Scripting Guide

Operations Center 5.6

June 2015

Legal Notices

Contents

About This Guide

Operations Center extends its reach into the scripting engine to allow for customization and definition of business-specific behavior. It does this by using NOC Script, which is an extension of the ECMAScript Version 3 scripting language.

For ECMAScript information and objects, see the ECMAScript Language Reference (http://www.webreference.com/javascript/reference/ECMA-262/E262-3.pdf)

The *Scripting Guide* provides instructions to use NOC Script to customize and define business-specific behavior:

Audience

This guide is intended for Operations Center system administrators using the NOC Script language.

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 *Scripting Guide* and a complete list of publications supporting Operations Center, visit our Online Documentation Web Site at Operations Center 5.6 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%5B%5D=Operations_Center): A collection of in-depth technical articles.
- NetIQ Support Forums (https://forums.netiq.com/forumdisplay.php?26-Operations-Center): A Web location where product users can discuss NetIQ product functionality and advice with other product users.

Technical Support

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

Use these resources for support specific to Operations Center:

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

Documentation Conventions

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

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

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

1 Introduction

Operations Center extends its reach into the scripting engine to allow for customization and definition of business-specific behavior. It does this by using NOC Script, which is an extension of the ECMAScript scripting language. In turn, ECMAScript is a standardization of JavaScript*.

1.1 About NOC Script

NOC Script is an embedding of ECMAScript within the Operations Center server and client software, and extends EMCAScript Version 3. For ECMAScript information and objects, see the ECMAScript Language Reference (http://www.webreference.com/javascript/reference/ECMA-262/E262-3.pdf)

Use NOC Script in many areas within the Operations Center interface:

- Set a series of actions to take when an adapter starts or stops
- Perform powerful sorting and matching operations as part of Operations Center Automation Events, allowing complex responses to alarms or events reported to Operations Center
- Launch a program on the server when certain conditions are met
- Launch a program from the browser by using a menu option
- Query the Operations Center database
- Send e-mails or pages when certain events occur, based on complex criteria
- Extend the functions of Adapter Hierarchy Files

1.2 NOC Script, JavaScript, and ECMAScript

JavaScript* is a popular scripting engine used by virtually every Web browser. Despite its name, JavaScript has little to do with the Java* programming language and environment. While the two environments do share some syntax, more differences than similarities exist between JavaScript and the Java programming language.

When competing versions of JavaScript-like languages threatened to break the scripting language into competing proprietary versions, the Ecma International standards group created ECMAScript as a vendor-neutral standard. NOC Script is an extension of this language.

For ECMAScript information and objects, see the ECMAScript Language Reference (http://www.webreference.com/javascript/reference/ECMA-262/E262-3.pdf)

A similarity does exist in the way a browser and Operations Center each interact with JavaScript. A browser exposes several top-level objects to the JavaScript engine before executing the script. Notably, the *navigator* and *document* objects represent gateways into the browser and HTML document, respectively. A script author uses these objects to “script” the HTML page, adding interactivity. Similarly, Operations Center exposes an object to the scripting environment to allow a script author to interact with Operations Center elements.

1.3 Learning JavaScript

This guide assumes you are familiar with JavaScript and its use in an embedding application. The O'Reilly JavaScript book is a good start, but please note that the information about browsers in this and most other JavaScript books is not relevant to NOC Script. This topic covers using JavaScript to leverage building dynamic Web pages, which is only marginally related to NOC Script as to hosting a JavaScript environment inside an application.

The latest O'Reilly book information, *JavaScript: The Definitive Guide (http://www.oreillynet.com/cs/catalog/view/au/156?x-t=book.view)* by David Flanagan, can be found at http://www.oreilly.com (http://www.oreilly.com).

2 Creating and Debugging Scripts

Scripts can be used with various features in Operations Center, including automations and algorithms. Operations Center ships with a library of default scripts. Scripts can be customized to suit your needs and new scripts can be added to the Script Library.

For more information about default scripts that ship with the Script Library, see Appendix A, “The Script Library,” on page 83.

2.1 Adding a Script to the Script Library

The Script Library allows scripts to be edited and reused among various product features, including automations and algorithms. A script can be added to the Script Library simply by saving it to the `/OperationsCenter_install_path/database/scripts` directory. It then surfaces automatically as an option when scripts are defined or selected.

Operations Center no longer supports scripts that were written using interfaces that existed prior to the publication of interfaces based on the Extensible Services effort in the 3.1 release.

The browser provides a simple way to create and debug new scripts and save them to the Script Library.

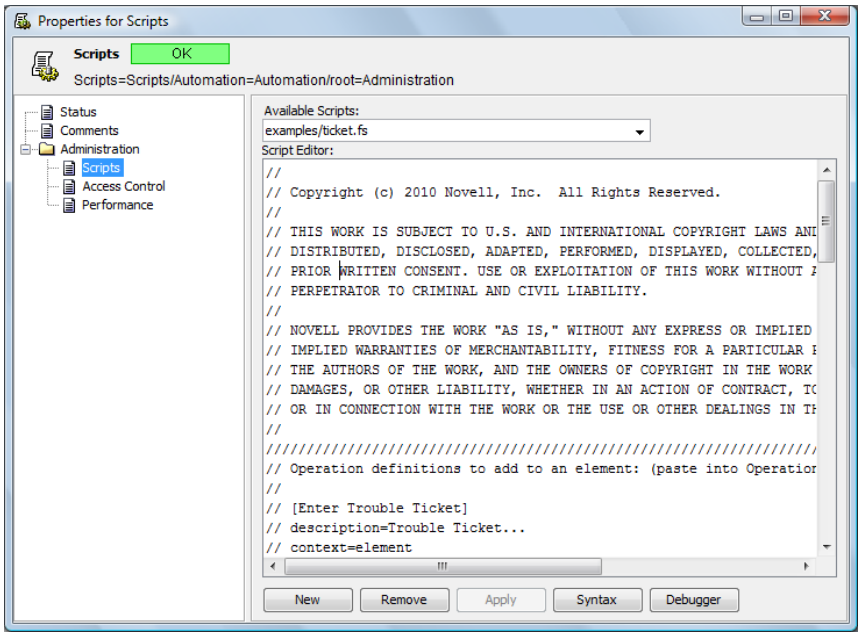
To create a script:

1. In the Explorer pane, expand *Administration* > *Automation*.
2. Right-click the *Scripts* element and select *Properties* to open the *Status* property page.
3. In the left pane, expand *Administration*.
4. Right-click *Scripts* and select *Properties*.

   The *Properties* dialog box opens.
5. Type a name for the script in the *Scripts* text box.

6 Type the script text in the *Script Editor* text box.

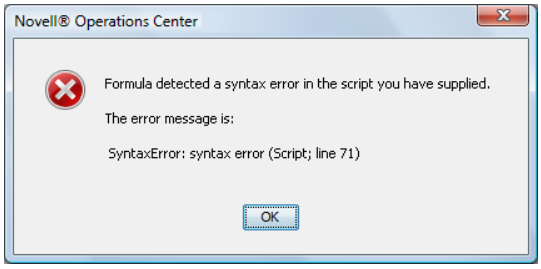
Text can be copied and pasted into this window from the Operations Center Script Debugger or another text editor. See Section 2.2, “Using the Script Debugger,” on page 13.



7 Click *Syntax* to display one of the following messages:

- If the syntax of the newly created script is correct, the message indicates `The script has no syntax errors`.
- If the script contains syntax errors, a message similar to the following displays. All errors found by the syntax checker display in this message, with line and column numbers to help locate the syntax error.

For example:



8 To evaluate the script, use the Operations Center Script Debugger.

For more information, see Section 2.2, “Using the Script Debugger,” on page 13.

