

Web 2.0 Connect Guide

Operations Center 5.5

March 3, 2014

Legal Notices

THIS DOCUMENT AND THE SOFTWARE DESCRIBED IN THIS DOCUMENT ARE FURNISHED UNDER AND ARE SUBJECT TO THE TERMS OF A LICENSE AGREEMENT OR A NON-DISCLOSURE AGREEMENT. EXCEPT AS EXPRESSLY SET FORTH IN SUCH LICENSE AGREEMENT OR NON-DISCLOSURE AGREEMENT, NETIQ CORPORATION PROVIDES THIS DOCUMENT AND THE SOFTWARE DESCRIBED IN THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW DISCLAIMERS OF EXPRESS OR IMPLIED WARRANTIES IN CERTAIN TRANSACTIONS; THEREFORE, THIS STATEMENT MAY NOT APPLY TO YOU.

For purposes of clarity, any module, adapter or other similar material ("Module") is licensed under the terms and conditions of the End User License Agreement for the applicable version of the NetIQ product or software to which it relates or interoperates with, and by accessing, copying or using a Module you agree to be bound by such terms. If you do not agree to the terms of the End User License Agreement you are not authorized to use, access or copy a Module and you must destroy all copies of the Module and contact NetIQ for further instructions.

This document and the software described in this document may not be lent, sold, or given away without the prior written permission of NetIQ Corporation, except as otherwise permitted by law. Except as expressly set forth in such license agreement or non-disclosure agreement, no part of this document or the software described in this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, or otherwise, without the prior written consent of NetIQ Corporation. Some companies, names, and data in this document are used for illustration purposes and may not represent real companies, individuals, or data.

This document could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein. These changes may be incorporated in new editions of this document. NetIQ Corporation may make improvements in or changes to the software described in this document at any time.

U.S. Government Restricted Rights: If the software and documentation are being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), in accordance with 48 C.F.R. 227.7202-4 (for Department of Defense (DOD) acquisitions) and 48 C.F.R. 2.101 and 12.212 (for non-DOD acquisitions), the government's rights in the software and documentation, including its rights to use, modify, reproduce, release, perform, display or disclose the software or documentation, will be subject in all respects to the commercial license rights and restrictions provided in the license agreement.

© 2014 NetIQ Corporation. All Rights Reserved.

For information about NetIQ trademarks, see https://www.netiq.com/company/legal/ (https://www.netiq.com/company/legal/).

All third-party trademarks are the property of their respective owners.

Contents

About This Guide 5

1 About Web 2.0 Connect 7

1.1 Architecture . . . 7
1.2 Accessing the Published API . . . 9

2 Installation 11

2.1 Requirements . . . 11
2.2 Deploying Web 2.0 Connect . . . 11
2.3 Verifying the Deployment . . . 13
2.3.1 Creating a Session . . . 13
2.3.2 Verifying Whether a Session Has Been Created . . . 13
2.3.3 Viewing a List of Child Elements . . . 14
2.3.4 Verifying the Dashboard Infrastructure . . . 14
2.4 Installing Related Libraries . . . 14
2.5 Upgrading Web 2.0 Connect . . . 15

3 Programming Interfaces 17

3.1 Understanding Representational State Transfer (REST) . . . 17
3.2 Understanding Simple Object Access Protocol (SOAP) . . . 18

4 Getting Connected 21

4.1 Using a Web Browser to Access Operations Center Data . . . 21
4.1.1 Creating a Session . . . 21
4.1.2 Reading an Element . . . 22
4.1.3 Encoding DNames in REST Calls . . . 23
4.1.4 Understanding Session Retention . . . 23
4.2 Using JavaScript and XHTML . . . 24
4.2.1 Creating a Session . . . 25
4.2.2 Reading an Element . . . 26
4.2.3 Encoding an Element DName . . . 28
4.2.4 Configuring Session Retention . . . 29
4.2.5 Performing an Operation . . . 31
4.2.6 Closing a Session . . . 34
4.3 ECMA/JavaScript Library . . . 35
4.3.1 Example Usage . . . 35
4.3.2 Library Source . . . 36
4.3.3 Methods and Properties . . . 54
4.4 Example Code for JavaScript Library . . . 55
4.4.1 Example Data . . . 63

5 Flex Quick Start 65

5.1 Using HTTPService . . . 65
5.1.1 Creating a Session using HTTPService . . . 65
5.1.2 Closing a Session using HTTPService . . . 67

5.2 Using Remote Procedure Calls 68
5.2.1 Configuring Remote Services 69
5.2.2 Services Configuration 70
5.2.3 Building and Deploying 71
5.2.4 Creating a Session using RPC 71
5.2.5 Closing a Session using RPC 72
5.2.6 A Word About Security 73

6 Flex Library 75

6.1 Accessing Element Properties 75

7 Flex Solutions 77

7.1 Example: Flex Alarm Ratio Bar 77
7.2 Example: Alarm Ratio Bar Application 78
7.3 Example: Alarm Ratio Bar Component 80
7.4 Example: Metamodel Property Page 84

About This Guide

The *Web 2.0 Connect Guide* provides information on a tool for developers of Rich Internet Applications to access and utilize Operations Center resources (data and/or functions) in their applications.

- Chapter 1, “About Web 2.0 Connect,” on page 7
- Chapter 2, “Installation,” on page 11
- Chapter 3, “Programming Interfaces,” on page 17
- Chapter 4, “Getting Connected,” on page 21
- Chapter 5, “Flex Quick Start,” on page 65
- Chapter 6, “Flex Library,” on page 75
- Chapter 7, “Flex Solutions,” on page 77

Audience

This guide is intended for Operations Center system administrators.

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 *Web 2.0 Connect 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

In NetIQ documentation, 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 the elements to expand them.

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

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.

1 About Web 2.0 Connect

Web 2.0 Connect is a tool for developers of Rich Internet Applications to access and utilize Operations Center resources (data and/or functions) in their applications. It:

- Can be used to build distributed solutions when there is a need to access Operations Center resources using the HTTP protocol
- Provides a more simple application programming interface (API) than SOAP-based Web services
- Has an API that is particularly suited to building Web 2.0 style applications, but is equally suitable as an API for non-Web based solutions
- Provides solutions architects and developers with a simple API for interapplication communication to Operations Center
- Is based on the Representational State Transfer (REST) architecture, which provides a consistent and simple method of accessing a system's resources

  For more information about REST, see http://en.wikipedia.org/wiki/Representational_State_Transfer.

Web 2.0 Connect has applications that can be diverse in their usage, ranging from Web-hosted HTML applications to stand-alone desktop clients. The distribution of the solution depends on the type of application architecture. Applications developed using Web 2.0 Connect are considered standard Web applications and can be distributed on any Java-based Web Server. They can also be distributed using the Operations Center dashboard. The Operations Center dashboard is a portal that allows users to customize their own Web pages with content from Operations Center, as well as other sources.

The Flex examples in this book were created using Adobe Flex Builder, which is highly recommended for developing Flash based solutions.

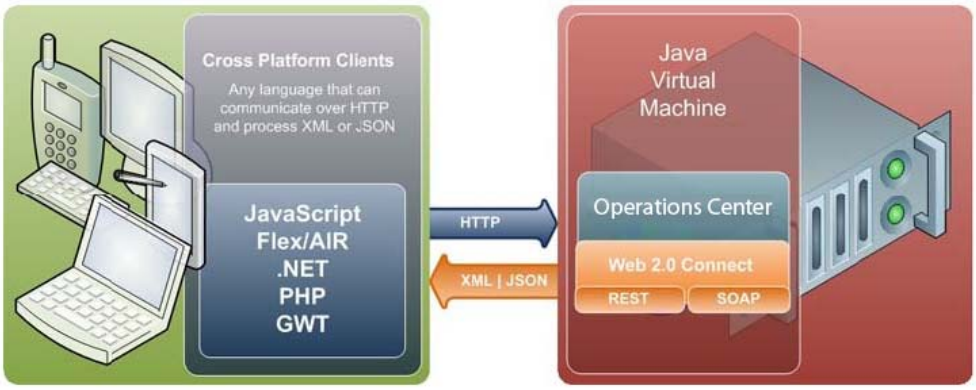
- Section 1.1, "Architecture," on page 7
- Section 1.2, "Accessing the Published API," on page 9

1.1 Architecture

Web 2.0 Connect is a platform-independent API layer for Operations Center. Client applications use the HTTP protocol to submit queries to one of the Web 2.0 Connect APIs and in response, receive a response in XML or JavaScript Object Notation (JSON).

Web 2.0 Connect can be used with any programming language that provides asynchronous support for the HTTP protocol and processing of XML documents or JSON streams.

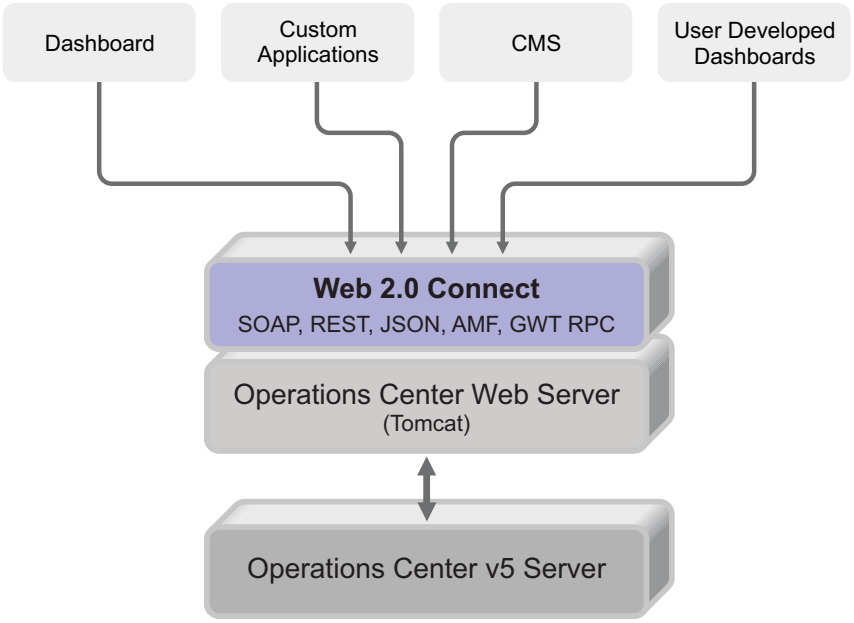
Developers can use development tools of their choice. The supported protocols include REST-style URI access using XML or JSON formats, classic SOAP access (with XML-Schema/XSD), GWT RPC and AMF (Action Media Format) RPC. Each of these protocols can be mixed and matched according to needs. As an example, a Flex-based application might utilize REST/XML and AMF RPC. A GWT application might use REST/JSON and GWT/RPC.



Web 2.0 Connect provides an expandable layer based on new environments used by Rich Internet Applications developers. Web2Connect provides middleware Web services access for browser-based applications. The Web services are provided in a variety of protocols and supply access to the Operations Center data model.

The data model access includes querying element information, element relationships, parent/child relationships, element properties, element alarms, and element time series data (for charting). In addition, element operations are supported, as well as element and relationship creates, updates, and removes.

Figure 1-1 *Web 2.0 Connect Architecture*



Integrated session management is provided in the dashboard Web application architecture. A developer writing a portlet can avoid performing any reauthentication and can focus on data access. Thus, when a portlet is rendered in a browser, it expects that prior authentication and session establishment with the dashboard server has occurred, leaving the developer to only focus on the query APIs required for reporting.

Conversely, a stand-alone application must start with the construction of a session with the backend, and must also log out when the application closes. The session management layer allows subsequent method calls to Web 2.0 Connect to avoid re-logins with credentials (as a classic SOAP application requires).

1.2 Accessing the Published API

Developers can access a published API to communicate with Operations Center. This API is continuously updated, as necessary. For the latest API references, always refer to the API list on the Web 2.0 Connect home page. To view the online documentation, deploy the `moweb2cn.war` file from the */OperationsCenter_Dashboard_install_path/*`web2connect` directory to the */OperationsCenter_Dashboard_install_path/*`server/webapps` directory of a Tomcat server or other application server.

2 Installation

This section covers the following Web 2.0 Connect installation topics:

- Section 2.1, “Requirements,” on page 11
- Section 2.2, “Deploying Web 2.0 Connect,” on page 11
- Section 2.3, “Verifying the Deployment,” on page 13
- Section 2.4, “Installing Related Libraries,” on page 14
- Section 2.5, “Upgrading Web 2.0 Connect,” on page 15

2.1 Requirements

Web 2.0 Connect requires:

- An instance of the Operations Center server and Operations Center dashboard server, or any other J2EE Webserver with a servlet engine, such as Apache Tomcat

  The recommended minimum versions are 4.5 with the July 2008 service packs for Operations Center and its dashboard.
- A separate license key than the Operations Center server
- Must be hosted by a Java Servlet container, such as Apache Tomcat

  The simplest deployment method is to deploy the application using the dashboard Tomcat container.

2.2 Deploying Web 2.0 Connect

To deploy Web 2.0 Connect:

1. Install Operations Center.

   For instructions, see the *Operations Center 5.5 Server Installation Guide*.
2. Install the Operations Center Dashboard.

   For instructions, see “Dashboard Installation” in the *Operations Center 5.5 Dashboard Guide*.
3. Configure and confirm the Dashboard is working.

   For instructions, see “Starting the Dashboard for the First Time and Verifying the Installation” in the *Operations Center 5.5 Dashboard Guide*.
4. Copy the Web 2.0 Connect Web archive (`/OperationsCenter_Dashboard_install_path/web2connect/moweb2cn.war`) to the `/OperationsCenter_Dashboard_install_path/server/webapps` directory.
5. Restart the Operations Center Dashboard server to deploy the .war file copied in Step 4.

For instructions, see "Stopping the Dashboard" and "Starting the Dashboard" in the *Operations Center 5.5 Dashboard Guide*.

6 Configure the dashboard `/OperationsCenter_Dashboard_install_path/server/webapps/moweb2cn/WEB-INF/web.xml` file to direct Web 2.0 Connect to the appropriate Operations Center server:

- **ManagedObjectsServerHost:** set to the Operations Center server host name.
- **ManagedObjectsServerwebPort:** defaults to `8080`, or replace with the Operations Center server Web port number.

  This value corresponds to the Web Service Port setting in the Configuration Manager. See "Web Server Pane" in the *Operations Center 5.5 Server Configuration Guide*.
- **ManagedObjectsServerPort:** defaults to `1099`, or replace with the Operations Center server RMI port number.

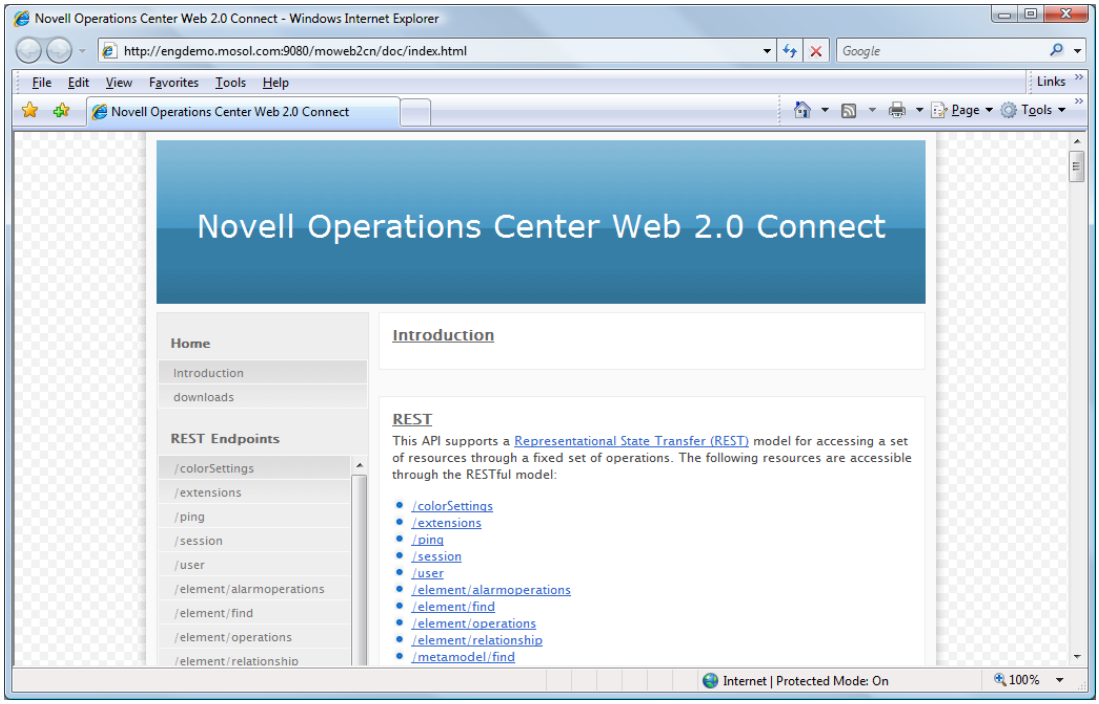
  This value corresponds to the Remote Services Port setting in the Configuration Manager. See "NOC Server Pane" in the *Operations Center 5.5 Server Configuration Guide*.
- **ManagedObjectsServerwebProtocol:** defaults to `http`.

7 Restart the dashboard server.

8 Navigate to the Web 2.0 Connect home page:

`http://OperationsCenter_Dashboard_Server:port/moweb2cn/`

The Operations Center 2.0 Connect Web page displays:



9 Continue to Section 2.3, "Verifying the Deployment," on page 13.

2.3 Verifying the Deployment

The following topics describe how to create sessions and verify your Web 2.0 Connect deployment:

- Section 2.3.1, “Creating a Session,” on page 13
- Section 2.3.2, “Verifying Whether a Session Has Been Created,” on page 13
- Section 2.3.3, “Viewing a List of Child Elements,” on page 14
- Section 2.3.4, “Verifying the Dashboard Infrastructure,” on page 14

2.3.1 Creating a Session

To access data from Operations Center, you must initiate a session first with a username and password.

To create a session:

1 Go to

```
http://OperationsCenter_Dashboard_Server:port/moweb2cn/rest/
session?username=admin&password=formula
```

The session is created and a valid XML document indicating success displays, as follows:

```
<response>
   <value>OK</value>
</response>
```

Note that all parameter values should be URL encoded.

2 If the validation displays, continue with Section 2.3.3, “Viewing a List of Child Elements,” on page 14.

2.3.2 Verifying Whether a Session Has Been Created

To verify a session:

1 Using any Web browser, go to:

```
http://OperationsCenter_Dashboard_Server:port/moweb2cn/rest/ping
```

If a session exists, the following validation displays:

```
<response>
    <value>OK</value>
</response>
```

2 If the validation displays, continue with Section 2.3.3, "Viewing a List of Child Elements," on page 14.

2.3.3 Viewing a List of Child Elements

To view a list of child elements:

1 To view a list of the child elements of the top-level *Elements* branch within Operations Center, go to

```
http://OperationsCenter_Dashboard_Server:port/moweb2cn/rest/element/children/
root=Elements
```

2 Continue with Section 2.3.4, "Verifying the Dashboard Infrastructure," on page 14.

2.3.4 Verifying the Dashboard Infrastructure

To verify the Dashboard infrastructure:

1 Log in to the dashboard.

For instructions, see the *Operations Center 5.5 Dashboard Guide*.

