDuration`	`interval`	*(Required)* Frequency with which the collector should produce reports, specified in milliseconds.
`onreport`	`Function`	*(Required)* Is used to specify the event-handler to be called when the collector has reports to be processed. The expected handler signature is `void yourHandlerName(SummaryReport[] reports)`.
`groupingFieldNames`	`String[]`	List of HVEvent field names whose unique values will produce a discrete report.
`inclusionFilters`	`FieldFilter[]`	List of filters that an HVEvent must match in order to be included in this collector's reports.
`exclusionFilters`	`FieldFilter[]`	List of filters that an HVEvent might not match if it is to be included in this collector's reports.
`ranges`	`ResponseTimeRange[]`	List of ranges for which a discrete set of statistics should be maintained.
`staleReportOption`	`String[]`	Instruction indicating how stale reports should be handled in a distributed metrics environment.

Methods

`Boolean isValid()`	Returns True if all required values are populated and all populated values are valid.
`String toString()`	Returns a string description of the SummaryDataCollector.

A.2.12 SummaryReport

SummaryReport objects are used to report on activity captured within a reporting interval. Deployed SummaryCollector objects generate a separate report for each unique set of grouping field values and pass them to the collector's onreport event handler. The key.groupValues property specifies the set of unique values that define the scope of the report. If the collector does not define any grouping fields, it produces a single report per interval in which it saw activity with no specified key.groupingValues.

Each report instance captures the overall statistics, as well as statistics for any defined ranges where at least one hit fell within the range's constraints. Report data is not generated for grouping values that saw no activity or for ResponseTimeRanges that saw no activity.

Example

The following example defines a SummaryCollector with a one-minute interval for which it reports any activity observed to the `processPageStats` function. This collector groups information by page title, so for each unique page title appearing in an HVEvent during the interval, the collector generates a report, with the name of the page being reported on specified in report.key.groupValues[0].

As this custom handler receives these reports, it generates metrics elements that appear on a connected Operations Center server. As this script is written, the element name matches the page title, and appears under *Elements* > *Experience Manager_adapter* > *Metrics* > *Page Stats* > *page_title* in the element tree. The properties page for this element then has data mined from the Summary Reports. (It can presumably also have Series data showing a history of the page performance, but that logic is omitted here for simplicity.)

This example uses custom logic for driving the condition of the element. The collector defines a set of ranges that presumably have business meaning in the context of the site being monitored. Based on the relative number of hits in each of these ranges, the script assigns an appropriate condition and message (description) for the element.

```
var collector = metrics.createSummaryCollector('Page Stats', 60000,
                processPageStats);
collector.groupingFieldNames=[metrics.constants.FN_PAGE_TITLE];
var targetRange = metrics.createResponseTimeRange('target', 0, 1500);
var acceptRange = metrics.createResponseTimeRange('accept', 1500, 4000);
var problemRange = metrics.createResponseTimeRange('problem', 4000, 60000);

collector.ranges=[targetRange,acceptRange,problemRange];

// ... add other collector props and deploy

function processPageStats(reports)
{
   for(var i=0; i < reports.length;i++)
```

```
   {
      var report = reports[i];
      // create element and automatically map stats into series data.
      var elem = metrics.createMetricsElement()
      elem.elementKeys=[report.key.collectorName, report.key.groupValues[0]];
      assignConditonAndMessage(elem, report);
      elem.properties.put('Reference URL', report.lastEvent.URL);
      elem.properties.put('Page Title', report.lastEvent.title);
      elem.properties.put('Hit Count', report.overall.count);
      // mine other properties ..
      metrics.sendMetricsElement(elem);

   }
}

/*
 * calculate element condition based on relative hit count
 */
function assignConditonAndMessage(elem,report)
{
   var targetHits = getRangeHitCount(report,'target');
   var acceptHits = getRangeHitCount(report,'accept');
   var problemHits = getRangeHitCount(report,'problem');

   if(targetHits > acceptHits + problemHits)
   {
      elem.condition=metrics.constants.SEV_OK;
      elem.message = 'Performance target achieved';
   }
   else if (targetHits  > problemHits || acceptHits > problemHits)
   {
      elem.condition = metrics.constants.SEV_MINOR;
      elem.message = 'Performance acceptable';
   }
   else if (targetHits + acceptHits > problemHits)
   {
      elem.condition = metrics.constants.SEV_MAJOR;
      elem.message = 'Performance degraded';
   }
   else
   {
      elem.condition = metrics.constants.SEV_CRITICAL;
      elem.message = 'Performance violation';
   }
   elem.message += ': T=' + targetHits + ' A='+ acceptHits + ' P=' + problemHits;
}
/*
 * Return hit count if range has data, or 0 if it does not
 */
function getRangeHitCount(report, rangeName)
{
   var hitCount = 0;
   var rangeStats = report.rangeStats.get(rangeName);
   if(rangeStats != null)
   {
      hitCount =  rangeStats.count;
   }
   return hitCount;
}
```

Properties

Property Name	Value Type	Description
key	ReportKey	Identifier describing the scope covered by this report.