9 Click *Apply*.

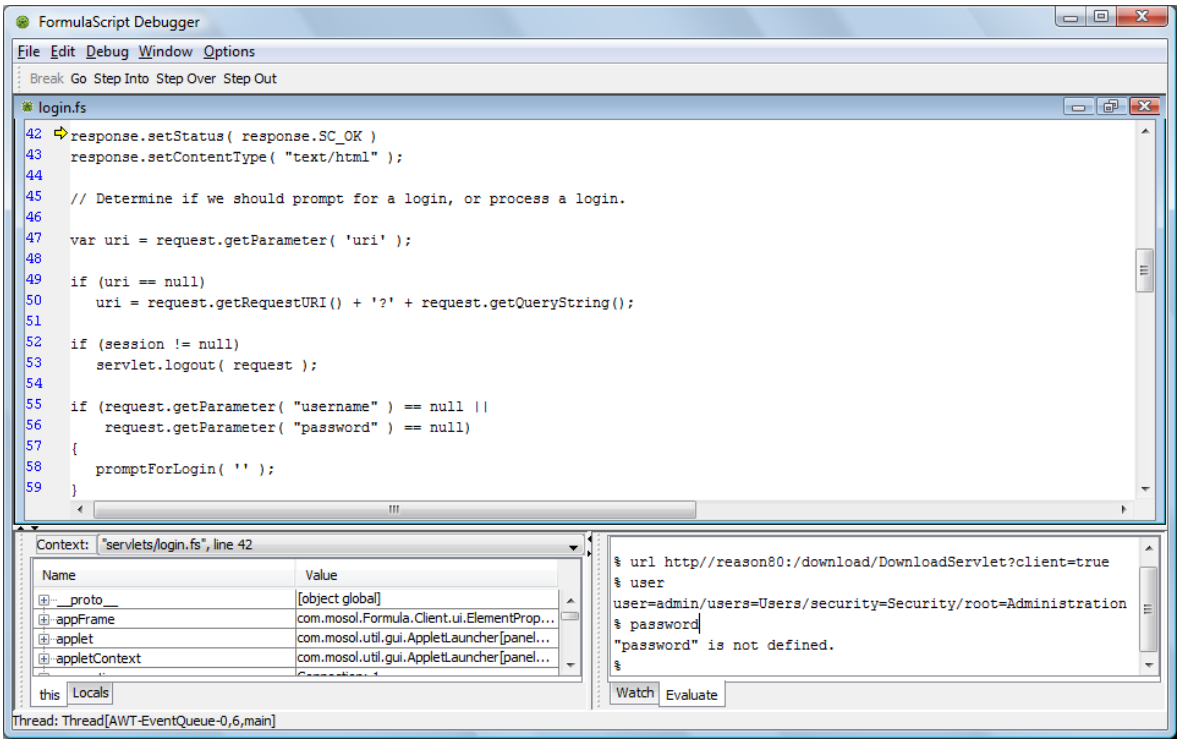
The new script is saved to the `/OperationsCenter_install_path/database/scripts directory` and can be accessed from anywhere that scripts are accepted.

10 Click *New* to clear the *Script Editor* and create another script.

2.2 Using the Script Debugger

The Operations Center Script Debugger can be used to develop and test scripts. This section describes accessing the Operations Center Script Debugger from the *Scripts* tab as a continuation from the previous steps in Section 2.1, “Adding a Script to the Script Library,” on page 11. See Section 2.3, “Additional Options for Running the Script Debugger,” on page 15 for other ways to access the debugger.

Figure 2-1 *The Debugger Window Showing Three Commands on the Evaluate Tab*



Script code displays in the debugger with numbered lines for identification and navigation.

Table 2-1 provides an overview and description of debugger features.

Table 2-1 *Debugger Features*

Debugger Feature	Description
Variable Display	Shows all the variables that are “in scope” for the current object. Even if your script declares local variables by using the var keyword, the debugger displays them as global variables if they are declared at the global scope. There is no way to declare a local variable at the global scope.

Debugger Feature	Description
Evaluate Tab	Type a command to execute and evaluate. Figure 2-1 shows how three variables were executed by the user: a URL, user account, and password.

The following topics cover debugging scripts using the debugger:

2.2.1 Debugging a Script Using the Debugger

To debug a script using the debugger:

1. From the *Scripts* tab, select *Debugger*.

   The *Debugger* opens and displays the script.
2. When a script loads, select to step through the script code using one of the following toolbar buttons:

   Step Into: If the current line of execution involves a function invocation, traces the invocation into that function.

   Step Over: If the current line of execution involves a function invocation, traces the invocation over that function, returning execution control to the Debugger after that function has completed.

   Step Out: If the current line of execution is inside a function invocation, traces the invocation out that function, after it has completed.
3. To set a breakpoint to pause the script, click the left margin on the desired line of code.

   A breakpoint symbol displays.
4. Click again to remove the breakpoint.
5. Click the *Go* button to evaluate the script.

   The Operations Center Script Debugger evaluates the script syntax and errors display in a dialog box.
6. Click *File* > *Exit* to close the debugger.

2.2.2 Automatically Running Debugger Scripts

To run the debugger automatically for a specific script or to open the debugger for all scripts at runtime, click the *Options* menu from the Debugger console and select one of the following options:

- **Never:** Sets the debugger to never open automatically when a script runs.
- **All Scripts:** Automatically opens the debugger for every script at runtime.
- **Prompt on Each Script:** Prompts to open the debugger for every script at runtime.
- **Specific Script:** Opens the Debugger only for the specified scripts.

2.3 Additional Options for Running the Script Debugger

The following sections cover additional options for running the script debugger:

2.3.1 Running the Debugger from the Command Line

From the command line, run the `fscript` command from */OperationsCenter_install_path/database/scripts* to launch the debugger. Table 2-2 describes debug command options.

Table 2-2 *Debug Commands*

Command	Result
`fscript -debug`	Launches the debugger
`Fscript –debug –f scriptname.fs`	Opens a script from the */OperationsCenter_install_path/database/scripts* directory in the debugger.

2.3.2 Adding Settings in Property Files to Run Scripts in the Debugger

The debugger can be set up to open all scripts run within the environment.

To update property files for the debugger:

1. For server-side script debugging, do the following:
   - **1a** Add the following properties to the `Formula.custom.properties` file:

     ```
     Script.debug=true
     ```
   - **1b** Restart Operations Center.
   - **1c** Launch the Operations Center console and open the debugger. All scripts open in the debugger as they are run.

     Note that only scripts run on that Operations Center server will open.
2. For client-side script debugging, do the following:
   - **2a** Add the following properties to the `applet_params.xml` file:

     ```
     <param name="Script.debug" value="true"/>
     ```
   - **2b** Relaunch the Operations Center console and open the debugger. All scripts open in the debugger as they are run.

2.3.3 Instrumenting a Script for Debugging

Adding a debug comment to a script can be used to automatically launch the debugger when the script executes. Be sure to remember to remove a `// @debug on` comment before sending the script to production. Table 2-3 lists the debug comment options.

Table 2-3 *Debug Comments for Script Files*

Comment Code	Result
// @debug on	The debugger launches when the script runs.
// @debug off	The debugger does not launch when the script runs.

3 Scripting Conventions

NOC Script uses common syntax and methods, similar to those used by other scripting languages.

3.1 Property Syntax

NOC Script objects generally have a set of associative properties tied to them. Access these properties by name using the dot (.) notation:

```
writeln( formula.Elements.name )
```

In addition to this syntax, access properties by using array notation:

```
writeln( alarm['class'] )
```

This is useful because sometimes, as in the previous example, a property name might be a reserved word for the language or might be a property name with an invalid character, such as a colon (:) or a space.

3.2 Loop Syntax

The scripting language supports looping over the properties of an object by using the following syntax:

```
for( var p in alarm.properties )
   writeln( p + '=' + alarm[p] )
```

3.3 Declaring Objects

Use one of the following methods to declare a new object in the scripting environment:

- Declare an object and assign it properties:

```
var obj = new Object()
obj.foo('bar')
obj.num('45')
obj.printit = function() { writeln( 'foo=' + this.foo + ', num=' + this.num ) }
```

- Alternatively, declare an object with properties:

```
var obj =
{
    foo: 'bar',
    num: 45,
    printit: function(){writeln( 'foo=' + this.foo + ', num=' + this.num ) }
}
```

3.4 Declaring Arrays

Declare arrays using one of the following methods:

- Declare an array and assign individual entries for values:

```
var arr = new Array()
arr[0] = 'a'
arr[1] = 'b'
arr[2] = 'c'
```

- Declare an array with values:

```
var arr = [ 'a', 'b', 'c' ]
```

To extend an array, use the following convention:

```
arr[ arr.length ] = anotherValue
```

3.5 Implementing a Java Interface

Scripts written in NOC Script can implement a Java* interface, because the environment is hosted under Java. This uses a special syntax, similar to Java “inner class” declarations, and similar to the object declaration syntax introduced above. An example:

```
var runnable = new java.lang.Runnable()
{
    run: function()
    {
        writeln( 'Yep, running on a thread, all right!' )
    }
}
var thread = new java.lang.Thread( runnable )
thread.start()
```

3.6 Exception Handling: try/catch/throw

Like Java, NOC Script contains a mechanism to catch exceptions thrown by a block of code. Unlike Java, however, exceptions are not declared by type. A single catch clause catches the thrown exceptions:

```
try
{
   if( somethingBadHappened )
      throw 'get outta here'
}
catch( Exception )
{
    writeln( 'Exception was thrown: ' + Exception )
}
```

To catch a specific exception, use the following exception catch syntax:

```
try
{
    // Open our file.
    var f = new java.io.FileInputStream( 'somefile.txt' )
    // …
}
catch( IOException if ( IOException instanceof java.io.IOException ) )
{
    writeln( 'I/O exception was thrown: ' + IOException )
}
```

4 The NOC Script Object Model

The convention of populating a scripting engine with application-specific extensions is not unique to Operations Center. In standard Internet browsers, a gateway or application-defined object is found in the browser's navigator object. This object holds methods and properties that a script can use when it is run from a Web page. Similarly, the actual Web page's document object model, or DOM, is exposed in the browser's document object.