2 Go to

```
http://OperationsCenter_Dashboard_Server:port/ManagedObjectsPortlets/rest/
element/children/root=Elements
```

A listing of the Operations Center's child elements of the top-level *Elements* branch displays.

If a `No context` response is received, you are not logged into the context. You must issue a sign in request, such as: `http://dashboardServer:dashboardPort/ManagedObjectsPortlets/rest/session?username=admin&password=formula`

3 Verify that the dashboard infrastructure exists and is correct.

2.4 Installing Related Libraries

Web 2.0 Connect has a dependency on common Operations Center classes.

To install related libraries:

1 If an updated `mocommon.jar` is provided with Web 2.0 Connect, deploy it to the servlet container (the `common/lib` directory).

In the case of Apache Tomcat, this is normally the `/OperationsCenter_Dashboard_install_path/server/common/lib` directory.

2 In addition, deploy any service packs or individual patches provided for the Operations Center server.

2.5 Upgrading Web 2.0 Connect

To upgrade Web 2.0 Connect:

1. If you are upgrading from a previous version of Web 2.0 Connect, do the following:
   - **1a** Delete the existing deployed Web 2.0 Connect Web application.
   - **1b** Replace the WAR file with the newer Web 2.0 Connect WAR file.
2. If you have applications built using Web 2.0 Connect, merge them again with the new Web 2.0 Connect WAR file.

3 Programming Interfaces

This section introduces the application programming interfaces (API) provided by Web 2.0 Connect. This guide primarily focuses on using the RESTful API, because it is very simple to use and allows for solution examples to be easier to understand and explain.

The REST and SOAP APIs are covered in detail and this guide is not a reference for these protocols. To use these APIs it is assumed that the developer has a good understanding of Web technologies.

Web 2.0 Connect is continually updated to provide the latest interfaces inline with the evolution of Operations Center server. For the latest API references always refer to the API list on the Web 2.0 Connect home page after the API is installed and running.

Review the following sections to understand REST and SOAP:

- Section 3.1, "Understanding Representational State Transfer (REST)," on page 17
- Section 3.2, "Understanding Simple Object Access Protocol (SOAP)," on page 18

3.1 Understanding Representational State Transfer (REST)

REST is defined in Wikipedia as a term coined by Roy Fielding in his Ph.D. dissertation to describe an architecture style of networked systems.

The Web 2.0 Connect API supports a Representational State Transfer (REST) model for accessing a set of resources through a fixed set of operations. The REST model provides a loose coupled API is the easiest to make use of in applications, particularly Web applications that implement most of their operations in client side JavaScript.

The following resources are accessible through the RESTful model in Web 2.0 Connect:

- element/alarms/{channel}/history
- element/alarms/{channel}/realtime
- element/attributes
- element/children
- element/childrenAttributes
- element/find
- element/impacted
- element/name
- element/parent
- element/perform/{operationName}/execute
- element/properties/pages
- element/query/performance

- element/query/performance/{profile}/{expression}/series/{dname}?from={fromDate}&to={toDate}&limit={return-points-limitation}

  Specify date formats as `y-M-d-H-m-s`, `y-M-d H:m:s`, or time in milliseconds.
- element/related
- element/relationship
- element/relationships
- element/rootcause
- element/search
- element/value
- elementClass/{className}/metamodelPages
- ping
- session

NOTE: All resource parameter values, as indicated with curly braces above, should be URL encoded. For example, {channel} and {fromDate}.

3.2 Understanding Simple Object Access Protocol (SOAP)

SOAP is defined in Wikipedia as an XML based data exchange protocol often used to transfer messages over HTTPS. The SOAP protocol is a key component in Web services, implementing the messaging layer of Web services to provide data access and remote procedure calls (RPC).

The Web 2.0 Connect API is exposed through a set of WSI Basic Profile compliant SOAP v1.1 endpoints. The API supports XML binary Optimized Packaging (XOP) and SOAP Message Transmission Optimization Mechanism (MTOM) for transmission of binary data. The SOAP API is fully described by the following endpoints:

For Web2ConnectService, the Namespace is:

`http://moweb2cn.mosol.com/`

The WSDL can be accessed using this URL:

`http://`*`OperationsCenterServer`*`:`*`port`*`/moweb2cn/moweb2cn.wsdl`

The following methods are available on this endpoint:

- closeSession
- createElement
- createRelationship
- createSession
- deleteElement
- findElement
- findElements
- findParent
- getAlarms
- getChildren
- getElementAttributes

- getElementChildrenAttributes
- getHistoricalAlarms
- getImpactedInfo
- getMetamodelPropertyPages
- getPerformanceSeriesInfo
- getPropertyPages
- getRelated
- getRelationships
- getRootCauseTree
- performOperation
- ping
- queryPerformanceSeries
- removeRelationship
- removeRelationships
- renameElement
- searchElements
- updateElement
- updateRelationship

4 Getting Connected

This section provides easy to understand examples that use Web 2.0 Connect to fetch data from Operations Center server.

It is not the intention of this guide to teach software programming. It is assumed that the reader has experience of high level programming languages and at the very least an understanding of the basic principles of modern Web technologies.

- Section 4.1, “Using a Web Browser to Access Operations Center Data,” on page 21
- Section 4.2, “Using JavaScript and XHTML,” on page 24
- Section 4.3, “ECMA/JavaScript Library,” on page 35
- Section 4.4, “Example Code for JavaScript Library,” on page 55

4.1 Using a Web Browser to Access Operations Center Data

Using the REST protocol, it is possible to access Operations Center data using a Web browser alone. Data is returned in XML format and all modern Web browsers simply render the XML as a viewable document.

- Section 4.1.1, “Creating a Session,” on page 21
- Section 4.1.2, “Reading an Element,” on page 22
- Section 4.1.3, “Encoding DNames in REST Calls,” on page 23
- Section 4.1.4, “Understanding Session Retention,” on page 23

4.1.1 Creating a Session

To access data from Operations Center, a session must be initiated first with a user name and password. The user name must be a valid account in Operations Center.

To create a session using the REST API, go to:

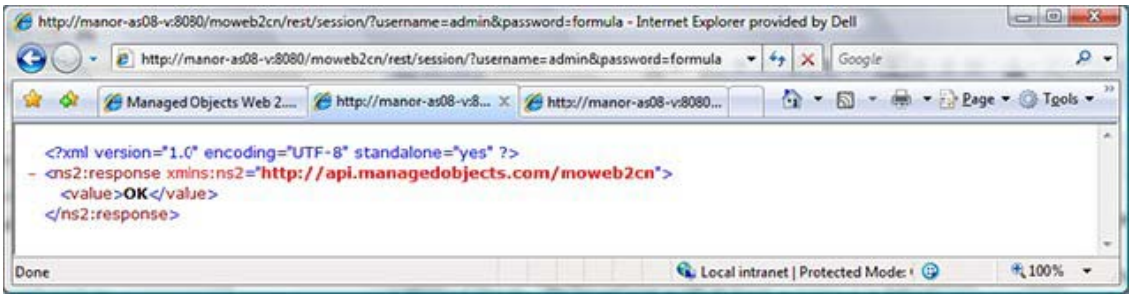
```
http://OperationsCenterServer:port/moweb2cn/rest/session/
?username=name&password=password
```

where *OperationsCenterServer*:*port* is the server host where the dashboard and Web 2.0 Connect are hosted and *name* and *password* is a valid credential for logging into the Operations Center application.

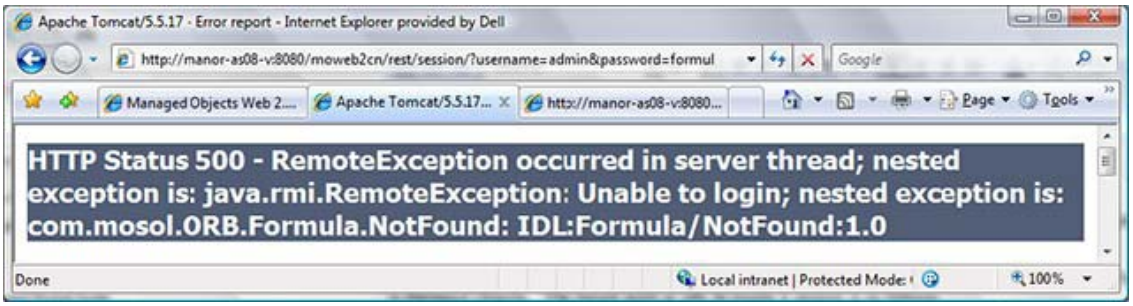
For example, where our server is named `manor-as08-v`:

```
http://manor-as08-v:8080/moweb2cn/rest/session/?username=admin&password=formula
```

The response is an XML document. If the session creation is successful the `<value/>` element contains the string `OK`, as shown:



If the session was not created, a HTTP Status 500 is returned:



After a session is created, subsequent attempts to create a session before the session expires, result in an error.

4.1.2 Reading an Element

After a session is successfully created other REST APIs can be used.

To read an element using the element/value mount point, go to:

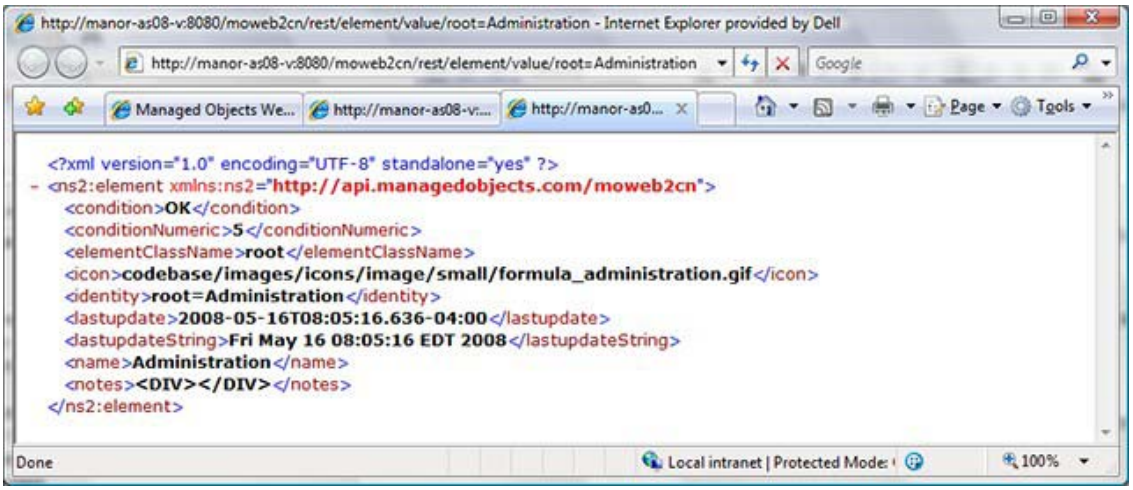
`http://`*`OperationsCenterServer`*`:`*`port`*`/moweb2cn/rest/element/value/`*`DName`*

where *OperationsCenterServer*:*port* is the server host where the dashboard and Web 2.0 Connect are hosted and *DName* is an element's distinguished name (DName).

For example, where our server is named `manor-as08-v`:

```
http://manor-as08-v:8080/moweb2cn/rest/element/value/root=Administration
```

The response is an XML document similar to the following:



TIP: The element XML metadata contains an `<identity>` tag, which can be useful to use as a unique key when building user interfaces.

4.1.3 Encoding DNames in REST Calls

As a best practice, and to avoid technical issues with REST calls when some characters are present in DNames, use a base 64 encoded hash to encode the DName.

When encoding the DName, be sure to precede the encoded DName with `identity:dname:`

For example, if the URL is:

```
http://manor-as08-v:8080/moweb2cn/rest/element/alarms/GENERAL/realtime/
Node=10.99.104.45/Nodes=Nodes/ncool=Adapter%3A+IBM+Micromuse+Netcool%2
FOMNIbus%28r%29/root=Elements
```

Then, the same URL with the base 64 encoded DName is:

```
http://manor-as08-v:8080/moweb2cn/rest/element/alarms/GENERAL/realtime/
identity:dname:Tm9kZT0xMC45OS4xMDQuNDUvTm9kZXM9Tm9kZXMvbmNvb2w9QWRhcHRlciUzQStJQk0
rTWljcm9tdXNlK05ldGNvb2wlMiBGT01OSWJ1cyUyOHIlMjkvcm9vdD1FbGVtZW50cw==
```

4.1.4 Understanding Session Retention

The Web 2.0 Connect session remains active for the duration specified by the Tomcat configuration. After session expiration, subsequent calls to REST APIs fails. Web 2.0 Connect provides a simple ping capability for maintaining a session.

To perform a session ping using the REST ping mount point, go to:

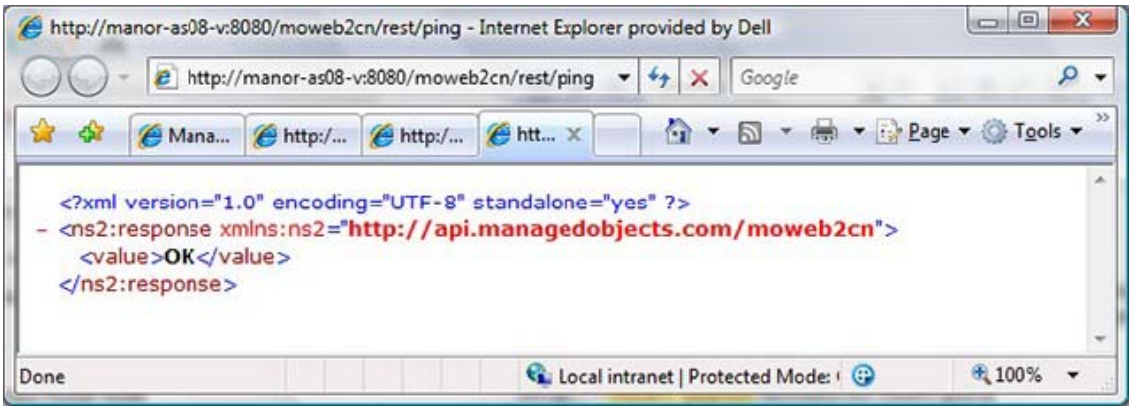
`http://`*`OperationsCenterServer`*`:`*`port`*`/moweb2cn/rest/ping`

where *OperationsCenterServer*:*port* is the server host where the dashboard and Web 2.0 Connect are hosted. The ping mount point does not require any parameters.

For example, where our server is named `manor-as08-v`:

```
http://manor-as08-v:8080/moweb2cn/rest/ping
```

The response is an XML document similar to the following:



If the ping is successful the `<value/>` element contains the string `OK`.

4.2 Using JavaScript and XHTML

Using the REST protocol it is possible to access Operations Center data using JavaScript. The data returns as an XML document, which can be easily manipulated by an XML parser. The sample code provided in this section is intended to convey the simplicity of using the REST API and not intended as a foundation for building a solution.

- Section 4.2.1, “Creating a Session,” on page 25
- Section 4.2.2, “Reading an Element,” on page 26
- Section 4.2.3, “Encoding an Element DName,” on page 28
- Section 4.2.4, “Configuring Session Retention,” on page 29
- Section 4.2.5, “Performing an Operation,” on page 31
- Section 4.2.6, “Closing a Session,” on page 34

4.2.1 Creating a Session

To access data from Operations Center, a session must be initiated first with a user name and password. The user name must be a valid account in Operations Center.

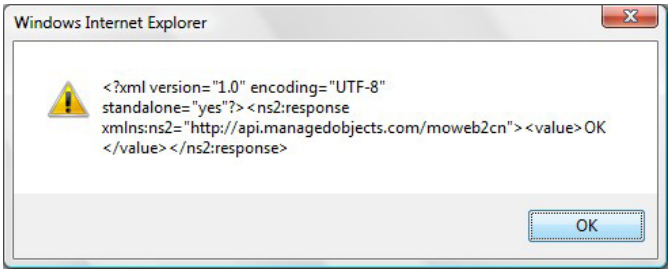
Use the following JavaScript code sample, which illustrates session creation using JavaScript and XML HTTP Request, to create a session:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Web 2.0 Connect</title>
  </head>
  <body>
    <script language="javascript">
     // Web 2.0 Connect host
    var NETIQ_OPERATIONS_CTR_DASHBOARD_HOST = "manor-as08-v";
    var NETIQ_OPERATIONS_CTR_DASHBOARD_PORT = 8080;
    function getWeb2ConnectRootUrl()
    {
      return "http://" + NETIQ_OPERATIONS_CTR_DASHBOARD_HOST  + ":" +
NETIQ_OPERATIONS_CTR_DASHBOARD_PORT + "/moweb2cn/";
    }
     function createWeb2ConnectSession(username, password)
    {
      var url = getWeb2ConnectRootUrl() + "rest/session/?username=" + username +
"&password=" + password;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.open("GET", url, true);
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          if (xhr.responseText)
          {
            alert(xhr.responseText);
          }
        }
      };
      xhr.send();
    }
    createWeb2ConnectSession("admin", "formula");
     </script>
  </body>
</html>
```

NOTE: If running in Firefox 3.0, supply a null parameter to `xhr.send()`, as follows:

```
xhr.send(null)
```

When the Web page is loaded, the `createWeb2ConnectSession()` function is invoked. If the session creation is successful a message box displays the XML responses and the `<value/>` element within the XML response contains the string `OK`:



After a session is created, subsequent attempts to create a session before the session expires, result in an error.

4.2.2 Reading an Element

After a session is successfully created, other REST APIs can be used. An element can be read using the element/value mount point.

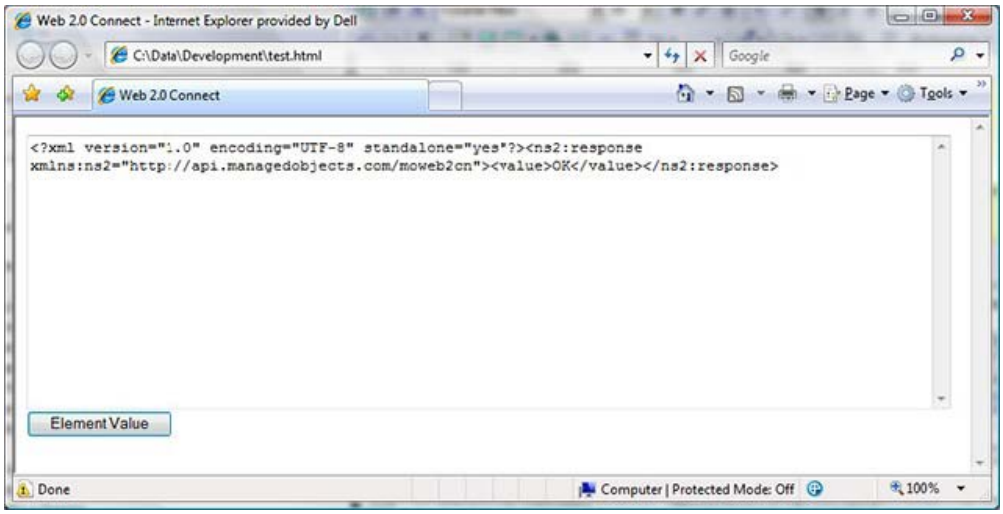
To read an element:

1 Use the following JavaScript code sample, which illustrates retrieving element information using JavaScript, to read an element:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Web 2.0 Connect</title>
  </head>
  <body>
    <script language="javascript">
    // Web 2.0 Connect host
    var NETIQ_OPERATIONS_CTR_DASHBOARD_HOST = "manor-as08-v";
    var NETIQ_OPERATIONS_CTR_DASHBOARD_PORT = 8080;
    var SESSION_ACTIVE = false;
    function getWeb2ConnectRootUrl()
    {
      return "http://" + NETIQ_OPERATIONS_CTR_DASHBOARD_HOST  + ":" +
NETIQ_OPERATIONS_CTR_DASHBOARD_PORT + "/moweb2cn/";
    }
    function createWeb2ConnectSession(username, password)
    {
      var url = getWeb2ConnectRootUrl() + "rest/session/?username=" + username
+ "&password=" + password;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          SESSION_ACTIVE = true;
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = xhr.responseText;
        }
      };
      xhr.open("GET", url, true);
```