Property Name	Value Type	Description
endTime	java.util.Date	End of the period covered by this report.
collector	SummaryCollector	Collector that generated this report.
lastEvent	String[]	A copy of the last event captured during the interval of this report.
overall	FieldFilter[]	Overall statistics for this report.
rangeStats	java.util.Map<String,*summary_stats*>	Map of SummaryStat for each range that had activity. Map key is the name of the configured ResponseTimeRange.
reportingMonitors	String[]	List of monitors that contributed to this report.
nonreportingMonitors	String[]	List of monitors in the group that could not be contacted when this report was assembled.

Methods

String toString()	Returns a string representation of this object.

A.2.13 SummaryStats

SummaryStats objects are a collection of statistics that appear in a number of contexts inside a SummaryReport or a SamplingReport. Refer to the containing report type for more information.

Properties

Property Name	Value Type	Description
count	Number	*(Read-only)* Number of hits in the reported scope.
high	Number	*(Read-only)* Highest response time (in milliseconds) in the scope.
low	Number	*(Read-only)* Lowest response time in the scope.
average	Number	*(Read-only)* Arithmetic average of the response times in the scope

Methods

String toString()	Returns a string representation of this object.

A.2.14 SyntheticAlarm

A SyntheticAlarm object represents an alarm generated by the Experience Manager Monitor test engine. All properties are read-only. SyntheticAlarm objects are reported to the testEngine.onScenarioComplete and testEngine.onTestComplete event handlers to support Synthetic Test Management functionality.

For an example, see Section A.2.18, "SyntheticTestResults," on page 177.

Properties

Property Name	Value Type	Description
`testName`	`String`	*(Read-only)* Name of the test that produced this alarm.
`scenarioName`	`String`	*(Read-only)* Name of the test scenario that produced this alarm.
`scenarioSeqNum`	`Number`	*(Read-only)* Sequence number of the originating test scenario within its enclosing test.
`responseTime`	`Number`	*(Read-only)* Execution time reported by the scenario. A negative number indicates an execution failure.
`severity`	`Object`	*(Read-only)* Severity of the alarm. Value corresponds to one of the values in testEngine.constants severities.
`description`	`String`	*(Read-only)* Text description of the alarm.
`userNum`	`Number`	(*(Read-only)* Simulated user number. The value is always 1, unless SyntheticTest.userCount is greater than 1.
`timestamp`	`java.util.Date`	Time when the alarm was generated.

Methods

`String toString()`	Returns a string representation of this object.
`Boolean isFailed()`	Returns a string representation of this object.

A.2.15 SyntheticDeployment

A SyntheticDeployment object provides access to the collection of SyntheticTest objects currently deployed to the monitor. This deployment is created and managed in the Experience Manager adapter using the Operations Center console. A read-only view is exposed in the Monitor's scripting layer to facilitate Web Operations and Synthetic Test Management functionality.

The following example shows a simple WebOps that displays the content of a requested SyntheticDeployment.

```
function showDeploymentOp(parmMap)
{
   var html = '<table border="2">';
   var tests = testEngine.getDeployment().getTests();

   for(var i=0;i< tests.length; i++)
   {
      html += addItem('Test -' + tests[i].name,testToHTML(tests[i]))
   }
   html+='</table>';
   return html;
}


function testToHTML(test)
{
   var html = '<table border="1">';
   html += addItem('name', test.name);
   html += addItem('userCount', test.userCount);
   html += addItem('interval', test.interval);
   html += addItem('enabled', test.enabled);
   html += addItem('includeRollup', test.includeRollup);
   var scenarios = test.getScenarios();
   for(var i=0;i< scenarios.length; i++)
   {
      html+= addItem('scenario-' +
scenarios[i].seqNum,scenarioToHTML(scenarios[i]));
   }

   html += '</table>';
   return html;

}
function scenarioToHTML(scenario)
{
   var html = '<table>';
   html += addItem('name', scenario.name);
   html += addItem('type', scenario.type);
   html += addItem('enabled', scenario.enabled);
   html += addItem('testName', scenario.testName);
   html += addItem('seqNum', scenario.seqNum);
   html += addItem('delay', scenario.delay);
   html += addItem('timeout', scenario.timeout);
   html += addItem('retries', scenario.retries);
   html += addItem('continueOnFailure', scenario.continueOnFailure);
   html += addItem('includeRollup', scenario.includeRollup);
   html += '</table>';
   return html;

}
function addItem(typeName, value)
{
   return '<tr><td>' + typeName + '</td><td>' + value + '</td></tr>';
}
```

There are no properties associated with the SyntheticDeployment object.