Operations Center adopts this convention, and exposes an object within the scripting engine that can interact with Operations Center.

In addition, Operations Center elements, alarms, and other objects are dynamically exposed to the scripting engine to allow a script author to access properties, events, and methods of Operations Center objects themselves.

4.1 The Formula Object

The `Formula` object appears in all scripts run within Operations Center. This object is a top-level placeholder object that can interact with Operations Center. From this object, you can locate elements, manage sessions, log messages to log files, and perform a host of other scripting actions.

4.2 Top-Level Elements in Hierarchy Structure

Top-level elements are exposed within the object, which can navigate and interact with Operations Centers's objects within its namespace.

The elements in Table 4-1 on page 22 are considered top-level and are always present. If a session is associated with a running script, access to these elements (and all contained elements) is restricted according to the access control policies set on the elements themselves. Scripts adhere to the same protocol as the Operations Center console software.

Table 4-1 The Top-Level Elements in the Operations Center Hierarchy

Top-Level Element Name	Description
Enterprise	This is the root element in Operations Center. It is the topmost element within Operations Center and every object has the *Enterprise* element as its ultimate ancestor.
Elements	Adapter-produced elements, organized per adapter, display under the *Elements* element tree.
Services	*Locations* and *Service Models* hierarchies display under the *Services* element tree.
Administration	Administrative elements display under the *Administration* element tree. These elements include adapters, the server object, sessions, access control, operation definitions, and automation.

4.3 Session Functions (formula.login)

The session functions log in to Operations Center, or they cancel or close a session:

- Section 4.3.1, “login(),” on page 22
- Section 4.3.2, “logout(),” on page 22

4.3.1 login()

The login() function logs in to Operations Center. This function takes three arguments:

- The Operations Center server’s host and port (Web) and protocol (the default is http.)
- The user name and the password for the session.
- A timeout, in seconds, to wait for establishing the connection.

This function returns either a valid session element or a string or exception if an error occurs. For example:

```
var session = formula.login( 'localhost', 80, 'http', 'admin', 'formula', 60 )
```

4.3.2 logout()

The logout() function cancels or closes a session obtained from the login() function. For example:

```
formula.logout( session )
```

4.4 Logging Functions (formula.log)

- Section 4.4.1, “log,” on page 23
- Section 4.4.2, “logCategory,” on page 23
- Section 4.4.3, “Creating a New Log Instance,” on page 23

4.4.1 log

The log object contained under the formula object is a gateway for the logging activities of the Operations Center server or client, depending on which one invokes the script:

- If on the client, log messages are written to the Client's java console output.
- If on the server, the log messages are handled by the server's log4j implementation. For more information regarding server trace logs and log settings, see the *Operations Center 5.6 Server Configuration Guide*.

Four methods that can log messages:

- **debug():** Send a debug message to the log.
- **info():** Send an informational message to the log.
- **warn():** Send a warning message to the log.
- **error():** Send an error message to the log.

For example:

```
formula.log.info( 'An informational message' )
formula.log.error( 'An error happened' )
```

4.4.2 logCategory

The logCategory of a script contains the name of the logging category that produces messages for the script. This property can be obtained or set.

To change or switch the logging category of a script, simply set this variable to another string. For example:

```
formula.logCategory = 'Script.MyScript'
formula.log.warn( 'Something bad might have just happened' )
```

4.4.3 Creating a New Log Instance

It might be necessary to create a separate log instance than the one provided for a script. To do this, call the getCategory() method on the log object:

```
formula.log.getInstance( 'Another.Category' )
```

Use a period to separate categories. For example:

```
var otherlog = formula.log.getInstance( 'Script.Other' )
otherlog.info( 'Now logging to another log category' )
```

4.5 Utility Functions (formula.util)

The formula.util object contains a number of helper or utility functions that scripts can use. In many cases, an analog or similar facility exists within JavaScript* itself.

4.5.1 String searchAndReplace (String s, String needle, String haystack)

Replace an instance of one string within String s with another string.

JavaScript defines a replace function for any string that can be used similarly. For example:

```
var s = 'bill me'
writeln ( formula.util.searchAndReplace( s, 'me', 'you' ) )
```

4.5.2 String encodeURL (String url)

It is possible to URL-encode a string to allow transmission as text. Operations Center distinguished names are components of URL-encoded strings.

JavaScript defines an encodeURL function that takes any string that can be used similarly. For example:

```
try {
var baseDName = 'element.dname' // Assumes element in-scope
var className = 'Router:Cisco'
var dname = formula.util.encodeURL( className ) + '=' + formula.util.encodeURL(
name ) + '/' + baseDName
}
```

4.5.3 String decodeURL (String url)

It is possible to URL-decode a string to allow transmission as text. Operations Center distinguished names are components of URL-encoded strings.

JavaScript defines a decodeURL function that takes any string that can be used similarly. For example:

```
js> var s = 'String to encode: @#$%^&*'
js> var encoded = formula.util.encodeURL( s )
js> encoded
String+to+encode%3A+%40%23%24%25%5E%26*
js> formula.util.decodeURL( encoded )
String to encode: @#$%^&*
js>
```

4.5.4 String encodeXML (String s)

It is possible to encode a string as allowed within an XML document. For example, XML documents cannot contain the & (ampersand) character. This function turns the & string into the string & which is suitable for an XML document fragment. For example:

```
js> var s = 'String to encode: &<>"'
js> formula.util.encodeXML( s )
String to encode: &amp;&lt;&gt;&quot;
js>
```

4.5.5 String[ ] breakOnTokens (String s, String tok)

The breakOnTokens function subdivides one string and returns an array of strings. The tok parameter is a string representing the break pattern. For example:

```
js> s = 'one|two|three'
one|two|three
js> a = formula.util.breakOnTokens( s, '|' )
[Ljava.lang.String;@152c4d9
js> a[0]
one
js> a[1]
two
js>
```

4.5.6 String[ ] breakOnCommas (String s)

The breakOnCommas function subdivides a string using the comma character as the token. For example:

```
js> s = 'one,two,three'
one,two,three
js> a = formula.util.breakOnCommas( s )
[Ljava.lang.String;@f99ff5
js> a[1]
two
```

4.5.7 InputStream captureOutputStream (String commandLine, String [ ] environment) throws an Exception

This function uses the supplied command line to execute the supplied program and return the output as a java.io.InputStream. It provides an environment array for supplying environment variables to the program when it runs. For example:

```
var stream = formula.util.captureOutputStream( 'cmd /c echo %FOO%', [ 'foo=bar' ] )
writeln( new java.lang.String( formula.util.toByteArray( stream ) ) )
```

4.5.8 InputStream captureOutputStream (String commandLine) throws an Exception

This function uses the supplied command line to execute the supplied program and return the output as a java.io.InputStream. For example:

```
var stream = formula.util.captureOutputStream( 'cmd /c echo %FOO%' )
writeln( new java.lang.String( formula.util.toByteArray( stream ) ) )
```

4.5.9 String captureOutputString (String commandLine, String [ ] environment) throws an Exception

This function uses the supplied command line to execute the supplied program and return the output as a string. It provides an environment array for supplying environment variables to the program when it runs. For example:

```
js> var s = formula.util.captureOutputString( 'cmd /c echo %FOO%', [ 'foo=bar' ] )
js> s
bar
```

4.5.10 String captureOutputString (String commandLine) throws an Exception

This function uses the supplied command line to execute the supplied program and return the output as a string. For example:

```
js> var s = formula.util.captureOutputString( 'cmd /c echo %FOO%' )
js> s
%FOO%
```