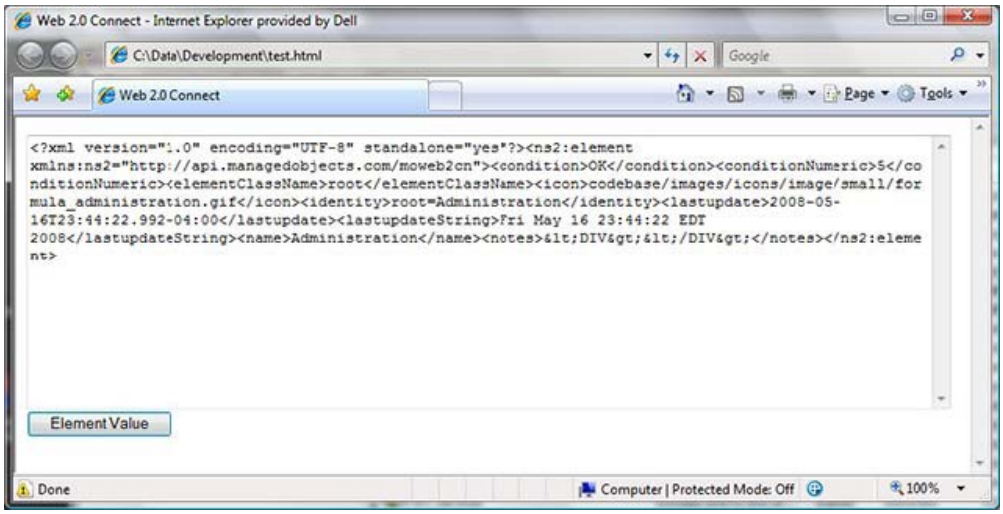
```
      xhr.send();
    }
     function getWeb2ConnectElementValue(identity)
    {
      var url = getWeb2ConnectRootUrl() + "rest/element/value/" + identity;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          SESSION_ACTIVE = true;
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = xhr.responseText;
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
     createWeb2ConnectSession("admin", "formula");
    </script>
    <textarea cols="100" rows="15" id="debug"></textarea>
    </br>
    <input type="button"
onclick="getWeb2ConnectElementValue('root=Administration')" value="Element
Value"/>
  </body>
</html>
```

When the Web page is loaded a Web 2.0 Connect session is created by the `createWeb2ConnectSession()` function and the text area display the result of the session creation operation:



2 After a session is successfully created, click *Element Value* to invoke the `getWeb2ConnectElementValue()` function that performs a REST element/value query for the element having the DName root=Administration.

The text area in the HTML page populates with the XML response for the element value query:



4.2.3 Encoding an Element DName

When using Web 2.0 Connect, it is often necessary to encode the element's dName in base64. The following example shows how to encode the dname using JavaScript; which is basically a JavaScript translation of the encode utility that is provided in the Operations Center `mocommon.jar` library:

```
var Base64

    // private property

    _keyStr :    "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz./
",
            tob64 : function ( buffer) {
            var notleading = false;
            var len = buffer.length;
            var pos = len % 3;
            var c;
            var b0 = 0;
            var b1 = 0;
            var b2 = 0;
            var sb = "";
            if ( pos == 1)
            {
                b2 = buffer.charCodeAt(0);
            }
            else if (pos == 2)
            {
                b1 = buffer.charCodeAt(0);
                b2 = buffer.charCodeAt(1);
            }
            while ( true )
            {
                c = (b0 & 0xfc) >>> 2;
                if(notleading || c != 0) {
                    sb = sb + this._keyStr.charAt(c);
                    notleading = true;
                }
                c = ((b0 & 3) << 4) | ((b1 & 0xf0) >>> 4);
                if(notleading || c != 0) {
                    sb = sb + this._keyStr.charAt(c);
```

```
                    notleading = true;
                }
                c = ((b1 & 0xf) << 2) | ((b2 & 0xc0) >>> 6);

                if(notleading || c != 0) {
                    sb = sb + this._keyStr.charAt(c);
                    notleading = true;
                }
                c = b2 & 0x3f;
                if(notleading || c != 0) {
                    sb = sb + this._keyStr.charAt(c);
                    notleading = true;
                }
                if(pos >= len)
                    break;
                else
                    try {
                        b0 = buffer.charCodeAt(pos++);
                        b1 = buffer.charCodeAt(pos++);
                        b2 = buffer.charCodeAt(pos++);
                    } catch(Exception) { break; }
            }
            if(notleading)
                return sb;
            else
                return "0";
            ;

       }

       }
alert(Base64.tob64('identity:dname:' + 'org=%2F+root/desktop=Under+System/
org=Another+Example/root=Organizations')
 );
```

For more information about base64 encoding of element DNames, see Section 4.1.2, "Reading an Element," on page 22.

4.2.4 Configuring Session Retention

The Web 2.0 Connect session remains active for the duration specified by the Tomcat configuration. After session expiration, subsequent calls to REST APIs fails. Web 2.0 Connect provides a simple ping capability for maintaining a session.

To configure session retention:

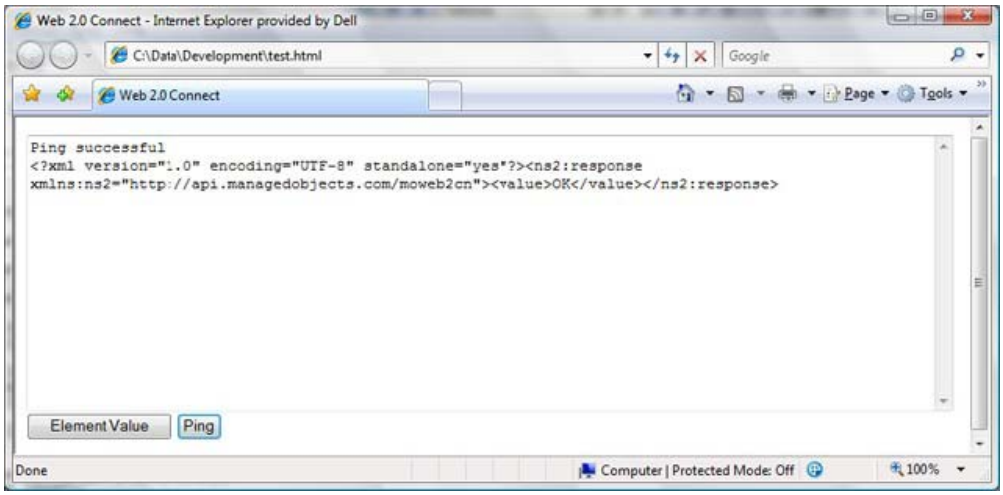
1 Use the following JavaScript code sample, which illustrates performing a REST ping using JavaScript, to configure session retention:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Web 2.0 Connect</title>
  </head>
  <body>
    <script language="javascript">
    // Web 2.0 Connect host
    var NETIQ_OPERATIONS_CTR_DASHBOARD_HOST = "manor-as08-v";
    var NETIQ_OPERATIONS_CTR_DASHBOARD_PORT = 8080;
    function getWeb2ConnectRootUrl()
    {
      return "http://" + NETIQ_OPERATIONS_CTR_DASHBOARD_HOST  + ":" +
NETIQ_OPERATIONS_CTR_DASHBOARD_PORT + "/moweb2cn/";
    }
    function createWeb2ConnectSession(username, password)
```

```
    {
      var url = getWeb2ConnectRootUrl() + "rest/session/?username=" + username
+ "&password=" + password;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = "Session creation failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
    function getWeb2ConnectElementValue(identity)
    {
      var url = getWeb2ConnectRootUrl() + "rest/element/value/" + identity;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = "Element/Value failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
     function doWeb2ConnectPing()
    {
      var url = getWeb2ConnectRootUrl() + "rest/ping";
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = "Ping failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
     createWeb2ConnectSession("admin", "formula");
    </script>
    <textarea cols="100" rows="15" id="debug"></textarea>
    </br>
    <input type="button"
onclick="getWeb2ConnectElementValue('root=Administration')" value="Element
Value"/>
  </body>
</html>
```

2 After a session is successfully created, click *Ping* to invoke the `doWeb2ConnectPing()` function that performs a REST ping.

The text area in the HTML page populates with the XML response for the ping query:



4.2.5 Performing an Operation

Invoking an operation requires a remote call rest/element/perform method. This method requires an identity to be passed in the URL, as well as the name of the operation to invoke:

`http://`*`OperationsCenterServer`*`:`*`port`*`/moweb2cn/rest/element/perform/`*`identity`*`?operationName=`*`operation_name`*

The following code extends the previous JavaScript and HTML examples to demonstrate invoking an operation:

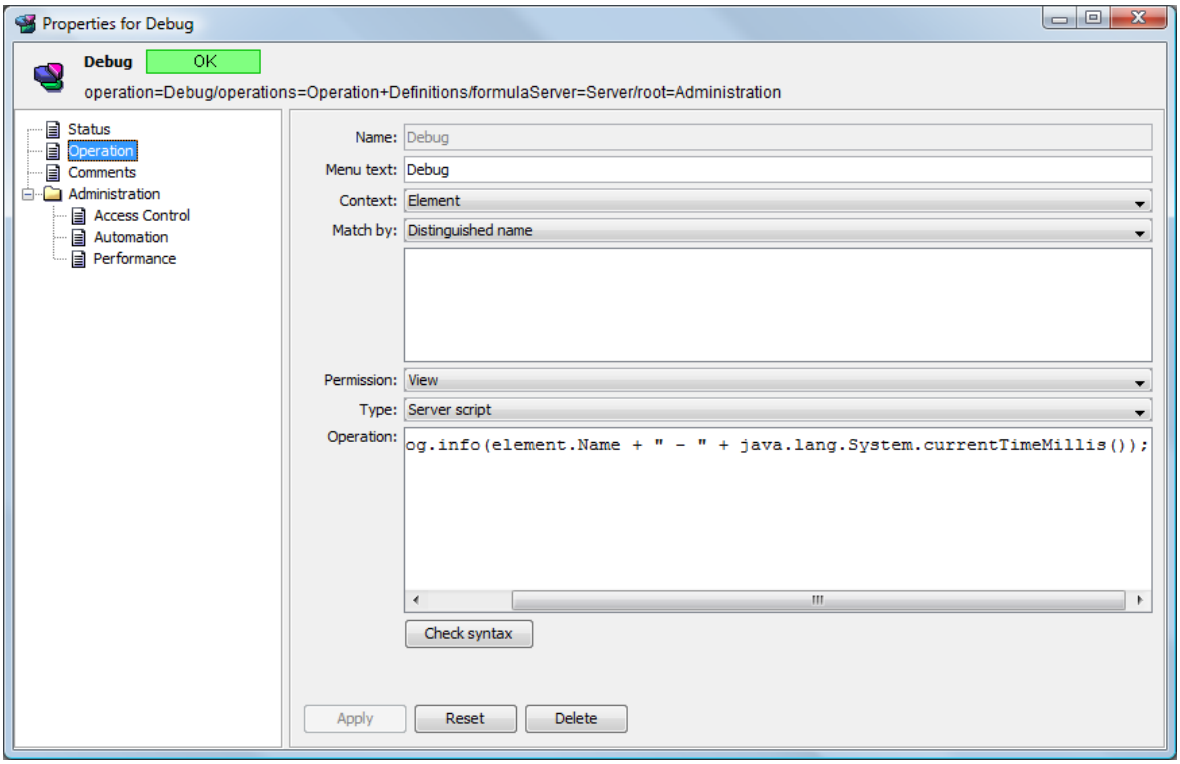
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "htstp://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Web 2.0 Connect</title>
    <script language="javascript">
    // Web 2.0 Connect host
    var NETIQ_OPERATIONS_CTR_DASHBOARD_HOST = "manor-as08-v";
    var NETIQ_OPERATIONS_CTR_DASHBOARD_PORT = 8080;
    function getWeb2ConnectRootUrl()
    {
      return "http://" + NETIQ_OPERATIONS_CTR_DASHBOARD_HOST  + ":" +
NETIQ_OPERATIONS_CTR_DASHBOARD_PORT + "/moweb2cn/";
    }
    function createWeb2ConnectSession(username, password)
    {
      var url = getWeb2ConnectRootUrl() + "rest/session?username=" + username +
"&password=" + password;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = "Session created\n" + xhr.responseText;
        }

        else
        {
          debug.innerText = "Session creation failed";
```

```
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
    function closeWeb2ConnectSession()
    {
      var url = getWeb2ConnectRootUrl() + "rest/session";
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = "Session terminated\n" + xhr.responseText;
        }
        else
        {
          debug.innerText = "Session termination failed";
        }
      };
      xhr.open("DELETE", url, true);
      xhr.send();
    }
    function getWeb2ConnectElementValue(identity)
    {
      var url = getWeb2ConnectRootUrl() + "rest/element/value/" + identity;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = xhr.responseText;
        }
        else
        {
          debug.innerText = "Element/Value failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
    function doWeb2ConnectPing()
    {
      var url = getWeb2ConnectRootUrl() + "rest/ping";
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = "Ping successful\n" + xhr.responseText;
        }
        else
        {
          debug.innerText = "Ping failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
     function doWeb2ConnectPerformOperation()
    {
      var url = getWeb2ConnectRootUrl() + "rest/element/perform/
root=Administration?operationName=Debug";
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.onreadystatechange = function()
      {
```

```
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = "Operation successful\n" + xhr.responseText;
        }
        else
        {
          debug.innerText = "Operation failed";
        }
      };
      xhr.open("GET", url, true);
      xhr.send();
    }
     function pageLoad()
    {
      createWeb2ConnectSession("admin", "formula");
    }
    </script>
  </head>
  <body onload="pageLoad()">
    <textarea cols="100" rows="15" id="debug"></textarea>
    </br>
    <input type="button"
onclick="getWeb2ConnectElementValue('root=Administration')" value="Element Value"/
>
    <input type="button" onclick="doWeb2ConnectPing()" value="Ping"/>
    <input type="button" onclick="doWeb2ConnectPerformOperation()"
value="Operation"/>
    <input type="button" onclick="closeWeb2ConnectSession()" value="End Session"/>
  </body>
</html>
```

This example requires an operation named Debug to have been defined in Operations Center, as follows:



The operation is server side with element context. The operation simply writes the name of the element it was invoked against to the Operations Center trace log using the following script:

```
formula.log.info(element.Name + " - " + java.lang.System.currentTimeMillis());
```

4.2.6 Closing a Session

Closing a session using JavaScript using XML HTTP Request simply requires a remote call to the rest/session method with an HTTP DELETE method:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Web 2.0 Connect</title>
    <script language="javascript">
    // Web 2.0 Connect host
    var NETIQ_OPERATIONS_CTR_DASHBOARD_HOST = "manor-as08-v";
    var NETIQ_OPERATIONS_CTR_DASHBOARD_PORT = 8080;
    function getWeb2ConnectRootUrl()
    {
      return "http://" + NETIQ_OPERATIONS_CTR_DASHBOARD_HOST  + ":" +
NETIQ_OPERATIONS_CTR_DASHBOARD_PORT + "/moweb2cn/";
    }
    function createWeb2ConnectSession(username, password)
    {
      var url = getWeb2ConnectRootUrl() + "rest/session/?username=" + username +
"&password=" + password;
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
      xhr.open("GET", url, true);
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          if (xhr.responseText)
          {
            alert(xhr.responseText);
          }
        }
      };
      xhr.send();
    }
     function closeWeb2ConnectSession()
    {
      var url = getWeb2ConnectRootUrl() + "rest/session";
      var xhr = window.XMLHttpRequest ? new XMLHttpRequest() : new
ActiveXObject("MSXML2.XMLHTTP.3.0");
       xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4 && xhr.status == 200)
        {
          debug.innerText = "Session terminated\n" + xhr.responseText;
        }
        else
        {
          debug.innerText = "Session termination failed";
        }
      };
      xhr.open("DELETE", url, true);
```

```
      xhr.send();
    }
    function pageLoad()
    {
      createWeb2ConnectSession("admin", "formula");
    }
     </script>
  </head>
   <body onload="pageLoad()">
    <textarea cols="100" rows="15" id="debug"></textarea>
    </br>
    <input type="button" onclick="closeWeb2ConnectSession()" value="End Session"/>
   </body>
</html>
```

4.3 ECMA/JavaScript Library

The JavaScript library provides encapsulated access to the REST APIs that are commonly needed. The following methods are available:

- createSession
- deleteSession
- ping
- readElementValue
- readElementAttributes
- readElementChildren
- readElementAlarmsRealtime
- performElementOperation

To learn more about the JavaScript library, see the following sections:

- Section 4.3.1, “Example Usage,” on page 35
- Section 4.3.2, “Library Source,” on page 36
- Section 4.3.3, “Methods and Properties,” on page 54

4.3.1 Example Usage

```
<script type="text/ecmascript" src="scripts/1.0/bsm.web2connect.js"></script>

<script language="text/ecmascript">
  var w = new Web2Connect();
  w.Session.Protocol = "http";
  w.Session.Host = "localhost";
  w.Session.Port = "80";
  w.Session.Username = "admin";
  w.Session.Password = "formula";
  w.createSession(logonSuccess, logonFail);
  function logonSuccess()
  {
    alert("yippee!");
    w.deleteSession(null, null);
  }
  function logonFail()
  {
    alert("boo hoo");
  }
</script>
```

All methods must be passed references to handler functions for success and fail asynchronous callbacks. Null references can be used if callbacks are not required.

4.3.2 Library Source

```
/* Web 2.0 Connect Lib  */

// Namespaces

if (!BSM)
{
var BSM = { __bsm : this, __namespace : true };
}

if (!BSM.Utils)
{
BSM.Utils = { __utils : this, __namespace :true };
}

/*

BSM utility class

*/

BSM.Utils.ColorSettings =
{
BackgroundColors :
{
  UNKNOWN       : "#808080",
  BUSY          : "#43CFED",
  ACTIVE        : "#43CFED",
  UNMANAGED     : "#BAA860",
  IDLE          : "#80FF80",
  INITIAL       : "#BAA860",
  CRITICAL      : "#ff4040",
  MAJOR         : "#FEA342",
  MINOR         : "#F3F171",
  INFORMATIONAL : "#43CFED",
  OK            : "#80FF80"
},