- "Methods" on page 174

Methods

`SyntheticTest[] getTests()`	Returns an array of all SyntheticTest objects currently deployed to this monitor.
`SyntheticTest getTest(String name)`	Returns the test in this deployment where the SyntheticTest.name is equal to the provided name. If no test has the provided name, null is returned.
`String toString()`	Returns a string representation of this object.

A.2.16 SyntheticScenario

A SyntheticScenario object represents a scenario in a SyntheticTest. These scenarios are created and managed in the Experience Manager adapter using the Operations Center console. A read-only view is exposed in the Monitor's scripting layer to facilitate Web Operations and Synthetic Test Management functionality.

The following example shows a simple WebOps that displays the content of a requested SyntheticScenario.

```
function showScenarioOp(parmMap)
{
   var testName=parmMap.get("testName");
   var index =parmMap.get("index");
   var deployment = testEngine.getDeployment();
   var test = deployment.getTest(testName);
   if(test == null)
   {
      return 'Test ' +testName + ' not found';
   }
   var scenario = test.getScenarioBySeqNum(index);
   if(scenario == null)
   {
      return 'Scenario ' + index + ' not found in test ' + testName;
   }

   return scenarioToHTML(scenario);
}

function scenarioToHTML(scenario)
{
   var html = '<table>';
   html += addItem('name', scenario.name);
   html += addItem('type', scenario.type);
   html += addItem('enabled', scenario.enabled);
   html += addItem('testName', scenario.testName);
```

```
   html += addItem('seqNum', scenario.seqNum);
   html += addItem('delay', scenario.delay);
   html += addItem('timeout', scenario.timeout);
   html += addItem('retries', scenario.retries);
   html += addItem('continueOnFailure', scenario.continueOnFailure);
   html += addItem('includeRollup', scenario.includeRollup);
   html += '</table>';
   return html;

}
function addItem(typeName, value)
{
   return '<tr><td>' + typeName + '</td><td>' + value + '</td></tr>';
}
```


- "Properties" on page 175
- "Methods" on page 175

Properties

Property Name	Value Type	Description
`name`	`String`	*(Read-only)* Name of the scenario.
`type`	`Object`	*(Read-only)* Type of scenario corresponding to one of the testEngine.constants values.
`enabled`	`Boolean`	*(Read-only)* Sequence number of the originating test scenario within its enclosing test.
testName	`String`	*(Read-only)* Name of the test containing the scenario.
`seqNum`	`Number`	*(Read-only)* Sequence number of the scenario within the enclosing test.
`delay`	`Number`	((*Read-only)* Artificial delay (in milliseconds) inserted before the scenario executes.
`timeout`	`Number`	*(Read-only)* Amount of time (in milliseconds) before the scenario is marked as failed because it took too long.
`retries`	`Number`	*(Read-only)* Number of times to retry the scenario in the event of a failure.
`continueOnFailure`	`Boolean`	Whether the test should continue to the next scenario if this scenario fails.
`includeRollup`	`Boolean`	Whether a roll-up alarm should be generated with this scenario with the cumulative response time since the previous roll-up.

Methods

`String toString()`	Returns a string representation of this object.

A.2.17 SyntheticTest

A SyntheticTest object represents a test in a SyntheticDeployment. These tests are created and managed in the Experience Manager adapter using the Operations Center console. A read-only view is exposed in the Monitor's scripting layer to facilitate Web Operations and Synthetic Test Management functionality.