4.5.11 String escapeRegExp (String regexp)

Regular expressions contain matching criteria, such as the period (.) and asterisk (*) characters. This function replaces these regular expression characters with escaped (\) string sequences to enable a literal match. For example:

```
js> s = 'dir *.*'
dir *.*
js> t = formula.util.escapeRegExp( s )
dir \*\.\*
```

4.5.12 void copyStream (InputStream input, OutputStream output) throws an IOException

This function copies one java.io.InputStream to an instance of a java.io.OutputStream. This function can throw a java.io.IOException, so place appropriate try/catch directives around its use. For example:

```
js> input = new java.io.FileInputStream( 'C:\\Config.Sys' )
java.io.FileInputStream@2d9c06
js> output = new java.io.FileOutputStream( 'C:\\Config.Sys.Copy' )
java.io.FileOutputStream@7b6889
js> formula.util.copyStream( input, output )
js> input.close()
js> output.close()
```

4.5.13 byte[ ] toByteArray (InputStream input) throws an IOException

This function returns the contents of a java.io.InputStream as a byte array. A byte array can be a constructor argument to java.lang.String, so it is possible to use this function in combination to return the contents of a stream as a string. This function can throw a java.io.IOException, so place appropriate try/catch directives around its use. For example:

```
js> input = new java.io.FileInputStream( 'D://OperationsCenter_install_path/
database//Adapters.ini' )
java.io.FileInputStream@99681b
js> ba = formula.util.toByteArray( input )
[B@181edf4
js> s = new java.lang.String( ba )

[Tivoli T/EC(r) on reason, iv]
AdapterInstanceId=3
ElementsTimeout=300
EventConsoleName=@Formula
AcknowlegeAvailable=true
SeedFile=
SyncClass=TEC_Sync
AlarmColumns=Status,Class,Description
AckAffectsCondition=true
Script.onInitialized=
SeverityMapping=Fatal=Critical;Critical=Critical;Minor=Minor;Warning=Major;Harmles
s=Informational;Unknown=Unknown
ClosedAlarmsTimeout=0
SuppressionTime=1800
```

```
CloseAvailable=true
HostsToMine=reason
TecORBPort=1576
Script.onStarted=
HierarchyFile=examples/TecHierarchy.xml
EventListenPort=54321
Script.onStopped=
StylesheetFile=
Class=com.mosol.Adapter.TEC.Adapter
MaxAlarms=500
WTDumperCommand=wtdumper -o DESC -dw "status<='20' AND severity>='20'"
SuppressAvailable=true
MiningLimit=2000
startOnStartup=false
```

4.5.14 String nameToFile (String name)

This function uses the candidate file name to change the string contents to an allowed file system compatible string. For example:

```
js> s = 'Make A Good File For This: %^*()!@#$'
Make A Good File For This: %^*()!@#$
js> formula.util.nameToFile( s )
Make_A_Good_File_For_This__%^*()!@#$
js>
```

4.5.15 void center (java.awt.Window w)

This function centers the supplied java.awt.* resource (window or derivative) on the screen. For example:

```
js> f = new java.awt.Frame( 'My Frame' )
java.awt.Frame[frame0,0,0,0x0,invalid,hidden,layout=java.awt.BorderLayout,title=My
Frame,resizable,normal]
js> f.setSize( 600, 400 )
js> formula.util.center( f )
js> f.setVisible( true )
js> f.addWindowListener( new java.awt.event.WindowAdapter() {
   windowClosing: function( evt ) { f.setVisible( false ) } } )
```

Close this window by clicking the *Close* button in the title bar frame after it displays. This example also shows how to add a java/awt window listener by using the inner class analog in JavaScript.

4.5.16 Object makeRemote(Object obj) throws JavaScriptException

This function manufactures a remote proxy for the supplied JavaScript object, suitable for sending to a remote target. Scripting uses this between client and server scripts to send callback and other remote reference code that can be accessed remotely. For example:

```
js> callback =
{
   callme: function() { writeln( 'Thanks for calling' ) }
}
[object Object]
js> remoted = formula.util.makeRemote( callback )
js>
```

4.5.17 void notify (Object signal)

JavaScript does not contain the concept of Java synchronization. Use this function to allow the standard notify method for any supplied object. For example:

```
var o = new java.lang.Object()
formula.util.notify( o )
```

4.5.18 void notifyAll (Object signal)

JavaScript does not contain the concept of Java synchronization. Use this function to allow the standard notifyAll method for any supplied object. For example:

```
var o = new java.lang.Object()
formula.util.notifyAll( o )
```

4.5.19 void wait (Object signal, long timeout)

JavaScript does not contain the concept of Java synchronization. Use this function to allow the standard notify method for any supplied object. The timeout is the wait time in milliseconds. Set it to –1 to indicate an infinite wait. For example:

```
var o = new java.lang.Object()
formula.util.wait( o , -1 )
```

4.5.20 void page (pagerid, message, host, port)

This function sends an alphanumeric page to an SNPP (Simple Network Paging Protocol) paging gateway. Required arguments are pagerid and message. The host and port are the destination targets of the SNPP gateway. For more information about SNPP, go to http://www.snpp.info (http://www.snpp.info). For example:

```
formula.util.page( 3331122, 'Can you help me?', 'pagegw.net', 444 )
```

4.5.21 user

This is a property containing the name of the user who executes the script. For example:

```
writeln( formula.util.user )
```

4.5.22 class UString

UString is a helper class that obtains a unique string cached in a string pool. The `getString()` method obtains the string. Only one copy of the string exists in memory if this method is called. For example:

```
general = formula.util.UString.getString( 'general' )
class = formula.util.UString.getString( 'class' )
```

4.5.23 class ORB

The ORB class is the utility class that wraps the CORBA ORB concept into a CORBA-portable abstraction. This generally is not used in scripts, except to stringify and destringify remote object references. These are standard methods of the CORBA binding for Java as defined by the OMG. For example:

```
s = formula.util.ORB.init().object_to_string( session )
writeln( s )
```

4.5.24 class Postemsg

The Postemsg class sends a message to a Tivoli* T/EC RIM. Construct the class with the argument signature (String serverName, String cls, String adapter).

An instance of the Postemsg class has properties that can be set or retrieved:

- **serverName:** The name of the server to send the message.
- **message:** The message to send.
- **severity:** The severity of the message.
- **tecClass:** The T/EC message class.
- **adapter:** The T/EC adapter string.
- **port;** The known port for the RIM message listener.

An instance of the Postemsg class that is set for sending goes to the RIM using the `sendMessage()` method. For example:

```
var postemsg = new formula.util.Postemsg( 'qasun', 'LogFile_Base', 'Formula' )
postemsg.severity = 'HARMLESS'
postemsg.sendMessage( 'Something bad has happened; please check' )
```

4.5.25 class TelnetFrame

The TelnetFrame class opens a telnet session directly to network management systems, where supported. The *Administration* > *Server* > *Console Definitions* > *Default* element uses the following code to define a Console menu option for applicable NMS elements.

```
var consoleFrame = formula.util.TelnetFrame();
var params = java.util.Hashtable();
params.put( "targetName", element.getName() );
params.put( "address", host.toString() );
params.put( "port", port.toString() );
params.put( "targetIcon",  element.getLabel().getIcon() );
consoleFrame.setParams( params );
consoleFrame;
```

4.5.26 class ViewBuilder

The ViewBuilder class allows an XML document that conforms to the Operations Center views DTD to be processed through the Operations Center console View generator gateway within Operations Center. The only methods of significance to an instance of this class are:

- **buildFromFile(java.lang.String):** Process the supplied argument as a file name to the ViewBuilder gateway.
- **buildFromReader(java.io.Reader:)** Process the supplied argument as a character stream to the ViewBuilder gateway.

For example:

```
try
{
   var vb = new formula.util.ViewBuilder()
   vb.buildFromFile( '/myviewbuilder.xml' )
}
catch( Exception )
{
   formula.log.error( 'Could not build from view builder: ' + Exception )
}
```

4.6 Condition Functions (formula.conditions)

Operations Center element conditions and alarm severities have assigned values. Table 4-2 lists the default values.