ForegroundColors :
{
  UNKNOWN       : "#FFFFFF",
  BUSY          : "#000000",
  ACTIVE        : "#000000",
  UNMANAGED     : "#FFFFFF",
  IDLE          : "#000000",
  INITIAL       : "#FFFFFF",
  CRITICAL      : "#FFFFFF",
  MAJOR         : "#000000",
  MINOR         : "#000000",
  INFORMATIONAL : "#FFFFFF",
  OK            : "#000000"
}
};

/*

Web2Connect API manager class

*/

BSM.Web2Connect = function()
{
_w2c = this;
```

```
// Constants

this._W2C_EXTERNAL_ROOT_CONTEXT = "/moweb2cn/";
this._W2C_INTERNAL_ROOT_CONTEXT = "/ManagedObjectsPortlets/";

// Namespaces

this.Data = {};
this.Data.JSON = {};

this.Utils = {};
this.Interface = {};

// Enums

this.HttpProtocol =
{
  HTTP: "http",
  HTTPS: "https"
};

this.HttpMethod =
{
  GET: "GET",
  POST: "POST",
  UPDATE: "UPDATE",
  DELETE: "DELETE"
};

this.HttpResponseCode =
{
  SUCCESS: 200,
  NOT_AUTHORIZED: 403,
  NOT_FOUND: 404,
  SERVER_ERROR: 500
};

this.XmlHttpRequestState =
{
  STARTED: 1,
  PROCESSING: 2,
  COMPLETE: 4
};

this.RootContext =
{
  EXTERNAL_ROOT_CONTEXT: _w2c._W2C_EXTERNAL_ROOT_CONTEXT,
  INTERNAL_ROOT_CONTEXT: _w2c._W2C_INTERNAL_ROOT_CONTEXT
};

this.ApiFormat =
{
  JSON: "json",
  REST: "rest",
  XML: "xml"
};

this.ApiMethodUri =
{
  SESSION: "session",
  USER: "user",
  PING: "ping",
  COLOR_SETTINGS: "colorSettings",
  ELEMENT_VALUE: "element/value/",
  ELEMENT_ATTRIBUTES: "element/attributes/",
  ELEMENT_CHILDREN: "element/children/",
  ELEMENT_CHILDREN_ATTRIBUTES: "element/childrenAttributes/",
  ELEMENT_RELATIONSHIPS: "element/relationships/",
  ELEMENT_RELATED: "element/related/",
```

```
  ELEMENT_PERFORM: "element/perform/",
  ELEMENT_ALARMS_REALTIME: "element/alarms/{CHANNEL}/realtime/",
  ELEMENT_QUERY_PERFORMANCE_SERIES: "element/query/performance/{PROFILE}/
{EXPRESSION}/series/",
  ELEMENT_QUERY_PERFORMANCE: "element/query/performance/"
};

this.ApiMethodParameter =
{
  USERNAME: "username",
  PASSWORD: "password",
  ATTRIBUTE_NAMES: "attributeNames",
  OPERATION_NAME: "operationName",
  LIMIT: "limit",
  FROM: "from",
  TO: "to"
};

this.AlarmChannel =
{
  ALL: "ALL",
  ANNOTATION: "ANNOTATION",
  AUDIT: "AUDIT",
  CHANGE: "CHANGE",
  GENERAL: "GENERAL",
  HISTORY: "HISTORY",
  SLA: "SLA",
  SLA_METRIC: "SLA_METRIC",
  OUTAGE: "OUTAGE"
};

this.ChildType =
{
  BOTH: "Both",
  LINKED: "Linked",
  STORED: "Stored"
};

this.AttributeType =
{
  STRING_LIST: "StringListType",
  STRING: "StringType",
  BOOLEAN: "BooleanType",
  INTEGER: "IntegerType",
  DOUBLE: "DoubleType",
  FLOAT: "FloatType",
  LONG: "LongType",
  DATE: "DateType",
  NULL: "NullType",
  UNKNOWN: "UnknownType"
};

this.Synchronicity =
{
  ASYNCHRONOUS: true,
  SYNCHRONOUS: false
};

this.Data.JSON.Constants =
{
  ALARMS: "moweb2cn.alarms",
  ELEMENTS: "moweb2cn.elements",
  ATTRIBUTES: "moweb2cn.attributes",
  PERFORMANCE_INFO: "moweb2cn.performanceInfo",
  PERFORMANCE_SERIES: "moweb2cn.performanceSeries"
};

/*

API methods
```

```
*/

// Create session

this.createSession = function(successHandler, faultHandler)
{
  var _createSession = {};

  _createSession._sh = successHandler;
  _createSession._fh = faultHandler;

  _createSession.createSessionStateHandler = function(response)
  {
  if (_w2c.Session.getStatus() == _w2c.Session.State.READY)
  {
    _createSession._sh(response);
  }
  else
  {
    _createSession._fh(response);
  }
  };

  _createSession.url = "";
  _createSession.url += _w2c.ApiMethodUri.SESSION;
  _createSession.url += "?" + _w2c.ApiMethodParameter.USERNAME + "=" +
_w2c.Session.Username;
  _createSession.url += "&" + _w2c.ApiMethodParameter.PASSWORD + "=" +
_w2c.Session.Password;

  _createSession.request = new _w2c._remoteServiceRequest();
  _createSession.request.method = _w2c.HttpMethod.GET;
  _createSession.request.handleSuccess = _createSession.createSessionStateHandler;
  _createSession.request.handleFault = _createSession.createSessionStateHandler;
  _createSession.request.setUrl(_createSession.url);
  _createSession.request.execute(_w2c.Synchronicity.SYNCHRONOUS);
};

// Delete session

this.deleteSession = function(successHandler, faultHandler)
{
  var _deleteSession = {};

  _deleteSession.url = "";
  _deleteSession.url += _w2c.ApiMethodUri.SESSION;

  _deleteSession.request = new _w2c._remoteServiceRequest();
  _deleteSession.request.method = _w2c.HttpMethod.DELETE;
  _deleteSession.request.handleSuccess = successHandler;
  _deleteSession.request.handleFault = faultHandler;
  _deleteSession.request.setUrl(_deleteSession.url);
  _deleteSession.request.execute();
};

// Element value

this.readElementValue = function(elementIdentity, successHandler, faultHandler)
{
  var _readElementValue = {};

  _readElementValue.url = "";
  _readElementValue.url += _w2c.ApiMethodUri.ELEMENT_VALUE;
  _readElementValue.url += elementIdentity;

  _readElementValue.request = new _w2c._remoteServiceRequest();
  _readElementValue.request.method = _w2c.HttpMethod.GET;
  _readElementValue.request.handleSuccess = successHandler;
  _readElementValue.request.handleFault = faultHandler;
```

```
  _readElementValue.request.setUrl(_readElementValue.url);
  _readElementValue.request.execute();
};

// Element attributes

this.readElementAttributes = function(elementIdentity, attributeNames,
successHandler, faultHandler)
{
  var _readElementAttributes = {};

  _readElementAttributes.url = "";
  _readElementAttributes.url += _w2c.ApiMethodUri.ELEMENT_ATTRIBUTES;
  _readElementAttributes.url += elementIdentity;
  _readElementAttributes.url += "?" + _w2c.ApiMethodParameter.ATTRIBUTE_NAMES + "="
+ attributeNames.getDelimitedList();

  _readElementAttributes.request = new _w2c._remoteServiceRequest();
  _readElementAttributes.request.method = _w2c.HttpMethod.GET;
  _readElementAttributes.request.handleSuccess = successHandler;
  _readElementAttributes.request.handleFault = faultHandler;
  _readElementAttributes.request.setUrl(_readElementAttributes.url);
  _readElementAttributes.request.execute();
};

// Element attribute variant (singule attribute)

this.readElementAttribute = function(elementIdentity, attributeName,
successHandler, faultHandler, synchronicity)
{
  var _readElementAttribute = {};

  _readElementAttribute.url = "";
  _readElementAttribute.url += _w2c.ApiMethodUri.ELEMENT_ATTRIBUTES;
  _readElementAttribute.url += elementIdentity;
  _readElementAttribute.url += "?" + _w2c.ApiMethodParameter.ATTRIBUTE_NAMES + "="
+ attributeName;

  _readElementAttribute.request = new _w2c._remoteServiceRequest();
  _readElementAttribute.request.method = _w2c.HttpMethod.GET;
  _readElementAttribute.request.handleSuccess = successHandler;
  _readElementAttribute.request.handleFault = faultHandler;
  _readElementAttribute.request.setUrl(_readElementAttribute.url);
  _readElementAttribute.request.execute(synchronicity);
};

// Element children

this.readElementChildren = function(elementIdentity, successHandler, faultHandler)
{
  var _readElementChildren = {};

  _readElementChildren._responseFormat = _w2c.Session.ApiFormat;
  _readElementChildren._successHandlerCallback = successHandler;
  _readElementChildren._faultHandlerCallback = faultHandler;

  _readElementChildren._successHandler = function(response)
  {
  var elements = null;

  try
  {
    if (_readElementChildren._responseFormat == _w2c.ApiFormat.JSON)
    {
    elements = _w2c.Data.JSON.parseElements(response);
    }
  }
  catch (ex)
  {
    response.exception = ex;
```

```
    _readElementChildren._faultHandlerCallback(response);
  }

  _readElementChildren._successHandlerCallback(response, elements);
  };

  _readElementChildren.url = "";
  _readElementChildren.url += _w2c.ApiMethodUri.ELEMENT_CHILDREN;
  _readElementChildren.url += elementIdentity;

  _readElementChildren.request = new _w2c._remoteServiceRequest();
  _readElementChildren.request.method = _w2c.HttpMethod.GET;
  _readElementChildren.request.handleSuccess =
_readElementChildren._successHandler;
  _readElementChildren.request.handleFault = faultHandler;
  _readElementChildren.request.setUrl(_readElementChildren.url);
  _readElementChildren.request.execute();
};

// Element children attributes

this.readElementChildrenAttributes = function(elementIdentity, attributeNames,
successHandler, faultHandler)
{
  var _readElementChildrenAttributes = {};

  _readElementChildrenAttributes.url = "";
  _readElementChildrenAttributes.url +=
_w2c.ApiMethodUri.ELEMENT_CHILDREN_ATTRIBUTES;
  _readElementChildrenAttributes.url += elementIdentity;
  _readElementChildrenAttributes.url += "?" +
_w2c.ApiMethodParameter.ATTRIBUTE_NAMES + "=" + attributeNames.getDelimitedList();

  _readElementChildrenAttributes.request = new _w2c._remoteServiceRequest();
  _readElementChildrenAttributes.request.method = _w2c.HttpMethod.GET;
  _readElementChildrenAttributes.request.handleSuccess = successHandler;
  _readElementChildrenAttributes.request.handleFault = faultHandler;

_readElementChildrenAttributes.request.setUrl(_readElementChildrenAttributes.url);
  _readElementChildrenAttributes.request.execute();
};

// Element relationships

this.readElementRelationships = function(elementIdentity, successHandler,
faultHandler)
{
  _readElementRelationships = {};

  _readElementRelationships.url = "";
  _readElementRelationships.url += _w2c.ApiMethodUri.ELEMENT_RELATIONSHIPS;
  _readElementRelationships.url += elementIdentity;

  _readElementRelationships.request = new _w2c._remoteServiceRequest();
  _readElementRelationships.request.method = _w2c.HttpMethod.GET;
  _readElementRelationships.request.handleSuccess = successHandler;
  _readElementRelationships.request.handleFault = faultHandler;
  _readElementRelationships.request.setUrl(_readElementRelationships.url);
  _readElementRelationships.request.execute();
};

// Element related

this.readElementRelated = function(elementIdentity, successHandler, faultHandler)
{
  var _readElementRelated = {};

  _readElementRelated.url = "";
  _readElementRelated.url += _w2c.ApiMethodUri.ELEMENT_RELATED;
  _readElementRelated.url += elementIdentity;
```

```
  _readElementRelated.request = new _w2c._remoteServiceRequest();
  _readElementRelated.request.method = _w2c.HttpMethod.GET;
  _readElementRelated.request.handleSuccess = successHandler;
  _readElementRelated.request.handleFault = faultHandler;
  _readElementRelated.request.setUrl(_readElementRelated.url);
  _readElementRelated.request.execute();
};

// Ping

this.ping = function(successHandler, faultHandler, synchronicity)
{
  if (typeof synchronicity == "undefined")
  {
  synchronicity = _w2c.Synchronicity.ASYNCHRONOUS;
  }

  var _ping = {};

  _ping.url = "";
  _ping.url += _w2c.ApiMethodUri.PING;

  _ping.request = new _w2c._remoteServiceRequest();
  _ping.request.method = _w2c.HttpMethod.GET;
  _ping.request.handleSuccess = successHandler;
  _ping.request.handleFault = faultHandler;
  _ping.request.setUrl(_ping.url);
  _ping.request.execute(synchronicity);
};

// User

this.readUser = function(successHandler, faultHandler, synchronicity)
{
  if (typeof synchronicity == "undefined")
  {
  synchronicity = _w2c.Synchronicity.ASYNCHRONOUS;
  }

  _user = {};

  _user.url = "";
  _user.url += _w2c.ApiMethodUri.USER;

  _user.request = new _w2c._remoteServiceRequest();
  _user.request.method = _w2c.HttpMethod.GET;
  _user.request.handleSuccess = successHandler;
  _user.request.handleFault = faultHandler;
  _user.request.setUrl(_user.url);
  _user.request.execute(synchronicity);
};

// Color settings

this.readColorSettings = function(successHandler, faultHandler)
{
  var _colorSettings = {};

  _colorSettings.url = "";
  _colorSettings.url += _w2c.ApiMethodUri.COLOR_SETTINGS;

  _colorSettings.request = new _w2c._remoteServiceRequest();
  _colorSettings.request.method = _w2c.HttpMethod.GET;
  _colorSettings.request.handleSuccess = successHandler;
  _colorSettings.request.handleFault = faultHandler;
  _colorSettings.request.setUrl(_colorSettings.url);
  _colorSettings.request.execute(synchronicity);
};
```

```
// Perform element operation

this.performElementOperation = function(elementIdentity, operationName,
successHandler, faultHandler)
{
  var _peo = {};

  _peo.url = "";
  _peo.url += _w2c.ApiMethodUri.ELEMENT_PERFORM;
  _peo.url += elementIdentity;
  _peo.url += "?" + _w2c.ApiMethodParameter.OPERATION_NAME + "=" + operationName;

  _peo.request = new _w2c._remoteServiceRequest();
  _peo.request.method = _w2c.HttpMethod.GET;
  _peo.request.handleSuccess = successHandler;
  _peo.request.handleFault = faultHandler;
  _peo.request.setUrl(_peo.url);
  _peo.request.execute();
};

// Read element alarms

this.readElementAlarmsRealtime = function(elementIdentity, channel, params,
successHandler, faultHandler)
{
  var _readElementAlarms = {};

  _readElementAlarms._responseFormat = _w2c.Session.ApiFormat;
  _readElementAlarms._successHandlerCallback = successHandler;
  _readElementAlarms._faultHandlerCallback = faultHandler;

  _readElementAlarms._successHandler = function(response)
  {
  var alarms = null;

  try
  {
    if (_readElementAlarms._responseFormat == _w2c.ApiFormat.JSON)
    {
    alarms = _w2c.Data.JSON.parseAlarms(response);
    }

    _readElementAlarms._successHandlerCallback(response, alarms);
  }
  catch (ex)
  {
    response.exception = ex;
    _readElementAlarms._faultHandlerCallback(response);
  }
  };

  _readElementAlarms.url = "";
  _readElementAlarms.url += _w2c.ApiMethodUri.ELEMENT_ALARMS_REALTIME.replace(/
\{CHANNEL\}/, channel);
  _readElementAlarms.url += elementIdentity;

  _readElementAlarms.request = new _w2c._remoteServiceRequest();
  _readElementAlarms.request.method = _w2c.HttpMethod.GET;
  _readElementAlarms.request.handleSuccess = _readElementAlarms._successHandler;
  _readElementAlarms.request.handleFault = faultHandler;
  _readElementAlarms.request.setUrl(_readElementAlarms.url);
  _readElementAlarms.request.execute();
};

// Read element performance information

this.readElementPerformanceInfo = function(elementIdentity, successHandler,
faultHandler)
{
  var readElementPerformanceInfo = {};
```

```
  readElementPerformanceInfo.url = "";
  readElementPerformanceInfo.url += _w2c.ApiMethodUri.ELEMENT_QUERY_PERFORMANCE;
  readElementPerformanceInfo.url += elementIdentity;

  readElementPerformanceInfo.request = new _w2c._remoteServiceRequest();
  readElementPerformanceInfo.request.method = _w2c.HttpMethod.GET;
  readElementPerformanceInfo.request.handleSuccess = successHandler;
  readElementPerformanceInfo.request.handleFault = faultHandler;
  readElementPerformanceInfo.request.setUrl(readElementPerformanceInfo.url);
  readElementPerformanceInfo.request.execute();
};

// Read element performance series

this.readElementPerformanceSeries = function(elementIdentity,
performanceSeriesQuery, successHandler, faultHandler)
{
  var _readElementPerformanceSeries = {};

  _readElementPerformanceSeries._responseFormat = _w2c.Session.ApiFormat;
  _readElementPerformanceSeries._successHandlerCallback = successHandler;
  _readElementPerformanceSeries._faultHandlerCallback = faultHandler;

  _readElementPerformanceSeries._successHandler = function(response)
  {
  var series = null;

  try
  {
    if (_readElementPerformanceSeries._responseFormat == _w2c.ApiFormat.JSON)
    {
    series = _w2c.Data.JSON.parsePerformanceSeries(response);
    }

    _readElementPerformanceSeries._successHandlerCallback(response, series);
  }
  catch (ex)
  {
    response.exception = ex;
    _readElementPerformanceSeries._faultHandlerCallback(response);
  }
  };

  var uriQueryString = _w2c.ApiMethodUri.ELEMENT_QUERY_PERFORMANCE_SERIES;

  uriQueryString = uriQueryString.replace(/\{PROFILE\}/,
performanceSeriesQuery.profileName);
  uriQueryString = uriQueryString.replace(/\{EXPRESSION\}/,
performanceSeriesQuery.expressionName);

  _readElementPerformanceSeries.url = "";
  _readElementPerformanceSeries.url += uriQueryString;
  _readElementPerformanceSeries.url += elementIdentity;

  _readElementPerformanceSeries.url += "?from=" +
performanceSeriesQuery.fromTimeEpoch;
  _readElementPerformanceSeries.url += "&to=" +
performanceSeriesQuery.toTimeEpoch;
  _readElementPerformanceSeries.url += "&limit=" +
performanceSeriesQuery.limitCount;