The following example shows a simple WebOps that displays the content of a requested SyntheticTest.

```
function showTestOp(parmMap)
{
   var testName=parmMap.get("testName");
   var deployment = testEngine.getDeployment();
   var test = deployment.getTest(testName);
   if(test == null)
   {
      return 'Test ' +testName + ' not found';
   }

   return testToHTML(test);
}


function testToHTML(test)
{
   var html = '<table border="1">';
   html += addItem('name', test.name);
   html += addItem('userCount', test.userCount);
   html += addItem('interval', test.interval);
   html += addItem('enabled', test.enabled);
   html += addItem('includeRollup', test.includeRollup);
   var scenarios = test.getScenarios();
   for(var i=0;i< scenarios.length; i++)
   {
      html+= addItem('scenario-' +
scenarios[i].seqNum,scenarioToHTML(scenarios[i]));
   }

   html += '</table>';
   return html;

}
function scenarioToHTML(scenario)
{
   var html = '<table>';
   html += addItem('name', scenario.name);
   html += addItem('type', scenario.type);
   html += addItem('enabled', scenario.enabled);
   html += addItem('testName', scenario.testName);
   html += addItem('seqNum', scenario.seqNum);
   html += addItem('delay', scenario.delay);
   html += addItem('timeout', scenario.timeout);
   html += addItem('retries', scenario.retries);
   html += addItem('continueOnFailure', scenario.continueOnFailure);
   html += addItem('includeRollup', scenario.includeRollup);
   html += '</table>';
   return html;

}
function addItem(typeName, value)
{
   return '<tr><td>' + typeName + '</td><td>' + value + '</td></tr>';
}
```


- "Properties" on page 177
- "Methods" on page 177

Properties

Property Name	Value Type	Description
`name`	`String`	*(Read-only)* Name of the test.
`userCount`	`Number`	*(Read-only)* Number of concurrent simulated users to run for this test.
`interval`	`Number`	*(Read-only)* Time (in milliseconds) between executions of this test.
`enabled`	`Boolean`	*(Read-only)* True if the test is activated.
`includeRollup`	`Boolean`	Whether a roll-up alarm should be generated with this test with the cumulative response time of all scenarios in the test.

Methods

`String toString()`	Returns a string representation of this object.
`SyntheticScenario[] getScenarios()`	Returns an ordered array of all scenarios in this test as SyntheticScenario objects.
`SyntheticScenario getScenarioByName(String name)`	Returns the scenario in this test where the SyntheticScenario.name is equal to the provided name. If no scenario has the provided name, null is returned.
`SyntheticScenario getScenarioBySeqNum(Number seqNum)`	Returns the scenario in this test where the SyntheticScenario.seqNum is equal to the provided value. If no scenario has the provided sequence number, null is returned.

A.2.18 SyntheticTestResults

A SyntheticTestResults object reports the results of a completed user test execution by the Experience Manager Monitor test engine. All properties are read-only. SyntheticTestResults objects are reported to the testEngine.onTestComplete event handler to support Synthetic Test Management functionality.

The following example shows how to use the SyntheticTestResults object and related objects and services to implement conditional execution of a test. The example is purposely simplified to show the core functionality without getting bogged down in details of configuration management.

```
/*
 * Conditional execution: execute test identified in 'conditionalTestName' once if
 * a triggerScenario in 'mainTestName' produces a Critical alarm
 */
var mainTestName='test1';
var triggerScenarios='[1][5]';
var conditionalTestName='test2';

testEngine.onTestComplete=handleTestComplete;
testEngine.onScenarioComplete=handleScenarioComplete;

function handleTestComplete(testResults)
{
   if(testResults.test.name == conditionalTestName)
   { // stop conditional test once it runs once
```

```
      testEngine.deactivateTest(conditionalTestName);
   }
}

function handleScenarioComplete(alarm)
{
   // if this is the alarm is from the right test and its critical
   if(alarm.testName == mainTestName &&
      alarm.severity == testEngine.constants.SEV_CRITICAL)
   {
      // and its one of the trigger scenarios
      if(triggerScenarios.indexOf('['+alarm.scenarioSeqNum + ']') >=0)
      {
         // and we did not already start the conditional test
         if(!testEngine.getDeployment().getTest(conditionalTestName).enabled)
         {
            // then start the conditional test
            testEngine.activateTest(conditionalTestName);
         }
      }
   }
}
```


- "Properties" on page 178
- "Methods" on page 178

Properties

Property Name	Value Type	Description
`test`	`SyntheticTest`	*(Read-only)* The test reporting these results.

Methods

`String toString()`	Returns a string representation of this object.
`SyntheticAlarm[] getAlarms()`	Returns an ordered array of all alarms generated during the latest test execution.
`Number getFailCount()`	Returns the number of alarms where SyntheticAlarm.isFailed() is True.
`SyntheticAlarm[] getAlarmsBySeverity(Object severity`	Returns an array containing any alarms with the specified severity. Valid severity values are listed in testEngine.constants severities.