Table 4-2 *Default Values for Element Condition and Alarm Severities*

Condition/Severity	Numeric Value
CRITICAL	1
MAJOR	2
MINOR	3
UNKNOWN	0
INFORMATIONAL	4
OK	5
UNMANAGED/INITIAL	6

Using NOC Script you can retrieve and set conditions and severities. The following two sections describe functions related to element condition and alarm severities:

- Section 4.6.1, “conditions,” on page 32
- Section 4.6.2, “severities,” on page 32

4.6.1 conditions

Operations Center elements allow for the retrieval and setting of their conditions. This property of the object exists as an associative array of conditions that can access the actual condition value used by Operations Center.

In the following example, we list all available conditions on elements in no specific order:

```
js> for( var p in formula.conditions )
   writeln( p )
0
UNKNOWN
2
4
3
1
5
6
7
UNMANAGED
8
CRITICAL
MINOR
USAGE_BUSY
9
INFORMATIONAL
USAGE_IDLE
OK
MAJOR
USAGE_ACTIVE
```

Both numeric and textual lookups can be performed using formula.conditions. For example:

```
js> formula.conditions.OK
OK
js> formula.conditions.MAJOR
MAJOR
js> formula.conditions[3]
MINOR
js>
```

4.6.2 severities

Operations Center's alarms allow for the retrieval and setting of severity. This property of the formula object exists as an associative array of severities which can be used to access the actual severity value used by Operations Center. The following example shows both numeric and textual lookups:

```
js> writeln( formula.severities.MINOR )
MINOR
js> writeln( formula.severities[3] )
MINOR
js> writeln( formula.severities.OK )
OK
js>
```

4.7 Navigating Relationships (formula.relations)

Operations Center elements allow for the navigation of relationships. This property of the object exists as an associative array of relations which can access the actual relation value used by Operations Center.

Values:

- **NAM:** The name or storage relationship. This is the primary relationship used by distinguished names.
- **ORG:** The organization or business view relationship.
- **MAP:** The location relationship (similar to ORG).

For example:

```
js> writeln( formula.relations.NAM )
NAM
js> writeln( formula.relations[1] )
ORG
js> writeln( formula.relations.ORG )
ORG
js>
```

4.8 formula.commands

The formula.commands element allows access to the server-side plug-in features of Operations Center, known as commands.

For example, to initialize the Suppression subsystem command:

```
if( ! state.suppression )
{
   state.suppression = formula.commands.Suppression( null )
   state.suppression.setOperationMatch( 'dnamematch:.*'  )
}
```

5 Element Functions

There are various mechanisms available to a script writer to interact with, query, and change Operations Center elements. Operations Center elements are exposed to the scripting engine in a way that allows access to properties, operations, and hierarchy in the system.

5.1 Understanding Distinguished Names

Each Operations Center element has a unique distinguished name, or dname, consisting of its path in the hierarchy and the individual ID (usually a name) and class components of each container in the hierarchy. Dnames are read from right to left, from least specific to most specific, unlike the well-known pattern of file system paths.

Distinguished names are persistent. After an element has a dname, it always retains that dname, even if the Operations Center server restarts.

Distinguished names have the individual textual portions of their components encoded in a URL format, so they can be transmitted over HTTP and other textual mediums. For example:

```
var theAdmin = formula.Root.findElement( 'root=Administration' )
var theSessions = formula.Root.findElement( 'sessions=Sessions/
formulaServer=Server/root=Administration' )
```

5.2 Understanding Element Standard Properties

Table 5-1 lists the standard properties available for elements.

Table 5-1 *Standard Properties and Descriptions*

Property	Description
dname	The distinguished name of the element.
name	The name of the element.

Property	Description
id	A system-wide unique identifier for the element, in integer format. This is not a persistent identifier, so it is expected to change if the Operations Center system is restarted.
condition	The condition of the element.
lastUpdate	The date/time of the last change to the element.
alarms	The alarms (if any) for this element.

5.3 Traversing the Element Hierarchy

Each Operations Center element contains location information concerning its own position in the element hierarchy, as well as relationships with other elements in the hierarchy.

5.3.1 Children Property

The children property is an array of elements contained within the target element. If it is empty, its length is zero. For example:

```
// Print the name of each of the elements in the 'Elements' root.
for( var i = 0 ; i < formula.Elements.children.length; ++i )
   writeln( formula.Elements.children[i].name )
```

5.3.2 Relationships Property

The relationships property is an array of elements of which the target element is a member, such as business views. If it is empty, its length is zero. For example:

```
// Lookup the relationships of this element (assuming it is in scope)
for( var i = 0 ; i < element.relationships.length; ++i )
   writeln(element.relationships[i].name )
```

5.3.3 Parent Property

The parent of the element exists within the naming relationship (NAM). The element has only one parent, and always has a parent, unless it is the topmost root element (Enterprise), or has been destroyed. For example:

```
writeln( 'The parent of ' + element + ' is ' + element.parent )
```

5.3.4 Walk Function

The walk function can visit all the children within a given element, including the entire tree. To use this function, pass another function as an argument. This passed function is “visited” along the entire tree. For example:

```
function visitor( child )
{
   writeln( 'Visited element: ' + child )
}

formula.Root.walk( visitor )
An alternate way to visit each node is to pass a NOC Script object that contains a
visit function.
Example:
var visitor =
{
   count: 0,
   visit: function ( child )
   {
      visitor.count++
   }
}

formula.Root.walk ( visitor )
writeln( 'The visitor saw ' + visitor.count + ' elements' )
```

5.3.5 Hierarchy Utilities

Lookup by FindElement()

The findElement function can find a hierarchy-relative element within a given element. To find an element within the entire Operations Center element tree, use Root.findElement. For example:

```
var sessionsElement = formula.Root.findElement( 'sessions=Sessions/
formulaServer=Server/root=Administration' )
writeln( sessionsElement )
```

Lookup by Relative Sub-Property

Access elements contained within the tree of another element as pseudo-properties of the element. For example:

```
var sessionsElement = formula.Administration.findElement( 'sessions=Sessions/
formulaServer=Server' )
writeln( 'There are: ' + sessionsElement.children.length + ' sessions logged into
Formula' )
```

This example finds the Sessions element, which is a child of the Operations Center Server element, and prints the count of the number of children in this element. Each session logged into Operations Center is represented by an element inside the Sessions element, so this accurately displays the number of sessions logged into Operations Center.

5.4 Browsing for Elements Function (formula.gui)

The following methods are used to browse and select elements from the Elements hierarchy. These are entry points to the BrowseForElementDialog class that enable bringing up the element dname browser.

- formula.gui.browseForDNames(Component parent, String multiSelect)
- formula.gui.browseForDNames(Component parent, String rootDName, String multiSelect)
- formula.gui.browseForDNames(Component parent, String rootDName, String title, String label, String multiSelect)

Each version returns a java.util.Vector of java.lang.String for each DName selected in the browser.

Create a client operation that uses a script containing this function, such as the example shown below. Fire the operation. If you select one or more elements in the browser, the operation shows a pop-up listing the selected elements' dnames.

The operation allows supplying a title and label prompt for the element browser. Otherwise, click *OK* in this initial pop-up dialog box to use the browser defaults.

```
var promptDialog = formula.gui.PromptDialog(appFrame, "Test Prompt Dialog Driver",
[ "Title", "Label" ], [ false, false ])

var title = null
var label = null

if ( promptDialog.process() == true )
{
  title = promptDialog.getInputText(0);

  label = promptDialog.getInputText(1);
}
var dnamesVector

if ( title != null && label != null && title.length() > 0 && label.length() > 0 )
    dnamesVector = formula.gui.browseForDNames(appFrame, "root=Elements", title,
label, true)
else
    dnamesVector = formula.gui.browseForDNames(appFrame, "root=Elements", true)
if ( dnamesVector.size() > 0 )
{
   var dnames = "";
   for (var i = 0; i < dnamesVector.size(); i++)
  dnames = dnames + dnamesVector.elementAt(i) + "\n"
   info(dnames)
}
```

5.5 Element Properties and the Properties Object

A Operations Center element has a number of functions associated with it, in addition to the properties that are exposed from either the underlying management system or Operations Center itself. To access a list of properties published by the underlying data source, use the properties property of the element. For example:

```
js> var serverElement = formula.Administration.findElement( 'formulaServer=Server'
)
writeln( 'Server properties:' )
for( var p in serverElement.properties )
   writeln( '   ' + p + ': ' + serverElement[p] )
Server properties:
   Object Heap Used Memory (KB): 6684
   Adapters: 42
   Auditable:
   Formula Server Start Timestamp: 11/13/2002 at 10:37 PM EST
   Condition: OK
   Algorithm:
   Total Alarms: 0
   Formula Server Has Been Up For: 0 Days 0 Hours 12 Minutes 8 Seconds
   Element: formulaServer=Server/root=Administration
   Object Heap Free Memory (KB): 4068
   Algorithm Parameters:
   Total Element Classes: 121
   Object Heap Allocated Memory (KB): 10860
   Last Reported: Wed Nov 13 22:37:45 GMT-0500 (EST) 2002
   Sessions: 1
   Total Elements: 353
js>
```

5.6 Getting and Setting Properties

To get any property set by the underlying management system, use either the dot notation element.property or the array notation element[property]. The array notation might be necessary if the property name itself is a reserved word of JavaScript, such as class, in, or function. For example:

```
js> var serverElement = formula.Administration.findElement( 'formulaServer=Server'
)
js> writeln( 'Memory in use by Formula: ' + serverElement[ 'Object Heap Used Memory
(KB)' ] + 'KB' )
Memory in use by Formula: 7464KB
js>
```

IMPORTANT: Assigning a property to an element exposed in the scripting engine might result in an actual transaction against a managed resource.