  _readElementPerformanceSeries.request = new _w2c._remoteServiceRequest();
  _readElementPerformanceSeries.request.method = _w2c.HttpMethod.GET;
  _readElementPerformanceSeries.request.handleSuccess =
_readElementPerformanceSeries._successHandler;
  _readElementPerformanceSeries.request.handleFault = faultHandler;
  _readElementPerformanceSeries.request.setUrl(_readElementPerformanceSeries.url);
  _readElementPerformanceSeries.request.execute();
};
```

```
// Data classes

this.Data.ElementItem = function()
{
  _item = this;
  _item.identity = null;
  _item.name = null;
  _item.condition = null;
  _item.conditionNumeric = null;
  _item.elementClassName = null;
  _item.icon = null;
  _item.notes = null;
  _item.lastUpdate = 0;
};

this.Data.ElementCollection = function()
{
  _collection = this;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;

  _collection.clear = function()
  {
  delete _collection.list;
  delete _collection.indexer;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;
  };

  _collection.getItem = function(index)
  {
  return _collection.list[_collection.indexer[index]];
  };

  _collection.add = function(elementIdentity, elementItem)
  {
  var key = elementIdentity;

  _collection.count++;
  _collection.indexer[_collection.count] = key;
  _collection.list[key] = elementItem;
  };
};  // ElementCollection

this.Data.AlarmItem = function()
{
  _item = this;
  _item.id = 0;
  _item.description = null;
  _item.lastUpdate = 0;
  _item.severityNumeric = null;
  _item.affectedElement = null;
  _item.affectedElementName = null;
};

this.Data.AlarmCollection = function()
{
  _collection = this;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;

  _collection.clear = function()
```

```
  {
  delete _collection.list;
  delete _collection.indexer;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;
  };

  _collection.getItem = function(index)
  {
  return _collection.list[_collection.indexer[index]];
  };

  _collection.add = function(alarmId, alarmItem)
  {
  var key = alarmId;

  _collection.count++;
  _collection.indexer[_collection.count] = key;
  _collection.list[key] = alarmItem;
  };
};  // AlarmCollection

this.Data.PerformanceSeriesItem = function(dateValue, seriesValue,
discontinuityValue)
{
  _item = this;
  _item.dateValue = (typeof dateValue == "undefined") ? 0 : dateValue;
  _item.seriesValue = (typeof seriesValue == "undefined") ? 0 : seriesValue;
  _item.discontinuity = (typeof discontinuityValue == "undefined") ? false :
discontinuityValue == "true";
};

this.Data.PerformanceSeriesCollection = function()
{
  _collection = this;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;
  _collection.profile = "";
  _collection.expression = "";
  _collection.lowestValue = Number.MAX_VALUE;
  _collection.highestValue = Number.MIN_VALUE;
  _collection.earliestDate = Number.MAX_VALUE;
  _collection.latestDate = Number.MIN_VALUE;

  _collection.clear = function()
  {
  delete _collection.list;
  delete _collection.indexer;

  _collection.list = {};
  _collection.indexer = [];
  _collection.count = 0;
  };

  _collection.add = function(dateValue, seriesValue, discontinuityValue)
  {
  var key = dateValue;
  var newItem = new _w2c.Data.PerformanceSeriesItem(dateValue, seriesValue,
discontinuityValue);

  _collection.count++;
  _collection.indexer[_collection.count] = key;
  _collection.list[key] = newItem;

  _collection.earliestDate = Math.min(_collection.earliestDate, dateValue);
  _collection.latestDate = Math.max(_collection.latestDate, dateValue);
```

```
  _collection.lowestValue = Math.min(_collection.lowestValue, seriesValue);
  _collection.highestValue = Math.max(_collection.highestValue, seriesValue);
  };

  _collection.getItem = function(index)
  {
  return _collection.list[_collection.indexer[index]];
  };

  _collection.getItemByDateEpoch = function(dateEpochKey)
  {
  return _collection.list[dateEpochKey];
  };
};  // PerformanceSeriesCollection


// Data JSON helpers

this.Data.JSON.parseAttributeValueType = function(attribute)
{
  var value = attribute.value;

  switch (attribute.type)
  {
  case _w2c.AttributeType.INTEGER:
    value = parseInt(value, 10);
    break;

  case _w2c.AttributeType.FLOAT || _w2c.AttributeType.DOUBLE:
    value = parseFloat(value);
    break;

  default:
  }

  return value;
};

this.Data.JSON.parseElementAttributes = function(json)
{
  var o = eval("(" + json + ")");

  var attributesMap = o[_w2c.Data.JSON.Constants.ATTRIBUTES].attributes;

  var attributes = {};

  for (var p in attributesMap)
  {
  if (attributesMap.hasOwnProperty(p))
  {
    attributes[p] = _w2c.Data.JSON.parseAttributeValueType(attributesMap[p]);
  }
  }

  return attributes;
};

this.Data.JSON.parseAlarms = function(json)
{
  var o = eval("(" + json + ")");

  var _JSON_ = _w2c.Data.JSON.Constants.ALARMS;

  var alarmCollection = new _w2c.Data.AlarmCollection();

  if (o[_JSON_].alarm)
  {
  for (var index = 0; index < o[_JSON_].alarm.length; index++)
  {
```

```
    var alarmItem = new _w2c.Data.AlarmItem();

    alarmItem.id = o[_JSON_].alarm[index].id;
    alarmItem.description = o[_JSON_].alarm[index].Summary ?
o[_JSON_].alarm[index].Summary : "";
    alarmItem.lastUpdate = o[_JSON_].alarm[index].lastUpdate;
    alarmItem.severityNumeric = o[_JSON_].alarm[index].severityNumeric;
    alarmItem.affectedElement = o[_JSON_].alarm[index].affectedElement;
    alarmItem.affectedElementName = o[_JSON_].alarm[index].affectedElementName;

    alarmCollection.add(alarmItem.id, alarmItem);
  }
  }

  return alarmCollection;
};

this.Data.JSON.parseElements = function(json)
{
  var o = eval("(" + json + ")");

  var _JSON_ = _w2c.Data.JSON.Constants.ELEMENTS;

  var elementCollection = new _w2c.Data.ElementCollection();
  var elementItem;

  if (o[_JSON_].element)
  {
  if (typeof o[_JSON_].element.length == "undefined")
  {
    // Single item

    elementItem = new _w2c.Data.ElementItem();

    elementItem.identity = o[_JSON_].element.identity;
    elementItem.name = o[_JSON_].element.name;
    elementItem.condition = o[_JSON_].element.condition;
    elementItem.conditionNumeric = o[_JSON_].element.conditionNumeric;
    elementItem.elementClassName = o[_JSON_].element.elementClassName;
    elementItem.icon = o[_JSON_].element.icon;
    elementItem.notes = o[_JSON_].element.notes;
    elementItem.lastUpdate = o[_JSON_].element.lastUpdateString;

    elementCollection.add(elementItem.identity, elementItem);
  }
  else
  {
    // Multiple items

    for (var index = 0; index < o[_JSON_].element.length; index++)
    {
    elementItem = new _w2c.Data.ElementItem();

    elementItem.identity = o[_JSON_].element[index].identity;
    elementItem.name = o[_JSON_].element[index].name;
    elementItem.condition = o[_JSON_].element[index].condition;
    elementItem.conditionNumeric = o[_JSON_].element[index].conditionNumeric;
    elementItem.elementClassName = o[_JSON_].element[index].elementClassName;
    elementItem.icon = o[_JSON_].element[index].icon;
    elementItem.notes = o[_JSON_].element[index].notes;
    elementItem.lastUpdate = o[_JSON_].element[index].lastUpdateString;

    elementCollection.add(elementItem.identity, elementItem);
    }
  }
  }

  return elementCollection;
};
```

```
this.Data.JSON.parsePerformanceSeries = function(json)
{
  var o = eval("(" + json + ")");

  var _JSON_ = _w2c.Data.JSON.Constants.PERFORMANCE_SERIES;

  var dates = [];
  var series = [];
  var discontinuities = [];

  dates = o[_JSON_].dateValues.split(" ");
  series = o[_JSON_].seriesValues.split(" ");
  discontinuities = o[_JSON_].discontinuityValues.split(" ");

  var seriesCollection = new _w2c.Data.PerformanceSeriesCollection();

  seriesCollection.profile = o[_JSON_].profile;
  seriesCollection.expression = o[_JSON_].expression;

  for (var index = 0; index < dates.length; index++)
  {
  seriesCollection.add(dates[index], series[index], discontinuities[index]);
  }

  return seriesCollection;
};

// Utility functions

this.Utils.PerformanceSeriesQuery = function()
{
  _psq = this;

  this.profileName = "";
  this.expressionName = "";
  this.fromTimeEpoch = 0;
  this.toTimeEpoch = 0;
  this.limitCount = 100;
};

this.Utils.AssociativeList = function()
{
  _eac = this;

  _eac.list = {};

  _eac.clear = function()
  {
  delete _eac.list;
  _eac.list = {};
  };

  _eac.add = function(listItem)
  {
  _eac.list[listItem] = listItem;
  };

  _eac.getDelimitedList = function(delimiter)
  {
  delimiter = (delimiter) ? delimiter : ",";

  var listString = "";

  for (var listItem in _eac.list)
  {
    if (_eac.list.hasOwnProperty(listItem))
    {
    listString += _eac.list[listItem] + delimiter;
    }
  }
```

```
  return listString.substring(0, listString.lastIndexOf(delimiter));
  };
};

// Interface helper methods

this.Interface.getMethodUri = function(apiMethodUri)
{
  return _w2c._getApiRootContext() + apiMethodUri;
};

// XML HTTP request object

this._xmlHttpRequest = function()
{
  if (window.XMLHttpRequest)
  {
  return new XMLHttpRequest();
  }
  else if (window.ActiveXObject)
  {
  return new ActiveXObject("Microsoft.XMLHTTP");
  }
  else
  {
  return null;
  }
};

// Generic wrapper for HTTP requester

this._remoteServiceRequest = function()
{
  _rpc = this;

  _rpc.xhr = _w2c._xmlHttpRequest();

  _rpc._callback = function()
  {
  if (_rpc.xhr.readyState == _w2c.XmlHttpRequestState.COMPLETE)
  {
    if (_rpc.xhr.status == _w2c.HttpResponseCode.SUCCESS)
    {
    _w2c.Fault.clear();
    _w2c.Fault.Code = _rpc.xhr.status;

    if (_rpc.handleSuccess)
    {
      _rpc.handleSuccess(_rpc.xhr.responseText);
    }
    }
    else
    {
    try
    {
      _w2c.Fault.clear();
      _w2c.Fault.Code = _rpc.xhr.status;
      _w2c.Fault.Message = _rpc.xhr.responseText;
      _w2c.Fault.RemoteServiceUrl = _rpc.url;

      if (_rpc.handleFault)
      {
      _rpc.handleFault(_rpc.xhr.responseText);
      }
    }
    catch (ignoredException)
    {
      alert("XHR callback exception");
    }
```

```
    finally
    {
    }
    }
  }
  };  // _rpc._callback()

  _rpc.setUrl = function(urlString)
  {
  _rpc.url = _w2c.Interface.getMethodUri(urlString);

  _w2c.Internal.RemoteServiceUrl = _rpc.url;
  };  // _rpc.setUrl()

  _rpc.execute = function(synchronicity)
  {
  if (typeof synchronicity == "undefined")
  {
    synchronicity = _w2c.Synchronicity.ASYNCHRONOUS;
  }

  _w2c.Internal.LastSynchronicity = synchronicity;
  _w2c.Internal.LastRequestEpoch = new Date().getTime();

  _rpc.xhr.abort();
  _rpc.xhr.onreadystatechange = function(){};

  if (synchronicity == _w2c.Synchronicity.ASYNCHRONOUS)
  {
    _rpc.xhr.onreadystatechange = _rpc._callback;
  }

  _rpc.xhr.open(_rpc.method, _rpc.url, synchronicity);
  _rpc.xhr.send(_rpc.payload);

  if (synchronicity == _w2c.Synchronicity.SYNCHRONOUS)
  {
    _rpc._callback();
  }
  }; // _rpc.execute()

  _rpc.url = _w2c._getApiRootContext();
  _rpc.method = _w2c.HttpMethod.GET;
  _rpc.payload = null;
  _rpc.handleSuccess = null;
  _rpc.handleFault = null;
};

// API root context

this._getApiRootContext = function()
{
  var apiRoot = "";

  if (_w2c.Session.Host.length > 0)
  {
  apiRoot += _w2c.Session.Protocol + "://";
  apiRoot += _w2c.Session.Host + ":";
  apiRoot += _w2c.Session.Port;
  }

  apiRoot += _w2c.Session.RootContext;
  apiRoot += _w2c.Session.ApiFormat + "/";

  return apiRoot;
};

// Fault capture

this._FaultDetails = function()
```

```
{
  _faultDetails = this;

  this.Code = 0;
  this.Message = "";
  this.FunctionName = "";
  this.RemoteServiceUrl = "";

  this.clear = function()
  {
  _faultDetails.Code = 0;
  _faultDetails.Message = "";
  _faultDetails.FunctionName = "";
  _faultDetails.RemoteServiceUrl = "";
  };
};

this._Log = function()
{
  _log = this;

  _log.LoggingLevel =
  {
  ERROR: 4,
  INFO: 2,
  DEBUG: 1,
  NONE: 0
  };

  _log.level = _log.LoggingLevel.NONE;
  _log.htmlElement = null;

  _log.setHtmlElement = function(logElementId)
  {
  _log.htmlElement = document.getElementById(logElementId);
  };

  _log.error = function(message)
  {
  if (_log.level & _log.LoggingLevel.ERROR)
  {
    _log.write("ERROR - " + message);
  }
  };

  _log.info = function(message)
  {
  if (_log.level & _log.LoggingLevel.INFO)
  {
    _log.write("INFO - " + message);
  }
  };

  _log.debug = function(message)
  {
  if (_log.level & _log.LoggingLevel.DEBUG)
  {
    _log.write("DEBUG - " + message);
  }
  };

  _log.write = function(message)
  {
  if (_log.htmlElement !== null)
  {
    _log.htmlElement.innerHtml += message + "<br/>";
  }
  };
};
```

```
// Defaults

this._SessionObject = function()
{
  _sessionDefaults = this;

  _sessionDefaults.State =
  {
  READY: "READY",
  CLOSED: "CLOSED"
  };

  _sessionDefaults.Protocol = "";
  _sessionDefaults.Host = "";
  _sessionDefaults.Port = "";
  _sessionDefaults.Username = "admin";
  _sessionDefaults.Password = "formula";
  _sessionDefaults.RootContext = _w2c.RootContext.EXTERNAL_ROOT_CONTEXT;
  _sessionDefaults.ApiFormat = _w2c.ApiFormat.REST;
  _sessionDefaults.Synchronicity = _w2c.Synchronicity.ASYNCHRONOUS;
  _sessionDefaults.Status = _sessionDefaults.State.CLOSED;

  _sessionDefaults.getStatus = function()
  {
  _getStatus = this;

  _getStatus.pingSuccessHandler = function()
  {
    _sessionDefaults.Status = _sessionDefaults.State.READY;
  };

  _getStatus.pingFaultHandler = function()
  {
    _sessionDefaults.Status = _sessionDefaults.State.CLOSED;
  };

  _w2c.ping(_getStatus.pingSuccessHandler, _getStatus.pingFaultHandler,
_w2c.Synchronicity.SYNCHRONOUS);

  return _sessionDefaults.Status;
  };

  _sessionDefaults.isSessionActive = function()
  {
  return _sessionDefaults.getStatus();
  };
};

this._SessionInternals = function()
{
  this.RemoteServiceUrl = "";
  this.LastSynchronicity = _w2c.Synchronicity;
  this.LastRequestEpoch = 0;
};

_w2c.Internal = new _w2c._SessionInternals();
_w2c.Session = new _w2c._SessionObject();
_w2c.Fault = new _w2c._FaultDetails();
_w2c.Log = new _w2c._Log();
};

// End of file
```

4.3.3 Methods and Properties

The following objects and properties are available:

- Session

The following methods are available:

- createSession()
- deleteSession()
- ping()
- readElementValue()
- readElementAttributes()
- readElementChildren()
- readElementAlarmsRealtime()
- performElementOperation()

The following describes the methods and their arguments:

- **createSession(successHandler, faultHandler):** Creates a new session using the credentials provided in Session.Username and Session.Password. This method is synchronous.

  Arguments:

  - **successHandler:** Callback for successful creation of a session.
  - **faultHandler:** Callback when session creation fails.
- **deleteSession(successHandler, faultHandler):** Deletes an existing session. This method is asynchronous.

  Arguments:

  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.
- **ping(successHandler, faultHandler, synchronicity):** Performs a session ping. This method can synchronous or asynchronous.

  Arguments:

  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.
  - **Synchronicity:** (Synchronicity.ASYNCHRONOUS || Synchronicity.SYNCHRONOUS) Synchronicity flag (True is asynchronous, False if synchronous).
- **readElementValue(elementIdentity, successHandler, faultHandler):** Retrieves generic element status. This method is asynchronous.

  Arguments:

  - **elementIdentity:** Element DName in x.500 or base64 encoded format.
  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.

- **readElementAttributes(elementIdentity, attributeNames, successHandler, faultHandler):**

  Retrieves specified element attributes. This method is asynchronous.

  Arguments:

  - **elementIdentity:** Element DName in x.500 or base64 encoded format.
  - **attributeNames:** Type is Utils.AssociativeList[], which is a list of attribute names.
  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.
- **readElementChildren(elementIdentity, successHandler, faultHandler):** Retrieves all children of an element. This method is asynchronous.

  Arguments:

  - **elementIdentity:** Element DName in x.500 or base64 encoded format.
  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.
- **readElementAlarmsRealtime(elementIdentity, channel, successHandler, faultHandler):**

  Retrieves real-time element alarms. This method is asynchronous.

  Arguments:

  - **elementIdentity:** Element DName in x.500 or base64 encoded format.
  - **channel:** Enum: AlarmChannels Alarm data channel.
  - **successHandler:** Callback if successful.
  - **faultHandler:** Callback if error.

4.4 Example Code for JavaScript Library

The following HTML/JavaScript source provides a simple test harness for the JavaScript library functions:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=8" />
    <title>Web 2.0 Connect</title>
    <style type="text/css">
      BODY
      {
        background-color: #242424;
        color: #FFAA00;
        font-family: Verdana;
        font-size: 10pt;
      }
      div#ContentContainer
      {
        margin: 0 auto 0 auto;
        width: 800px;
        height: 90%;
      }
      div#OuputContainer
      {
        background-color: #101010;
        padding: 4px 4px 4px 4px;
        text-align: left;
      }
```

```
    input
    {
      background-color: #101010;
      color: #FFAA00;
      border: solid 0px black;
      padding-bottom: 2px;
      margin: 2px 1px 2px 1px;
      border: solid 1px #202020;
      font-family: Arial;
    }
    input[type="button"]
    {
      background-color: #141414;
      color: #FFAA00;
      font-size: 12pt;
      border: solid 0px black;
      padding-bottom: 2px;
      margin: 2px 1px 2px 1px;
      border: solid 1px #202020;
      cursor: pointer;
    }
    input[type="text"]
    {
      background-color: #141414;
      color: #FFCC00;
      font-size: 10pt;
      width: 99%;
      border: solid 0px black;
      margin: 1px 1px 1px 1px;
      padding: 0px 4px 0px 4px;
    }
    input[type="radio"]
    {
      color: #FFCC00;
      font-size: 8pt;
    }
    textarea#debugData
    {
      background-color: black;
      color: #FFFF00;
      border: solid 0px black;
      margin: 4px 4px 8px 4px;
      overflow: auto;
    }
    h1[id="appTitle"]
    {
      font-family: Courier New, Verdana;
      font-size: 18pt;
      font-weight: bold;
      color: #FFBB00;
    }
  </style>
  <script type="text/javascript" src="scripts/1.0/bsm.web2connect.js"></script>
  <script type="text/javascript">

    var BSM_HOST = document.location.hostname;
    var BSM_PORT = document.location.port;

    var testElementIdentity;
    var rootElementIdentity;
    var debug;
    var w;

    function pageLoad()
    {
      document.getElementById("host").value = BSM_HOST;
      document.getElementById("port").value = BSM_PORT;

      testElementIdentity = document.getElementById("testElementIdentity");
      rootElementIdentity = document.getElementById("rootElementIdentity");
```

```
  debug = document.getElementById("debugData");
  debug.value = "Ready";

  w = new BSM.Web2Connect();

  setFormatJson();

  doCreateSession();
}

function doCreateSession()
{
  w.Session.Protocol = w.HttpProtocol.HTTP;
  w.Session.Host = document.getElementById("host").value;
  w.Session.Port = document.getElementById("port").value;
  w.Session.Username = document.getElementById("username").value;
  w.Session.Password = document.getElementById("password").value;

  w.createSession(createSessionHandler, createSessionFaultHandler);
}

function createSessionHandler(response)
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += getLastSynchronicityMessage() + "\n";
  debug.value += "Session created " + Date() + "\n" + response;
}

function createSessionFaultHandler(response)
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += getLastSynchronicityMessage() + "\n";
  debug.value += response;
}

function doDeleteSession()
{
  w.deleteSession(deleteSessionHandler, deleteSessionFaultHandler);
}

function deleteSessionHandler(response)
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += "Session deleted " + Date() + "\n" + response;
}

function deleteSessionFaultHandler(response)
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += response;
}

function doSessionStatus()
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += getLastSynchronicityMessage() + "\n";
  debug.value += w.Session.getStatus() + "\n";
}

function doUserInfo(synchronicity)
{
  w.readUser(userHandler, userFaultHandler, synchronicity);
}

function userHandler(response)
{
  debug.value = w.Internal.RemoteServiceUrl + "\n";
  debug.value += "User info " + Date() + "\n" + response;
}
```

```
      function userFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += response;
      }

      function doPing(synchronicity)
      {
        w.ping(pingHandler, pingFaultHandler, synchronicity);
      }

      function pingHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += getLastSynchronicityMessage() + "\n";
        debug.value += "Pinged at " + Date() + "\n" + response;
      }

      function pingFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += getLastSynchronicityMessage() + "\n";
        debug.value += response;
      }

      function doElementOperation()
      {
        var identity = new String(rootElementIdentity.value);
        w.performElementOperation(identity, "Debug", elementOperationHandler,
elementOperationFaultHandler);
      }

      function elementOperationHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "Operation performed " + Date() + "\n" + response;
      }

      function elementOperationFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += response;
      }

      function doElementValue()
      {
        var identity = new String(testElementIdentity.value);
        w.readElementValue(identity, elementValueHandler,
elementValueFaultHandler);
      }

      function elementValueHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "Read element value " + Date() + "\n" + response;
      }

      function elementValueFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += response;
      }

      function doElementChildren()
      {
        var identity = new String(testElementIdentity.value);
        w.readElementChildren(identity, elementChildrenHandler,
elementChildrenFaultHandler);
      }
```

```
      function elementChildrenHandler(response, elements)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";

        if (w.Session.ApiFormat == w.ApiFormat.JSON)
        {
          debug.value += "\nelements.count is " + elements.count;

          if (elements.count > 0)
          {
            for (var i = 1; i <= elements.count; i++)
            {
              debug.value += "\nelements.getItem(" + i + ").name = " +
elements.getItem(i).name;
            }
          }
        }

        debug.value += "\n\nRead element children " + Date() + "\n" + response;
      }

      function elementChildrenFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += response;
      }

      function doElementAttributes()
      {
        var identity = new String(testElementIdentity.value);
        var attributes = new w.Utils.AssociativeList();

        attributes.add("lastAlarmSummary");
        attributes.add("childCount");
        attributes.add("relationshipCount");
        attributes.add("alarmCount");

        w.readElementAttributes(identity, attributes, elementAttributesHandler,
elementAttributesFaultHandler);
      }

      function elementAttributesHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "Read element attributes " + Date() + "\n" + response;
      }

      function elementAttributesFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += response;
      }

      function doElementAlarmsRealtime()
      {
        var identity = new String(testElementIdentity.value);

        w.readElementAlarmsRealtime(identity, w.AlarmChannel.GENERAL, null,
elementAlarmsHandler, elementAlarmsFaultHandler);
      }

      function elementAlarmsHandler(response, alarms)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";

        if (w.Session.ApiFormat == w.ApiFormat.JSON)
        {
          if (alarms.count > 0)
          {
```

```
            debug.value += "\nalarms.count is " + alarms.count;

            for (var i = 1; i <= alarms.count; i++)
            {
             debug.value += "\nalarms.getItem(" + i + ").affectedElementName = " +
alarms.getItem(i).affectedElementName;
            }
          }
        }

        debug.value += "\n\nRead element alarms realtime " + Date() + "\n" +
response;
      }

      function elementAlarmsFaultHandler(response)
      {
        debug.value = "Exception [" + response.exception + "]\n";
        debug.value += w.Internal.RemoteServiceUrl + "\n";
        debug.value += response
      }

      function doElementQueryPerformanceInfo()
      {
        var identity = new String(testElementIdentity.value);
        w.readElementPerformanceInfo(identity, elementPerformanceInfoHandler,
elementPerformanceInfoFaultHandler);
      }

      function elementPerformanceInfoHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "Read element performance information " + Date() + "\n" +
response;
      }

      function elementPerformanceInfoFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "\nEnsure Data Warehouse is enabled\n\n";
        debug.value += response;
      }

      function doElementQueryPerformanceSeries()
      {
        var identity = new String(testElementIdentity.value);
        var query = new w.Utils.PerformanceSeriesQuery();

        query.profileName = "Demo";
        query.expressionName = "Script";
        query.fromTimeEpoch = 1220482800000;
        query.toTimeEpoch = 1220569200000;
        query.limitCount = 200;

        w.readElementPerformanceSeries(identity, query,
elementPerformanceSeriesHandler, elementPerformanceSeriesFaultHandler);
      }

      function elementPerformanceSeriesHandler(response, series)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";

        if (w.Session.ApiFormat == w.ApiFormat.JSON)
        {
          if (series.count > 0)
          {
            debug.value += "\nseries.count is " + series.count;
            debug.value += "\nseries.Item(1).dateValue is " +
series.getItem("1").dateValue;
            debug.value +=
"\nseries.getItemByDateEpoch(series.getItem(1).dateValue).seriesValue is " +
```

```
series.getItemByDateEpoch(series.Item(1).dateValue).seriesValue;
            debug.value += "\nseries.earliestDate is " + series.earliestDate;
            debug.value += "\nseries.latestDate is " + series.latestDate;
            debug.value += "\nseries.lowestValue is " + series.lowestValue;
            debug.value += "\nseries.highestValue is " + series.highestValue;
          }
        }

        debug.value += "\n\nRead element performance series " + Date() + "\n" +
response;
      }

      function elementPerformanceSeriesFaultHandler(response)
      {
        debug.value = w.Internal.RemoteServiceUrl + "\n";
        debug.value += "\nEnsure Data Warehouse is enabled\n\n";
        debug.value += response;
      }

      function getLastSynchronicityMessage()
      {
        return "Internal.LastSynchronicity " +
w.Internal.LastSynchronicity.toString().toUpperCase();
      }

      function setFormatXml()
      {
        w.Session.ApiFormat = w.ApiFormat.XML;
        document.getElementById("radioXML").checked = true;
      }

      function setFormatJson()
      {
        w.Session.ApiFormat = w.ApiFormat.JSON;
        document.getElementById("radioJSON").checked = true;
      }
    </script>
  </head>
  <body onload="pageLoad()">
    <div id="ContentContainer">
      <h1 id="appTitle">// Web 2.0 Connect JavaScript Library</h1>
      <div id="OuputContainer">
        <textarea rows="15" cols="95" id="debugData"></textarea>
        <br />
        Response Format
        <input type="radio" onclick="setFormatXml()" value="XML" name="ApiFormat"
id="radioXML" />XML
       <input type="radio" onclick="setFormatJson()" value="JSON" name="ApiFormat"
id="radioJSON" />JSON
        <br />
        <table width="100%">
          <tr>
            <td width="20">
              Host
            </td>
            <td>
              <input type="text" size="60" id="host" value="localhost" />
            </td>
          </tr>
          <tr>
            <td width="20">
              Port
            </td>
            <td>
              <input type="text" size="30" id="port" value="8080" />
            </td>
          </tr>
          <tr>
            <td>Username</td>
            <td><input type="text" size="90" id="username" value="admin" /></td>
```

```
          </tr>
          <tr>
            <td>Password</td>
            <td><input type="text" size="90" id="password" value="formula" /></td>
          </tr>
          <tr>
            <td>Element</td>
            <td><input type="text" size="90" id="testElementIdentity"
value="Test=Test/root=Organizations" /></td>
          </tr>
          <tr>
            <td>
              Root
            </td>
            <td><input type="text" size="90" id="rootElementIdentity"
value="root=Organizations" /></td>
          </tr>
        </table>
        <br/>
        <input type="button" onclick="doCreateSession()" value="Start Session"/>
        <input type="button" onclick="doDeleteSession()" value="End Session"/>
        <input type="button" onclick="doSessionStatus()" value="Session Status"/>
        <input type="button" onclick="doUserInfo(true)" value="User Info" />
        <br/>
        <input type="button" onclick="doPing(true)" value="Ping Async"/>
        <input type="button" onclick="doPing(false)" value="Ping Sync"/>
        <input type="button" onclick="doElementOperation()" value="Perform
Operation"/>
        <br/>
        <input type="button" onclick="doElementValue()" value="Element Value"/>
        <input type="button" onclick="doElementAttributes()" value="Element
```

```
Attributes"/>
        <input type="button" onclick="doElementChildren()" value="Element
Children"/>
        <input type="button" onclick="doElementAlarmsRealtime()" value="Element
Alarms Realtime"/>
        <br/>
        <input type="button" onclick="doElementQueryPerformanceInfo()"
value="Element Query Performance Info"/>
        <input type="button" onclick="doElementQueryPerformanceSeries()"
value="Element Query Performance Series"/>
      </div>
    </div>
  </body>
</html>
```


- Section 4.4.1, "Example Data," on page 63

4.4.1 Example Data

The configuration store export in the following example defines an element named AlarmCounts with five attributes used by the JavaScript library example. Save the following XML text as `AlarmCounts.config.xml` and import it into Operations Center:

```
<configstore>
  <entry>
    <id>26</id>
    <name>Enterprise</name>
    <className>root</className>
  </entry>
  <entry>
    <id>220</id>
    <parent>26</parent>
    <name>Organizations</name>
    <className>root</className>
  </entry>
  <entry>
    <id>2785</id>
    <parent>220</parent>
    <name>AlarmCounts</name>
    <className>visualization</className>
    <attributes>
      <entry>
        <string>okayCount</string>
        <string>50</string>
      </entry>
      <entry>

        <string>criticalCount</string>
        <string>10</string>
      </entry>      <entry>
        <string>infoCount</string>
        <string>5</string>
      </entry>
      <entry>
        <string>majorCount</string>
        <string>30</string>
      </entry>
      <entry>
        <string>minorCount</string>
        <string>50</string>
      </entry>
```

```
      <entry>
        <string>Flags</string>
        <long>16</long>
      </entry>
    </attributes>
    <exportRoot>true</exportRoot>
  </entry>
  <relationship>
    <left>220</left>
    <right>2785</right>
    <id>2786</id>
    <name>internal:match:static</name>
  </relationship>
</configstore>
```

5 Flex Quick Start

This section provides examples using Adobe Flex framework for rich internet applications to communicate with Operations Center using Web 2.0 Connect and build solutions that can be compiled into Flash (SWF) applications.

Flex applications can be built using the free Flex SDK from Adobe and deployed as desktop applications using Adobe AIR framework or as Flash SWF files hosted within Web pages. The ubiquity of the Flash player plug-in for Web browsers means wide cross platform support.

Flex is a programmer centric way of developing rich internet applications (RIA). It is not the intention of this guide to teach the Flex software development kit (SDK) or the Flex integrated development environment (IDE). It is assumed that the reader has experience of high level programming languages and at the very least an understanding of the basic principles used in modern Web technologies.

- Section 5.1, “Using HTTPService,” on page 65
- Section 5.2, “Using Remote Procedure Calls,” on page 68

5.1 Using HTTPService

This section provides a simple introduction to using the Flex HTTPService component to interface with the REST API.

- Section 5.1.1, “Creating a Session using HTTPService,” on page 65
- Section 5.1.2, “Closing a Session using HTTPService,” on page 67

5.1.1 Creating a Session using HTTPService

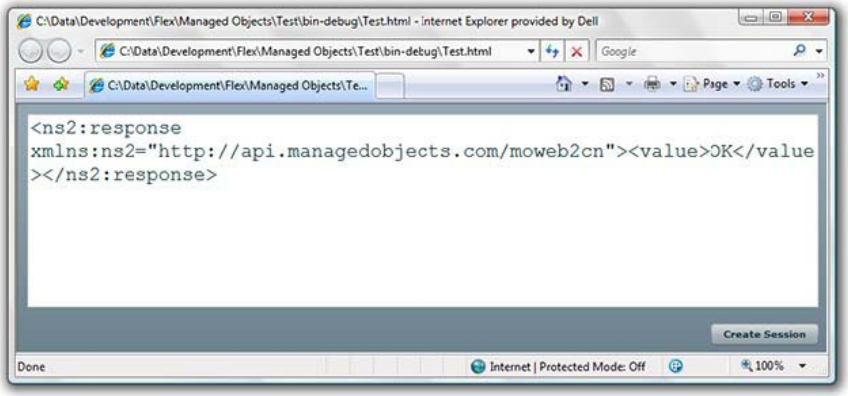
The following code sample illustrates creating a Web 2.0 Connect session using the REST API and the native HTTPService component available in Flex. The principles in this example are very similar to the JavaScript examples in Chapter 4, “Getting Connected,” on page 21.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  horizontalAlign="left"
  verticalGap="15"
  horizontalGap="15"
  layout="absolute"
  height="258" width="523">
  <mx:Script>
  <![CDATA[
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    private function createSession() : void
    {
      w2cService.cancel();
      var params : Object = new Object();
```

```
      params.username = "admin";
      params.password = "formula";
      w2cService.send(params);
    }
    private function sessionCreateHandler(event : ResultEvent) : void
    {
      responseText.text = event.result.toString();
    }
    private function sessionCreateFault(event : FaultEvent) : void
    {
      responseText.text = event.fault.toString();
    }
   ]]>
  </mx:Script>
  <mx:HTTPService
    id="w2cService"
    url="http://manor-as08-v:8080/moweb2cn/rest/session"
    resultFormat="xml"
    result="sessionCreateHandler(event)"
    fault="sessionCreateFault(event)"
  />
  <mx:Button
    label="Create Session"
    click="createSession()"
    textAlign="center"
    x="451" y="218"
  />
  <mx:TextArea
     id="responseText"
     text="Click Logon to create a session"
     wordWrap="true"
     x="10" y="10"
     width="503" height="200"
  />
</mx:Application>
```

This source code for this sample can be cut and pasted into a new Flex MXML file and compiled into a SWF file.

When *Create Session* is clicked, a new Web 2.0 Connect session is created. If create session is successful, the field displays the XML response; or, if the create session fails, the error message from the server is displayed.



Subsequent requests to create a session before a session times out results in error messages from the server alerting users that they are already logged in.

5.1.2 Closing a Session using HTTPService

The following code expands on the create session example and adds a function to terminate an existing session. The `closeSession()` function uses a declarative HTTPService instance to overcome the lack of support for native HTTP DELETE method in Flex. The Flex HTTPService component only supports HTTP methods PUT and DELETE in proxy mode.