To set a property, assign a value to the right-hand portion of an assignment operation. The following example shows the two ways to denote property accessors: one using dot (.) notation and one using array notation ([]).

For example:

```
// Warning, this sets an arbitrary organization to have new contact information;
// DO NOT use on a production system!
if( formula.Organizations.children.length > 0 )
{
   var someOrg = formula.Organizations.children[0]
   writeln( 'Updating: ' + someOrg )
   someOrg[ 'Contact' ] = 'Acme Support'
   someOrg.Company = 'Acme Corporation'
   someOrg.Email = 'support@acme.com'
}
else
   writeln( 'You have no organizations to set properties' )
```

5.7 Accessing Custom Properties

It is possible to search for custom properties assigned to metamodel classes. Use the `getattr()` command.

The `attr` name requires the prefix metamodel.class. For example, assume there is a Service Model element named metamodel.class.SVG_Drawing. Use `getattr()` to search the Service Model element for an attribute named metamodel.class.SVG_Drawing or SVG_Drawing.

```
element.getAttr( 'metamodel.class.[AttrName]' )
```

If the attribute is found in the Service Model element, it returns the attribute. If the attribute is not found, it searches the metamodel class of the element. If the attribute is found in the class element, it returns the attribute.

5.8 Performing Operations with Menu Options

A script can perform operations defined as a menu option for a Operations Center element by using the operation command name.

5.8.1 Using the Perform Method

The perform method takes a session as its first argument, followed by the command name of the operation to perform. An optional parameter for the function is an array of any length, if the operation supports it.

```
perform( session, operationName, /* array */ alarms, /* array */ params )
Example:
js> // Find the "users" group.
js> var users = formula.Administration.findElement( 'group=users/groups=Groups/
security=Security' )
js> // Tell the group to force off all sessions.
js> var result = users.perform( session, 'Connect|ForceOff', [], [] )
js> writeln( result )
Warning. You are not able to logout your own session.

There were no sessions active with the given account users
js>
```

5.8.2 Using Customized Operations

If an operation is defined for an element (it is displayed in the user interface), you can invoke it programmatically via a script. For example:

```
// Assume there is an operation named 'Cut Ticket' on the alarms
// of the first (and only) adapter element, and that there are valid
// alarms on this adapter.
var someAdapterElement = formula.Elements.children[0]
var alarms = someAdapterElement.alarms
someAdapterElement.perform( session, 'Cut Ticket', [ alarms[0] ], [] )
```

This script performs three actions:

1. Locates a top-level adapter element; it assumes there is at least one.
2. Obtains the alarms for this element; it assumes there is at least one.
3. Performs the Cut Ticket operation on this alarm, through the element. It passes only one alarm.

5.8.3 Loading Another Script

If the only intent is to load another script with the operation, use the @ function. For example:

```
@login.fs
```

However, to load the script from within a script, it is necessary to use the load function. For example:

```
formula.log.info( "loading" );
load( 'test.fs' );
formula.log.info( "done" );
```

5.8.4 Using Hidden Operations

To specify an operation for an element that is not visible in the user interface, but is usable from scripts, set the “hidden” property of the operation.

5.9 Printing Operations for Elements

The following script prints the operation name and command for a selected element. If the command is blank, then the command is the same as the name. It works as a client script only.

```
//@debug on
  formula.log.info("Printing operations for " + element.dname);
  var ops = element.operations;
  for(i=0; i < ops.length; i++) {
    formula.log.info(ops[i].name + "|" + ops[i].command);
  }
```

5.10 Client-Side Methods

Client-side methods can display various windows for an element. These methods are available in the Operations Center console only. For example:

```
var element = formula.Root
element.showAlarms()
```

Table 5-2 describes client-side methods.

Table 5-2 Client-Side Methods for Displaying Element Windows

Method	Description
showAlarms()	Display the Alarms window for this element.
hideAlarms()	Hide the Alarms window for this element.
showConsole()	Display the Console window for this element.
hideConsole()	Hide the Console window for this element.
showProperties()	Display the property pages for this element.
hideProperties()	Hide the property pages for this element.
showPerformance()	Display the Performance window for this element.
hidePerformance()	Hide the Performance window for this element.

5.11 Server-Side Methods

A script that executes on the Operations Center server can interact with the session that invoked it, if it is an operation script. Some interesting functions are available to allow for interaction with the session:

```
invokeScript( name, script, values[], names[] )
```

This causes invocation of a script on the client machine to which the session refers.

Table 5-3 lists server-side methods.

Table 5-3 Server-Side Methods

Method	Description
name	The name of the script.
script	The actual script content to execute.
values	An array of object values to pass to the script.
names	The names of the values to pass to the script. Ensure these are valid JavaScript identifier names and that there is exactly one entry for each entry in the values array.

5.11.1 sendMessage (message)

This function sends a message to the session.

5.11.2 monitorProcess (name, command)

This function causes the session to start monitoring the output of a process which executes on the server.

6 Alarm Functions

Various mechanisms are available to create, interact with, query, and change Operations Center alarms, which are attached to Operations Center elements. An alarm has a number of functions associated with it, in addition to the properties that are exposed from either the underlying management system or Operations Center itself.

6.1 Creating Alarms

Create an alarm for some of the supported adapters by using the `createAlarm()` method. Alarms are associated with a specific integration (adapter) and the properties vary according to the adapter. After getting a reference to the adapter from any element generated by the adapter under the Elements hierarchy, use this method to create an alarm against the adapter. The format is:

```
adapter.createAlarm( "Formula", Flds, Vals, null );
```

Where,

- **Formula:** The class of the alarm.
- **Flds:** An array of alarm fields.
- **Vals:** An array that matches Flds and contains the alarm values.
- **null:** The last parameter is always null.

For example:

```
var mgrElement =
formula.Root.findElement("script=Adapter%3A+NOC+-+Universal+Adapter/
root=Elements");

var adapter = mgrElement.adapter;

formula.log.info("Adapter: " + adapter);
var fields = ["Summary","Class","Severity"];
var Vals = ["AlarmStorm detected for Class EMOS", "EMOS","Critical"];
adapter.createAlarm( "EMOS", fields, Vals, null );
```

6.2 Getting and Setting Alarm Properties

To access a list of properties or fields published by the underlying data source to the alarm, use the properties property of the alarm. For example:

```
// Print out all the alarm's of the elements tree
var al = element.alarms;

formula.log.info("Alarm count: " + al.length);
for (var i = 0; i < al.length; ++i ) {
    formula.log.info( 'Alarm properties of: ' + al[i] )
    for (var p in al[i].properties) {
      formula.log.info( '   ' + p + ': ' + al[i][p] )
    }
}
```

To get any property set by the underlying management system, use either the dot notation alarm.property or the array notation alarm[property]. The array notation might be necessary if the property name itself is a reserved word of JavaScript, such as “class”, “in”, or “function.” For example:

```
var alarm = formula.Elements.alarms[0] // Assume at least one!
writeln( 'Alarm: ' + alarm + ' is attached to ' + alarm.element )
```

To set a property, assign a value to the right-hand portion of an assignment operation. The two ways to denote property accessors are using dot (.) notation and using array notation ([]). Assigning a property to an alarm exposed in Operations Center scripting engine might result in an actual transaction against a managed resource.

An example:

```
// Warning, this adds some random field values to a random alarm
// DO NOT use on a production system!
try
{
   var alarm = formula.Elements.alarms[0] // Assume at least one!
   alarm.foo = 'bar'
   alarm[ 'class' ] = 'Zinger'
}
catch( Exception )
{
   formula.log.warn( 'Could not modify alarm: ' + Exception )
}
```

6.3 Getting the Element Associated with the Alarm

The alarms[x][element] can be used to access the element associated with an alarm.

6.4 Modifying Alarm Content

Alarms cannot always be modified within Operations Center. Some management systems do not support modifying alarm content. Therefore, when setting properties for an alarm, always use a try/catch block, in case the alarm modification is not supported.

6.5 Developing Automation Scripts for Alarms

The Automation feature in Operations Center allows users to define actions that are triggered by activities or condition changes that occur in the network. The feature is explained in *Defining and Managing Automation Events* in the *Operations Center 5.6 Server Configuration Guide*. One of the automation actions is running a script.

For example, the following script instructs Operations Center to audit the actions performed against alarms.

```
////////////////////////////////////////////////////////  AuditActions.fs
writeln( "Auditing Actions Against Alarms..." );

//  Filename that the Actions should be logged too
var fpAudActionFile = "../logs/ActionAudit.log";

//  Variable to hold the auditing output
var str = "" + '\n';

//  Build the current date and time
var RightNow = new Date();
str += RightNow.getDate() + "-";
str += RightNow.getDay() + "-";
str += RightNow.getFullYear() + " ";
str += RightNow.getHours() + ":";
str += RightNow.getMinutes() + ":";
str += RightNow.getSeconds();

//  Tag the Action to a person
str += ' User "' + user + '" ran the command "' + command + '" against alarm:\n';

// Grab the top level alarm information
for( var i = 0; i < alarms.length; ++i )
{
  str += 'Alarm: ' + alarms[i].getID() + ', element: ' +
alarms[i].getElement().getDName();

  // Grab the actual alarm... if you issued a "Delete" it might be gone already
  for( p in alarms[i].properties )
  {
    str += '\n\t' + p + " = " + alarms[i][p];
  }
}

//   Open the logging file
var writer = new java.io.PrintWriter( new java.io.FileOutputStream(
fpAudActionFile, true ) );

//   Write the string we built to the file
writer.println( str );

//   Close the file
writer.close();

// EOF() AuditActions.fs
```

To implement this script:

1. Copy the script file (in the previous example, it is `AuditActions.fs`) to `/`*`OperationsCenter_install_path`*`/database/scripts/util`.
2. Create an action under the Automation Server element that runs a script from the library, and point to `util/AuditActions.fs`.
3. Create an automation server item that is hooked at a level (Enterprise, Elements, Adapter, and so on) that is linked to An Alarm operation was performed.

4. Select the action defined in Step 2 and apply it.
5. Close, clear, and acknowledge an alarm.
6. View the */OperationsCenter_install_path*/logs directory to locate the ActionAudit.log file.

The variables available during script execution depend on the event that triggered that script. Table 6-1 lists the global variables associated with alarm events.

Table 6-1 *Global Variables for Alarm Events*

Event	Variable
Alarm created	alarm
Alarm deleted	list (integer list of IDs)
Alarm updated	alarm
Alarm operation performed	Alarms

7 User Functions

Various mechanisms are available to interact with, and query Operations Center user accounts. User information is exposed to the scripting engine in a way that allows access to properties.

A script can create and delete users, as well as assign them to groups. Refer to the examples in this section.

7.1 Introduction

A Operations Center user has a number of functions associated with it, in addition to the properties that Operations Center exposes. For example:

```
// get user's full name
var userName = user.fullName
To access a list of properties or fields for the user, use the properties property
of the user. However, here is a short list of what is available for the User
object:
user.name
user.fullName
user.email
user.phone
user.fax
```

7.2 Getting and Setting Properties

To obtain any property for the user, use either the dot notation user.property or the array notation user[property]. For example:

```
try
{
    formula.log.info ('Attempting to update user info')
        var uid = '321665949'
        // Find the user.
        var userElement = formula.Root.findElement( 'user=' +
            formula.util.encodeURL( uid ) +
            '/users=Users/security=Security/root=Administration' )
    var aUser = userElement.user;
        aUser.fullName( 'Joe Thomas')
        aUser.email('joe@acme.com')
        aUser.phone('888-555-1212')
        aUser.pager( '800-555-1212')
}
catch( Exception )
{
   formula.log.warn( 'Could not modify user: ' + Exception )
}
```

7.3 Setting Properties for LDAP Users

If users are imported using LDAP, the following properties can be set:

- LDAP.authentication
- LDAP.dname
- LDAP.protocol
- LDAP.url
- LDAP.urlalternate

To access these properties in a script, or to create these properties for users who were manually created, use the following functions:

- element.user.setProperty("LDAP.url", "ldap://host");
- element.user.setProperty("LDAP.url.alternate", "ldap://host");
- element.user.setProperty("LDAP.authentication", "value1");
- element.user.setProperty("LDAP.protocol", "value2");
- element.user.setProperty("LDAP.dname", "full query for the user usually the cn");

7.4 Managing Users

Use a script to perform the following user access functions:

- Create, find, or delete users
- Create or find groups

The following sections describe user and group script functions and changing group membership:

- Section 7.4.1, “Understanding Script Functions,” on page 48
- Section 7.4.2, “Changing a User’s Group Membership,” on page 50

7.4.1 Understanding Script Functions

The following sample script shows how to perform these functions. Read the comments to understand the purpose of each function.

```
// @debug off

// Locate the groups and users elements.
var groups = formula.Root.findElement( 'groups=Groups/security=Security
root=Administration' )
var users = formula.Root.findElement( 'users=Users/security=Security/
root=Administration' )

// Do we have the testGroup?  If not, create the new group.
try
{
   formula.log.info( 'Finding group testGroup' )
   formula.Root.findElement( 'group=testGroup/groups=Groups/security=Security/
root=Administration' )
}
catch( missing )
{
   formula.log.info( 'Group testGroup missing; creating...' )
   groups.perform( session, 'LifeCycle|Create', [], [
```

```
                    'testGroup',                  // Group name
                    'Test group description',     // Group description
                   '' ] )                       // Group membership (comma-delimited
list)
}

// Delete some users.
var userNames = ['jim', 'anne', 'lisa', 'neil', 'john' ]
for( var i = 0; i < userNames.length; ++i )
{
   try
   {
      // Find the user.
      formula.log.info( 'Finding user ' + userNames[i] )
      var user = formula.Root.findElement( 'user=' + formula.util.encodeURL(
userNames[i] ) + '/users=Users/security=Security/root=Administration' )
      formula.log.info( 'Deleting user' )
      user.perform( session, 'LifeCycle|Delete', [], [] )
   }
   catch( missing )
   {
   }
}

// Create some users.
for( var i = 0; i < userNames.length; ++i )
{
   try
   {
      // Find the user.
      formula.log.info( 'Finding user ' + userNames[i] )
      formula.Root.findElement( 'user=' + formula.util.encodeURL( userNames[i] ) +
'/users=Users/security=Security/root=Administration' )
      formula.log.info( 'Found existing user ' + userNames[i] )
   }
   catch( missing )
   {
      // Create the user.
      var memberOf = 'testGroup' + ( ( ( i % 2 ) == 1 ) ? ',users' : '' ) // Test
for group membership
     formula.log.info( 'User ' + userNames[i] + ' missing; creating with groups: '
+ memberOf + '...' )
      users.perform( session, 'LifeCycle|Create', [], [
                     userNames[i], // User name
                     'password',   // Password
                     '',           // Full name
                     '',           // email
                     '',           // phone
                     '',           // fax
                     '',           // pager
                     memberOf      // Group membership (comma-delimited list)
                     ] )
   }
}

// Now, set the user's profile to one we know (guest).
var guest = formula.Root.findElement( 'user=guest/users=Users/security=Security/
root=Administration' )
if( guest )
```

```
{
   var guestProfile = guest.profile
   formula.log.info( 'Setting profile to guest profile: ' + guestProfile )

   for( var i = 0; i < userNames.length; ++i )
   {
      formula.log.info( 'Updating profile for user ' + userNames[i] )
      var user = formula.Root.findElement( 'user=' + formula.util.encodeURL(
userNames[i] ) + '/users=Users/security=Security/root=Administration' )
      user.profile = guestProfile
   }
}

formula.log.info( 'Done' )
```

7.4.2 Changing a User's Group Membership

To change a user's group membership, use the following function and replace 'group1, group2' with actual group names:

```
user.perform( session, 'LifeCycle|SetGroupNames', [], ['group1,group2'] )
```

For example:

```
  var tech = 'jim'
     var user = formula.Root.findElement( 'user=' + formula.util.encodeURL( tech )
+ '/users=Users/security=Security/root=Administration' )
     user.perform( session, 'LifeCycle|SetGroupNames', [], ['group1,group2'] )
```

8 Using the State Variable to Cache and Store Information

In situations when a script runs multiple times on a Operations Center server or console, it is advantageous to cache or store information that subsequent invocations of the script can use. Use the Operations Center state variable to store and share data among scripts.