```
<?xml version="
1.0" encoding="utf-8"?>

<mx:Application  xmlns:mx="http://www.adobe.com/2006/mxml"
  verticalGap="16"
  horizontalGap="16"
  layout="absolute"
  height="256" width="830">
  <mx:Script>
  <![CDATA[
    import mx.rpc.events.FaultEvent;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;
    private function createSession() : void
    {
      var params : Object = new Object();
      params.username = "admin";
      params.password = "formula";
      w2cService.cancel();
      w2cService.send(params);
    }
    private function createSessionHandler(event : ResultEvent) : void
    {
      responseText.text = "Session created\n" + event.result.toString();
    }
    private function closeSession() : void
    {
      var w2cHttpService : HTTPService = new HTTPService();
      w2cHttpService.addEventListener(ResultEvent.RESULT, closeSessionHandler);
      w2cHttpService.headers = {"X-HTTP-Method-Override" : "DELETE"};
      w2cHttpService.method = "POST";

      w2cHttpService.contentType = "application/xml";
      w2cHttpService.resultFormat = "xml";
      w2cHttpService.url = "http://manor-as08-v:8080/moweb2cn/rest/session";
      w2cHttpService.send(new XML("<response/>"));
    }    private function closeSessionHandler(event : ResultEvent) : void
    {

      responseText.text = "Session closed\n" + event.result.toString();    }
     private function httpServiceFaultHandler(event : FaultEvent) : void
    {
      responseText.text = "Ooops!\n" + event.fault.toString();

    }  ]]>
  </mx:Script>
  <mx:HTTPService
    id="w2cService"
    url="http://manor-as08-v:8080/moweb2cn/rest/session"
    resultFormat="xml"
    result="createSessionHandler(event)"
    fault="httpServiceFaultHandler(event)"
  />
  <mx:TextArea
    id="responseText"
    text="Click button to create a session"
    fontSize="20" fontFamily="Courier New"
    x="10" y="10"
    width="811" height="200"
    wordWrap="true"
```

```
  />
  <mx:Button
    label="Create Session"
    click="createSession()"
    textAlign="center"
    x="593" y="224"
  />
  <mx:Button
    label="Close Session"
    click="closeSession()"
    textAlign="center"
    x="711" y="224"
  />
</mx:Application>
```

The workaround uses the X-HTTP-Method-Override modifier in the request header to perform a DELETE request. This method should also work with firewalls that can be configured to block HTTP operations using the PUT and DELETE methods. While the RPC `closeSession()` method is simpler this workaround is the only way to terminate a session using the REST API.

Remember, with Web 2.0 Connect it is possible to mix and match protocols so there is no reason why an application cannot initiate and manage sessions using RPC while performing simpler read operations using the HTTPService component for ease of its data binding capabilities.

5.2 Using Remote Procedure Calls

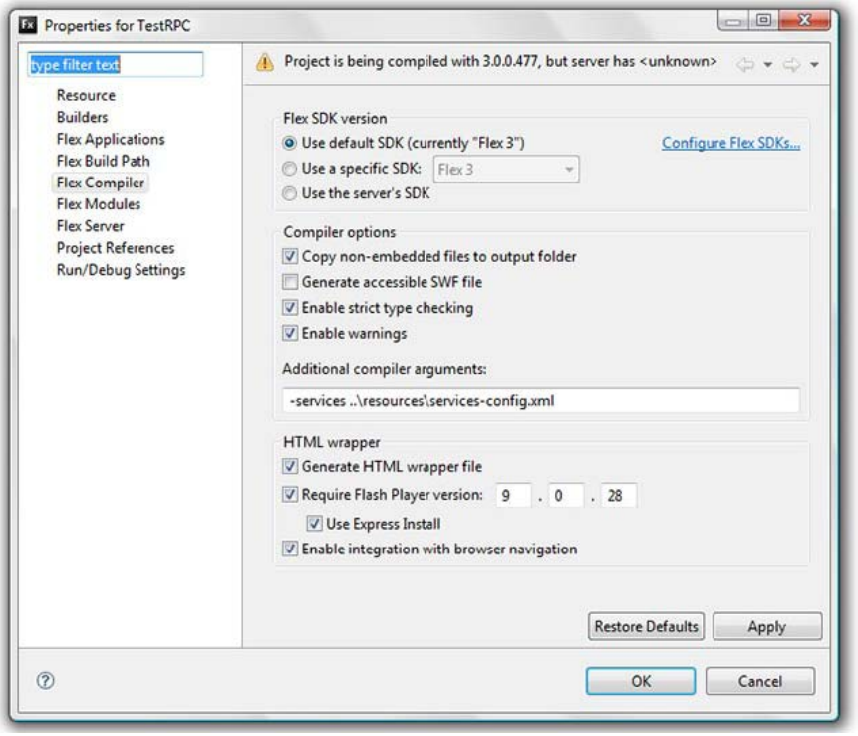
This section provides an introduction to using the Web 2.0 Connect Remote Procedure Call (RPC) library for Flex to interface with the REST API.

- Section 5.2.1, “Configuring Remote Services,” on page 69
- Section 5.2.2, “Services Configuration,” on page 70
- Section 5.2.3, “Building and Deploying,” on page 71
- Section 5.2.4, “Creating a Session using RPC,” on page 71
- Section 5.2.5, “Closing a Session using RPC,” on page 72
- Section 5.2.6, “A Word About Security,” on page 73

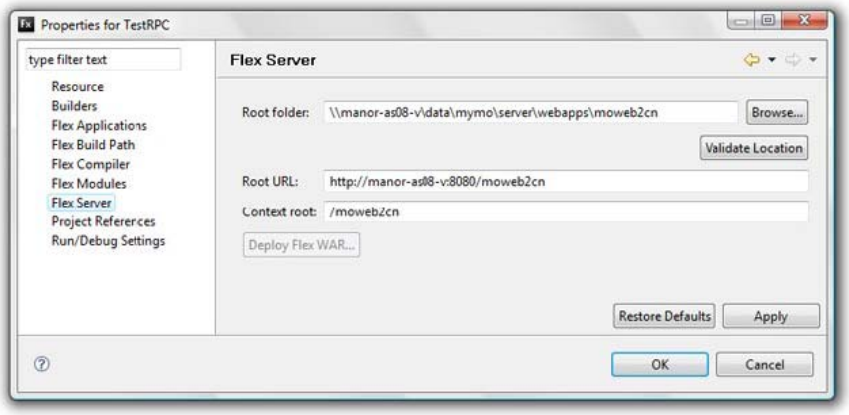
5.2.1 Configuring Remote Services

A Flex application needs to know how to access remote services. The configuration of remote services is specified using the `-services` *`path_to_services-config.xml`* compiler directive. For more information about the `services-config.xml` file refer to the online Flex SDK documentation.

In Flex Builder this is specified as a compiler argument in the *Flex Compiler* options in the Flex project properties dialog box:



Flex Builder also needs to know server options for a Flex server application. To keep it simple, the Flex projects.



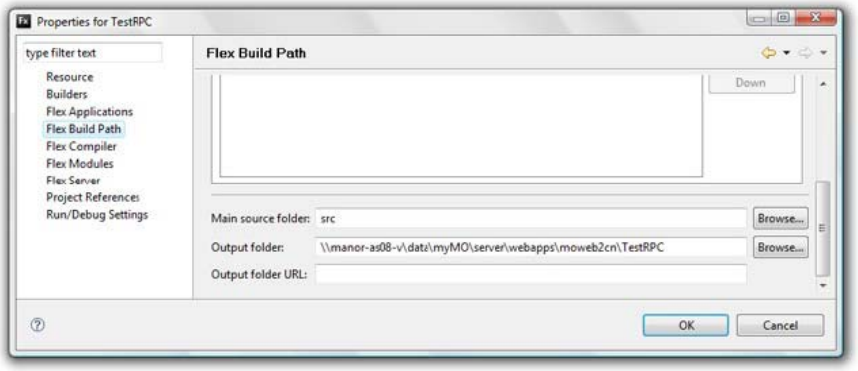
5.2.2 Services Configuration

The `services-config.xml` file should be placed in the `WEB-INF/flex` directory where the Flex application is deployed.

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <services>
    <service class="flex.messaging.services.RemotingService" id="remoting-service"
messageTypes="flex.messaging.messages.RemotingMessage">
      <adapters>
        <adapter-definition
class="flex.messaging.services.remoting.adapters.JavaAdapter" default="true"
id="java-object"/>
      </adapters>
      <destination id="Web2Internal">
        <channels>
          <channel ref="Web2InternalAMFChannel"/>
        </channels>
        <properties>
          <factory>spring</factory>
          <source>Web2InternalService-amf</source>
        </properties>
      </destination>
      <destination id="Web2Connect">
        <channels>
          <channel ref="Web2ConnectAMFChannel"/>        </channels>
        <properties>
          <factory>spring</factory>
          <source>Web2ConnectService-amf</source>
        </properties>
      </destination>
    </service>
  </services>
  <channels>
    <channel-definition class="mx.messaging.channels.AMFChannel"
id="Web2InternalAMFChannel">
      <endpoint class="flex.messaging.endpoints.AMFEndpoint" uri="http://manor-
as08-v:8080/moweb2cn/amf/Web2InternalService"/>
    </channel-definition>
    <channel-definition class="mx.messaging.channels.AMFChannel"
id="Web2ConnectAMFChannel">
      <endpoint class="flex.messaging.endpoints.AMFEndpoint" uri="http://manor-
as08-v:8080/moweb2cn/amf/Web2ConnectService"/>
    </channel-definition>
  </channels>
  <factories>
    <factory class="org.codehaus.enunciate.modules.amf.SpringFactory" id="spring"/
>
  </factories>
  <!--
  <logging>
     <target class="flex.messaging.log.ConsoleTarget" level="Debug" />
  </logging>
  -->
</services-config>
```

5.2.3 Building and Deploying

A Web hosted Flex application needs to be deployed to a Web server. Flex Builder is able to copy the compiled project and associated assets to target output path. In Flex Builder configure the project's *Flex Build Path* option for the *Output Folder* by specifying a path to a directory where Flex has write permissions:



5.2.4 Creating a Session using RPC

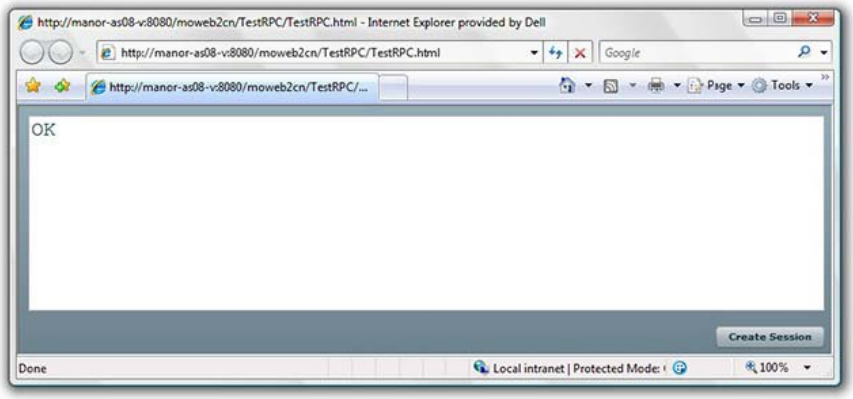
The supplied SWC archive file contains a library of functions and data types that saves developers from having to write their own library from scratch. The Web 2.0 Connect Flex library file can be download from the downloads section on the home page of the Web 2.0 Connect installation.

The following source code demonstrates creating a Web 2.0 Connect session using the RPC library:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  verticalGap="16"
  horizontalGap="16"
  layout="absolute"
  height="256" width="830">
  <mx:Script>
  <![CDATA[
    import com.mosol.moweb2cn.Web2Connect;
    import com.mosol.moweb2cn.Web2Connect.CreateSessionResultEvent;
    private function createSession() : void
    {
      var rpc : Web2Connect = new Web2Connect();
      rpc.addEventListener("createSession", sessionCreateHandler);
      rpc.addEventListener("fault", rpcFaultHandler);
      rpc.createSession("admin", "formula");
    }
    private function sessionCreateHandler(event : CreateSessionResultEvent) : void
    {
      responseText.text = event.result.value;
    }
    private function rpcFaultHandler(event : Event) : void
    {
      responseText.text = event.toString();
    }
   ]]>
  </mx:Script>
  <mx:TextArea
```

```
    id="responseText"
    text="Click button to create a session"
    fontSize="20" fontFamily="Courier New"
    x="10" y="10"
    width="811" height="200"
    wordWrap="true"
  />
  <mx:Button
    label="Create Session"
    click="createSession()"
    textAlign="center"
    x="711" y="226"
  />
</mx:Application>
```

The RPC library provides a set of managed code to access the Web 2.0 Connect APIs. Unlike the examples in Chapter 3, "Programming Interfaces," on page 17, the RPC created session responds with an event of type CreateSessionResultEvent. If the session creation is successful, the value property of the event's result object is the string `OK`.



And remember, all remote services are handled by applications using asynchronous requests.

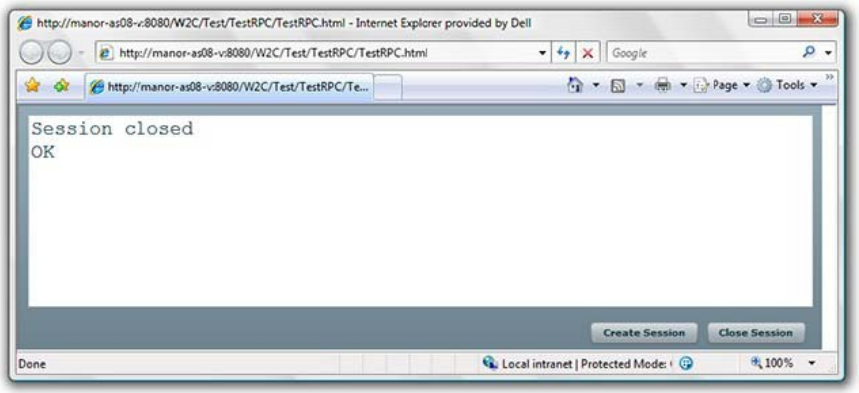
5.2.5 Closing a Session using RPC

Although a session eventually times out as far as the Web server is concerned, the Operations Center Web client session remains until the session is terminated.

By adding the following ActionScript functions to the previous example and a new button to invoke the function, the Web 2.0 Connect session can be closed and the Managed Object Web client session terminated:

```
private function closeSession() : void
{
  var rpc : Web2Connect = new Web2Connect();
  rpc.addEventListener("closeSession", closeSessionHandler);
  rpc.addEventListener("fault", rpcFaultHandler);
  rpc.closeSession();
}
private function closeSessionHandler(event : CloseSessionResultEvent) : void
{
  responseText.text = "Session closed\n" + event.result.value;
}
```

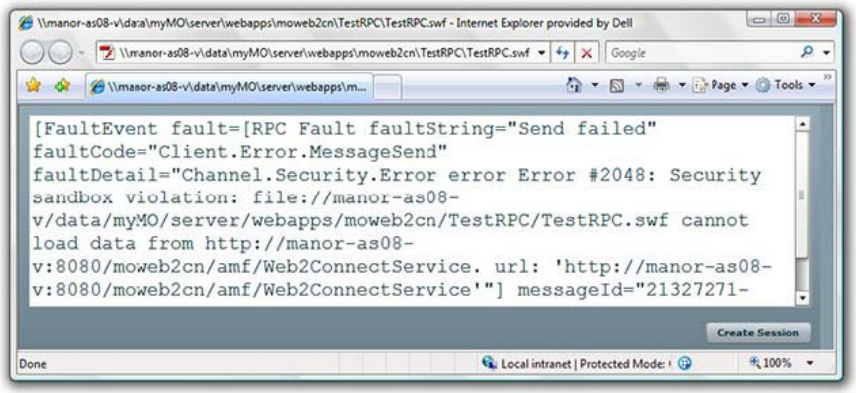
The `closeSession()` RPC method response event is of type CloseSessionResultEvent and the response result value is simply an `OK` string, if successful; otherwise, an error message is returned:



Crashed sessions leave orphaned Web client sessions on the Operations Center server, which might need to be manually terminated.

5.2.6 A Word About Security

Flex applications are compiled into Flash SWF format files and use the Flash Player as their runtime library. Flash applications are typically hosted within a Web page, but there is nothing to stop a user from downloading the SWF file to their desktop and open it directly from their local file system. Doing so, however, causes a security violation within the Flash runtime and any attempts to access remote services results in a `Security sandbox violation` error.



Flex applications targeted for the Web must be hosted on a Web server to access remote services without compromising client security. Offline Flex applications intended for desktop deployment must use the Adobe AIR framework that provides a secure offline sandbox for Flash.

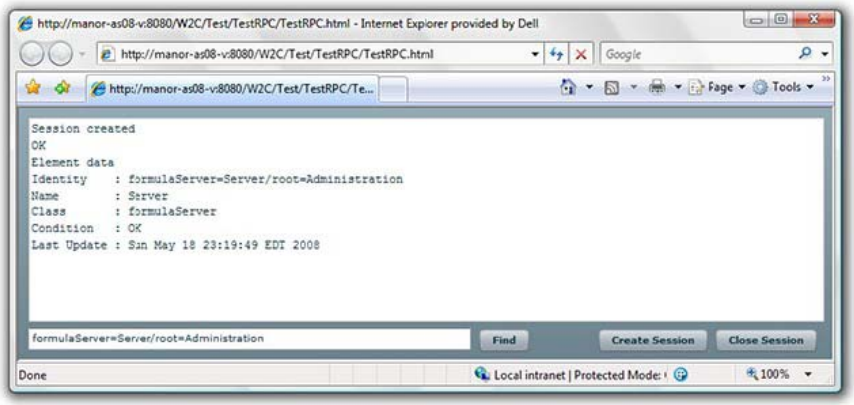
6 Flex Library

This section focuses on accessing elements and element properties. All examples in this section use Flex components and the Web 2.0 Connect remote services, mixing APIs as necessary to provide easier to understand examples.

- Section 6.1, “Accessing Element Properties,” on page 75

6.1 Accessing Element Properties

This example uses RPC to manage session and REST to fetch a single element using the rest/element/value/{identity} URI. When the application is executed a field is available to enter the DName of an element. Clicking *Find* submits the REST element/value request and display the standard properties inherent to all elements.



Using the Flex HTTPService component, the XML response returned by the REST API is preparsed, providing easy access to the element properties through the ResultEvent object.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  verticalGap="16"
  horizontalGap="16"
  layout="absolute"
  height="256" width="830"
  >
  <mx:Script>
  <![CDATA[
    import mx.rpc.http.HTTPService;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import com.mosol.moweb2cn.Web2Connect;
    import com.mosol.moweb2cn.ElementValue;
    import com.mosol.moweb2cn.Web2Connect.CreateSessionResultEvent;
    import com.mosol.moweb2cn.Web2Connect.CloseSessionResultEvent;
    private var rpc : Web2Connect = new Web2Connect();
```

```
    private var httpService : HTTPService = new HTTPService();
    private const MO_WEB2CONNECT_REST_ROOT : String = "/moweb2cn/rest/";
    private function createSession() : void
    {
      rpc.addEventListener("createSession", createSessionHandler);
      rpc.addEventListener("fault", rpcFaultHandler);
      rpc.createSession("admin", "formula");
    }
    private function createSessionHandler(event : CreateSessionResultEvent) : void
    {
      responseText.text = "Session created\n" + event.result.value + "\n";
    }
    private function closeSession() : void
    {
      rpc.addEventListener("closeSession", closeSessionHandler);
      rpc.addEventListener("fault", rpcFaultHandler);
      rpc.closeSession();
    }
    private function closeSessionHandler(event : CloseSessionResultEvent) : void
    {
      responseText.text = "Session closed\n" + event.result.value;
    }
     private function getElementValue() : void
    {
      // There is no RPC method for ElementValue so use REST API
      httpService.url = MO_WEB2CONNECT_REST_ROOT + "element/value/" +
elementIdentity.text;
      httpService.addEventListener(ResultEvent.RESULT, getElementValueHandler);
      httpService.resultFormat = "object";
      httpService.send(null);
    }
    private function getElementValueHandler(event : ResultEvent) : void
    {
      // Note that the response XML is pre-processed
      // providing easy access to element properties
      responseText.text += "Element data\n"
        + "Identity    : " + event.result.element.identity + "\n"
        + "Name        : " + event.result.element.name + "\n"
        + "Class       : " + event.result.element.elementClassName + "\n"
        + "Condition   : " + event.result.element.condition + "\n"
        + "Last Update : " + event.result.element.lastupdateString + "\n";
    }
     private function httpServiceFaultHandler(event : FaultEvent) : void
    {
      responseText.text = event.fault.toString();
    }
    private function rpcFaultHandler(event : Event) : void
    {
      responseText.text = event.toString();
    }
   ]]>
  </mx:Script>
       <mx:TextArea id="responseText" text="Click button to create a session"
         fontSize="12" fontFamily="Courier New" x="10" y="10"
         width="811" height="208" wordWrap="true" />
       <mx:TextInput id="elementIdentity"
         text="formulaServer=Server/root=Administration"
         x="10" y="224" width="451" />
       <mx:Button label="Find" click="getElementValue()"
         textAlign="center" x="470" y="226" />
       <mx:Button label="Create Session" click="createSession()"
         textAlign="center" x="591" y="226" />
       <mx:Button label="Close Session" click="closeSession()"
         textAlign="center" x="711" y="226" />
</mx:Application>
```

7 Flex Solutions

This section provides some complete examples developed using Adobe Flex and Web 2.0 Connect.

- Section 7.1, “Example: Flex Alarm Ratio Bar,” on page 77
- Section 7.2, “Example: Alarm Ratio Bar Application,” on page 78
- Section 7.3, “Example: Alarm Ratio Bar Component,” on page 80
- Section 7.4, “Example: Metamodel Property Page,” on page 84

7.1 Example: Flex Alarm Ratio Bar

This Flex example uses the REST methods to fetch alarm condition counts for a given element identity. The visual component was inspired by the alarm count stacked bars from the classic Tivoli Netcool client.



The Alarm Ratio Bar can be used directly from a URL:

AlarmRatioBar.swf?Username=admin&Password=formula&CaptionText=Alarms&RefreshInterval=10&Identity=identity:dname:root=Organization

Table 7-1 lists the parameters that can be passed in the URL query string.

Table 7-1 *URL Query String Parameters*

Parameter	Description
`Username`	Operations Center logon user name.
`Password`	Operations Center logon password.
`CaptionText`	Text to display on alarm ratio bar.
`RefreshInterval`	Refresh interval in seconds (defaults to 30 seconds if not specified).
`Identity`	Operations Center DName in base 64 format.

The identity can be specified in x.500 format, also; however, some combinations have been known to cause an issue in Flex.

This solution assumes that the Flex application is hosted using the HTTP server that is hosting the Web 2.0 Connect service.

Flex source code for component and component implementation is detailed in the remaining sections:

- Section 7.2, “Example: Alarm Ratio Bar Application,” on page 78
- Section 7.3, “Example: Alarm Ratio Bar Component,” on page 80
- Section 7.4, “Example: Metamodel Property Page,” on page 84

7.2 Example: Alarm Ratio Bar Application

Source for Alarm Ratio Bar application, which implements the Alarm Ratio Bar component:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:mosh="managedobjects.visualization.*"
  backgroundColor="#ffffff"
  backgroundGradientColors="#ffffff,#ffffff"
  backgroundGradientAlphas="0.0,0.0"
  applicationComplete="main()"
  >
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import mx.rpc.http.HTTPService;
      import mx.rpc.events.ResultEvent;
      import mx.rpc.events.FaultEvent;
      private var okayCount : int = 10;
      private var infoCount : int = 20;
      private var minorCount : int = 30;
      private var majorCount : int = 40;
      private var criticalCount : int = 10;
      // Constants
      private const RPC_REST_ROOT : String = "/moweb2cn/rest/";
      private const DEFAULT_REFRESH_INTERVAL : Number = 30;
      // Private
      private var rpcUsername : String;
      private var rpcPassword : String;
      private var rpcIdentity : String;
      private var captionText : String;
      private var refreshTimer : Timer;
      private var refreshInterval : Number;
      // Functions
      private function terminateSession() : void
      {
        var rpc : HTTPService = new HTTPService();
        rpc.addEventListener(ResultEvent.RESULT, terminateSessionHandler);
        rpc.headers = {"X-HTTP-Method-Override" : "DELETE"};
        rpc.method = "POST";
        rpc.contentType = "application/xml";
        rpc.resultFormat = "xml";
        rpc.url = RPC_REST_ROOT + "session";
        rpc.send(new XML("<response/>"));
      }
      private function terminateSessionHandler(event : ResultEvent) : void
      {
        // Do nothing
      }
      private function createSession(username : String, password : String) : void
      {
        var rpc : HTTPService = new HTTPService();
        var url : String;
        url = RPC_REST_ROOT + "session?username=" + username + "&password=" +
password;
        rpc.addEventListener(ResultEvent.RESULT, createSessionHandler);
        rpc.method = "GET";
```

```
        rpc.url = url;
        rpc.send(null);
      }
      private function createSessionHandler(event : ResultEvent) : void
      {
        this.updateAlarmConditionCounts(this.rpcIdentity);
      }
      private function updateAlarmConditionCounts(identity : String) : void
      {
        var rpc : HTTPService = new HTTPService();
        var url : String;
        url = RPC_REST_ROOT + "element/attributes/";
        url += identity;
        url +=
"?attributeNames=okayCount,infoCount,minorCount,majorCount,criticalCount";
        rpc.addEventListener(ResultEvent.RESULT, alarmConditionCountsHandler);
        rpc.resultFormat = "object";
        rpc.method = "GET";
        rpc.url = url;
        rpc.send(null);
      }
      private function alarmConditionCountsHandler(event : ResultEvent) : void
      {
        this.terminateSession();
        var a : Array = [0,1,2,3,4];
        a[0] = new Number(event.result.attributes.attribute[0].stringValue);
        a[1] = new Number(event.result.attributes.attribute[1].stringValue);
        a[2] = new Number(event.result.attributes.attribute[2].stringValue);
        a[3] = new Number(event.result.attributes.attribute[3].stringValue);
        a[4] = new Number(event.result.attributes.attribute[4].stringValue);
        this.alarmRatioBar.setCountValues(a);
        this.alarmRatioBar.CaptionText = this.captionText;
        this.refreshTimer.start();
      }
      private function refreshTimerHandler(event : TimerEvent) : void
      {
        this.refreshTimer.stop();
        this.alarmRatioBar.CaptionText = "Refreshing...";
        createSession(this.rpcUsername, this.rpcPassword);
      }
      private function configureComponentFromQueryString() : void
      {
        this.rpcUsername = mx.core.Application.application.parameters.Username;
        this.rpcPassword = mx.core.Application.application.parameters.Password;
        this.rpcIdentity = mx.core.Application.application.parameters.Identity;
        this.captionText = mx.core.Application.application.parameters.CaptionText;
        this.alarmRatioBar.CaptionText = captionText;
        this.alarmRatioBar.CaptionFontSize =
mx.core.Application.application.parameters.CaptionFontSize;
        this.refreshInterval = new
Number(mx.core.Application.application.parameters.RefreshInterval);
       this.refreshInterval = isNaN(refreshInterval) ? DEFAULT_REFRESH_INTERVAL :
refreshInterval;
      }
      private function main() : void
      {
        configureComponentFromQueryString();
        this.refreshTimer = new Timer(this.refreshInterval * 1000);
        this.refreshTimer.addEventListener(TimerEvent.TIMER, refreshTimerHandler);
        createSession(this.rpcUsername, this.rpcPassword);
      }
    ]]>
  </mx:Script>
  <mosh:IAlarmRatioBar
    id="alarmRatioBar"
    OkayCount="10"
    InfoCount="10"
    MinorCount="10"
    MajorCount="10"
```

```
    CriticalCount="10"
    CaptionText="Alarm Ratios"
    CaptionColor="#ffffff"
    CaptionFontFamily="Tahoma"
    CaptionFontSize="48"
    CaptionAlign="center"
    FrameColor="#ffffff"
    FrameThickness="5"
    ShowShadows="true"
    width="100%"
    height="100%"
  />
  <mx:Label id="debugOutput"/>
</mx:Application>
```

7.3 Example: Alarm Ratio Bar Component

Source for Alarm Ratio Bar component:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas
  xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="initComponent()"
x="0" y="0" width="100%" height="100%" >
  <mx:Script>
    <![CDATA[
      private var _stageWidth : Number;
      private var _showShadows : Boolean;
      private var _okayCount : Number;
      private var _infoCount : Number;
      private var _minorCount : Number;
      private var _majorCount : Number;
      private var _criticalCount : Number;
      private var _totalCount : Number;

      [Bindable]

      private var _okaySkinImage : String = "images/ratioBarOkay.jpg";

      [Bindable]

      private var _infoSkinImage : String = "images/ratioBarInformational.jpg";

      [Bindable]

      private var _minorSkinImage : String = "images/ratioBarMinor.jpg";

      [Bindable]

      private var _majorSkinImage : String = "images/ratioBarMajor.jpg";
      [Bindable]
      private var _criticalSkinImage : String = "images/ratioBarCritical.jpg";
      private function initComponent() : void
      {
        this._okayCount = this.OkayCount;
        this._infoCount = this.InfoCount;
        this._minorCount = this.MinorCount;
        this._majorCount = this.MajorCount;
        this._criticalCount = this.CriticalCount;
        this._totalCount = this._okayCount;
        this._totalCount += this._infoCount;
        this._totalCount += this._minorCount;
        this._totalCount += this._majorCount;
        this._totalCount += this._criticalCount;
        this.drawComponent();
      }
      public function setCountValues(values : Array) : void
      {
```

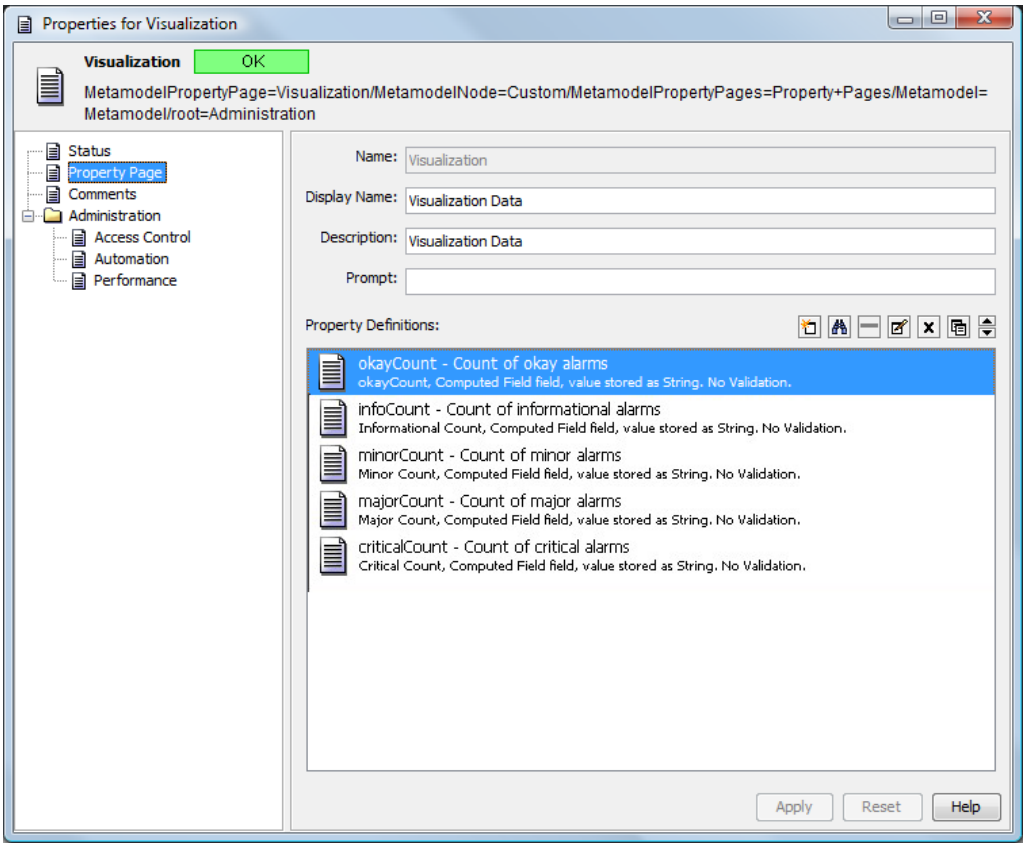
```
  this._okayCount = values[0];
  this._infoCount = values[1];
  this._minorCount = values[2];
  this._majorCount = values[3];
  this._criticalCount = values[4];
  this._totalCount = this._okayCount;
  this._totalCount += this._infoCount;
  this._totalCount += this._minorCount;
  this._totalCount += this._majorCount;
  this._totalCount += this._criticalCount;
  this.drawComponent();
}
private function getSeverityBarWidth(count : Number) : Number
{
  var pixels : Number;
  pixels = this._graphCanvas.width / this._totalCount;
  return count * pixels;
}
private function drawComponent() : void
{
  var okayWidth : Number;
  var infoWidth : Number;
  var minorWidth : Number;
  var majorWidth : Number;
  var criticalWidth : Number;
  this._stageWidth = this._graphCanvas.width;
  okayWidth = getSeverityBarWidth(this._okayCount);
  infoWidth = getSeverityBarWidth(this._infoCount);
  minorWidth = getSeverityBarWidth(this._minorCount);
  majorWidth = getSeverityBarWidth(this._majorCount);
  criticalWidth = getSeverityBarWidth(this._criticalCount);
  if (okayWidth != 0)
  {
    this._okaySkin.x = 0;
    this._okaySkin.y = 0;
    this._okaySkin.width = okayWidth;
    this._okaySkin.visible = true;
  }
  else
  {
    this._okaySkin.visible = false;
  }
  if (infoWidth != 0)
  {
    this._infoSkin.x = okayWidth;
    this._infoSkin.width = infoWidth;
    this._infoSkin.visible = true;
  }
  else
  {
    this._infoSkin.visible = false;
  }
  if (minorWidth != 0)
  {
    this._minorSkin.x = okayWidth + infoWidth;
    this._minorSkin.width = minorWidth;
    this._minorSkin.visible = true;
  }
  else
  {
    this._minorSkin.visible = false;
  }
  if (majorWidth != 0)
  {
    this._majorSkin.x = okayWidth + infoWidth + minorWidth;
    this._majorSkin.width = majorWidth;
    this._majorSkin.visible = true;
  }
  else
  {
```

```
            this._majorSkin.visible = false;
          }
          if (criticalWidth != 0)
          {
            this._criticalSkin.x = okayWidth + infoWidth + minorWidth + majorWidth;
            this._criticalSkin.width = criticalWidth;
            this._criticalSkin.visible = true;
          }
          else
          {
            this._criticalSkin.visible = false;
          }
          if (this._showShadows)
          {
            showShadows();
          }
          else
          {
            hideShadows();
          }
        }
        private function hideShadows() : void
        {
          this._okaySkin.filters = [];
          this._infoSkin.filters = [];
          this._minorSkin.filters = [];
          this._majorSkin.filters = [];
          this._criticalSkin.filters = [];
          this._graphCanvas.filters = [];
        }
        private function showShadows() : void
        {
          this._okaySkinShadow.strength = 1;
          this._infoSkinShadow.strength = 1;
          this._minorSkinShadow.strength = 1;
          this._majorSkinShadow.strength = 1;
          this._criticalSkinShadow.strength = 1;
          this._graphCanvasShadow.strength = 1;
        }
        public function set ShowShadows(state : Boolean) : void
        {
          _showShadows = state;
        }
        public function get ShowShadows() : Boolean
        {
          return _showShadows;
        }
      ]]>
    </mx:Script>
    <mx:Image id="_okaySkin" source="{_okaySkinImage}" scaleContent="true"
maintainAspectRatio="false" height="100%">
      <mx:filters>
        <mx:DropShadowFilter id="_okaySkinShadow" distance="4" angle="90"
blurX="8" blurY="8" quality="4" />
      </mx:filters>
    </mx:Image>
    <mx:Image id="_infoSkin" source="{_infoSkinImage}" scaleContent="true"
maintainAspectRatio="false" height="100%" >
      <mx:filters>
        <mx:DropShadowFilter id="_infoSkinShadow" distance="4" angle="90"
blurX="8" blurY="8" quality="4" />
      </mx:filters>
    </mx:Image>
    <mx:Image id="_minorSkin" source="{_minorSkinImage}" scaleContent="true"
maintainAspectRatio="false" height="100%" >
      <mx:filters>
        <mx:DropShadowFilter id="_minorSkinShadow" distance="4" angle="90"
blurX="8" blurY="8" quality="4" />
      </mx:filters>
    </mx:Image>
```

```
  <mx:Image id="_majorSkin" source="{_majorSkinImage}" scaleContent="true"
maintainAspectRatio="false" height="100%" >
    <mx:filters>
      <mx:DropShadowFilter id="_majorSkinShadow" distance="4" angle="90"
blurX="8" blurY="8" quality="4" />
    </mx:filters>
  </mx:Image>
  <mx:Image id="_criticalSkin" source="{_criticalSkinImage}" scaleContent="true"
maintainAspectRatio="false" height="100%" >
    <mx:filters>
      <mx:DropShadowFilter id="_criticalSkinShadow" distance="4"     angle="90"
blurX="8" blurY="8" quality="4" />
    </mx:filters>
  </mx:Image>
  <mx:Label id="_graphText" text="{CaptionText}" fontFamily="{CaptionFontFamily}"
fontSize="{CaptionFontSize}" fontWeight="bold" textAlign="{CaptionAlign}"
color="{CaptionColor}"    alpha="0.9" left="24" top="24"
right="{this._graphCanvas.width - 24}" bottom="{this._graphCanvas.height - 24}" >
    <mx:filters>
      <mx:DropShadowFilter distance="4" angle="90" blurX="8" blurY="8" quality="5"
/>
    </mx:filters>
   </mx:Label>
  <mx:Canvas id="_graphCanvas" x="0" y="0" width="100%" height="100%"
borderThickness="8" borderColor="white" >
    <mx:filters>
      <mx:DropShadowFilter id="_graphCanvasShadow" distance="0"       angle="90"
blurX="8" blurY="8" quality="5" alpha="0.25" />
    </mx:filters>
  </mx:Canvas>
  <mx:Number id="OkayCount">10</mx:Number>
  <mx:Number id="InfoCount">20</mx:Number>
  <mx:Number id="MinorCount">30</mx:Number>
  <mx:Number id="MajorCount">40</mx:Number>
  <mx:Number id="CriticalCount">50</mx:Number>
  <mx:Number id="FrameColor">#ffffff</mx:Number>
  <mx:Number id="FrameThickness">0x08</mx:Number>
  <mx:String id="CaptionText">Element Name</mx:String>
  <mx:Number id="CaptionColor">#ffffff</mx:Number>
  <mx:String id="CaptionFontFamily">Tahoma</mx:String>
  <mx:Number id="CaptionFontSize">40</mx:Number>
  <mx:String id="CaptionAlign">center</mx:String>
</mx:Canvas>
```

7.4 Example: Metamodel Property Page

The Alarm Ratio Bar implementation example uses an Operations Center metamodel property page with five computed fields providing the alarm counts for each of the five main alarm conditions.



The Web 2.0 Connect remote method for element attributes returns the requested element attributes as a collection <attribute/> tags with each attribute having a <stringValue/> tag for the actual value and a <type/> tag for the data type of the attribute:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<ns2:attributes xmlns:ns2="http://api.bsm.netiq.com/moweb2cn">
<identity>identity:dname:7PfStLXR6bwONHfRsuzGMnXScr3RtLkT7ClSczlT3rFScTXRcbwONHfRs
vp</identity>
  <attribute>
    <stringValue>50</stringValue>
    <type>StringType</type>
  </attribute>
<attribute>
    <stringValue>5</stringValue>
    <type>StringType</type>
  </attribute>
<attribute>
    <stringValue>50</stringValue>
    <type>StringType</type>
  </attribute>
<attribute>
    <stringValue>30</stringValue>
    <type>StringType</type>
  </attribute>
<attribute>
    <stringValue>10</stringValue>
    <type>StringType</type>
  </attribute>
</ns2:attributes>
```