The state variable is by default an empty object. Its unique feature is having only one value in the entire process. It is shared across invocations of scripts and between scripts running on different threads in the same process. For this reason, it is possible to place variables on the state object that are used elsewhere.

For example, assume that a script must open a file and write some data to it. Later, run the script again, which means opening and closing the file again. Use the state variable to cache the value of the stream that the script opens, and then reuse the stream during another invocation of the script. For example:

```
if( ! state.fileStream )
   state.fileStream = new java.io.PrintWriter( new java.io.OutputStreamWriter( new
java.io.FileOutputStream( '/output', true ) ) )
state.fileStream.println( 'Some more data right now: ' + new java.util.Date() )
state.fileStream.flush()
```

The convention on line 1 checks for the presence of the fileStream value in the state variable. If it is not found, fileStream is opened. Whether it is initially opened or reused from the state variable, the script simply writes data to the stream, then flushes it.

If the script places many variables in the state variable, consider placing them in an object held by the state variable:

```
if( ! state.mystuff )
{
   state.mystuff = new Object()
   state.mystuff.fileStream = new java.io.FileInputStream( '/output' )
   state.mystuff.iterations = 0
}
state.mystuff.iterations++
```

9 Miscellaneous Scripting Functions

The following sections describe various scripting functions that are available:

9.1 Script Functions to Set Root Cause

If a script sets the conditionState object state and result, it should also set the root cause information indicating why the condition was set to a specific value as shown in the following declaration.

```
conditionState.setState() as FINISHED and conditionState.setResult(
resultingCondition, reason);
```

Set the root cause information by using the following method:

```
conditionState.getRootCause().setContributorsReasonWhere( contributors, reason,
where );
```

Table 9-1 defines the parameters used to set the root cause information.

Table 9-1 *Parameters Used to Set Root Cause*

Parameter	Description
ElementImpl[ ] contributors	Specifies the elements that caused the element to have a specific condition. For example, contributors can be all critical children of the element. Set this to null if no other elements contribute.
String reason	Text description of the reason why the element has a specific condition. For example, “I have CRITICAL children elements.”
String where	A text description of where the condition was set. For example, “Script MakeACondition.” Set it to null to automatically fill in by using the algorithm name.

For customized algorithms to support root cause and show impact, set root cause information that indicates why the condition was set to a particular value.

9.2 Scripting Functions in Automation Tasks

Automation events can trigger an alert when a network event occurs that might require intervention. Scripting capabilities are available to define more complex actions. *Defining and Managing Automation Events* in the *Operations Center 5.6 Server Configuration Guide* describes the automation features.

9.2.1 Accessing SLA Information

Table 9-2 lists the objects that are available when defining a script action for an SLA objective breach. It is possible to mine Agreement Name, Reason, Compliance, etc., from these objects:

Table 9-2 *Objects for Accessing SLA Information*

Object	Description
objectiveContext	This context provides information about the defined objective/offer.
computeContext	This real-time context provides information about the real time objective/offer instances.

```
var ObjectiveName = objectiveContext.getObjectiveType().getName()
var SLAName = objectiveContext.getOfferDefinition().getName()
var reason = computeContext.getObjectiveInstance().getReason()
var ObjectiveHealth = computeContext.getObjectiveInstance().getHealth()
```

In the following sample script, messages display in the formula trace file when an objective breach occurs:

```
formula.log.info( "SLA  = " + objectiveContext.getOfferDefinition().getName())
formula.log.info( "Objective  = " + objectiveContext.getObjectiveType().getName())
formula.log.info("Obj Health = " +
computeContext.getObjectiveInstance().getHealth());
formula.log.info("Objective Breach Reason = " +
computeContext.getObjectiveInstance().getReason());
```

9.2.2 Using Event Variables

All script-based automations have an event variable that represents the event that triggers the automation. Table 9-3 describes event variable fields.

Table 9-3 *Event Variable Fields*

Field	Description
sourceEvent	The name of the triggered automation event.
eventCondition	The current event condition.

Field	Description
priorCondition	The event condition before the automation event occurred.
eventType	The type of automation filter that triggered the event: ◆ EventType_ConditionChange ◆ EventType_AlarmAdded ◆ EventType_AlarmRemoved ◆ EventType_AlarmOperationPerformed ◆ EventType_ElementOperationPerformed ◆ EventType_AlarmChange
element	The element to which the triggered automation action is assigned.
createTime	The time the automation event occurred,

9.2.3 Sample Code

The following sample code shows the fields documented in Table 9-3 on page 54:

```
package com.mosol.Formula.Automation;

import java.util.EventObject;
import com.mosol.ORB.Formula.ElementCondition;
import com.mosol.util.StringMap;


public class AutomationEvent extends EventObject
{
   public String what = null ;
   public Automation automation = null;
   public EventObject sourceEvent = null ;
   public ElementCondition eventCondition = null ;
   public ElementCondition priorCondition = null;
   public AutomationElement element = null ;
   public StringMap eventDataObjects = new StringMap() ;
   public int eventType = 0 ;
   public long createTime = System.currentTimeMillis();

   static public final int EventType_ConditionChange = 1 ;
   static public final int EventType_AlarmAdded = 2 ;
   static public final int EventType_AlarmRemoved = 3 ;
   static public final int EventType_AlarmOperationPerformed = 4 ;
   static public final int EventType_ElementOperationPerformed = 5 ;
   static public final int EventType_AlarmChange = 6;

   public AutomationEvent( String what,
                           Automation automation,
                           EventObject sourceEvent,
                           ElementCondition eventCondition,
                           AutomationElement element,
                           int eventType )
   {
     this( what, automation, sourceEvent, eventCondition, element, eventType, null
);
   }

   public AutomationEvent( String what,
                           Automation automation,
                           EventObject sourceEvent,
                           ElementCondition eventCondition,
```

```
                            ElementCondition priorCondition,
                            AutomationElement element,
                            int eventType )
   {
     this( what, automation, sourceEvent, eventCondition, priorCondition, element,
eventType, null );
   }

   public AutomationEvent( String what,
                            Automation automation,
                            EventObject sourceEvent,
                            ElementCondition eventCondition,
                            AutomationElement element,
                            int eventType,
                            String eventDataName,
                            Object eventData )
   {
     this( what, automation, sourceEvent, eventCondition, element, eventType, null
);
      eventDataObjects.put( eventDataName, eventData );
   }

   public AutomationEvent( String what,
                            Automation automation,
                            EventObject sourceEvent,
                            ElementCondition eventCondition,
                            AutomationElement element,
                            int eventType,
                            StringMap eventDataObjects )
   {
     this(what, automation, sourceEvent, eventCondition, null, element, eventType,
eventDataObjects);
   }
   public AutomationEvent( String what,
                            Automation automation,
                            EventObject sourceEvent,
                            ElementCondition eventCondition,
                            ElementCondition priorCondition,
                            AutomationElement element,
                            int eventType,
                            StringMap eventDataObjects )
   {
      super( sourceEvent.getSource() ) ;
      this.what = what ;
      this.automation = automation ;
      this.sourceEvent = sourceEvent ;
      this.eventCondition = eventCondition ;

      this.priorCondition = priorCondition ;
      this.element = element ;
      this.eventType = eventType ;
      if( eventDataObjects != null )
         this.eventDataObjects = eventDataObjects ;
   }

   public Object getEventObject( String key )
   {
      return eventDataObjects.get( key );
   }
}
```

9.3 Remotely Calling One Script from Another

It is possible to remotely make calls from a client script to a server script and vice-versa. This requires sending a prompt script that causes the recipient to execute the script via a remote proxy. The following example script defines a variable named callback and sends a prompt script to the client. The result is executing this script through a remote proxy.

```
////////////////////////////////////////////////////////
// Operation definitions to add to an element: (paste into Operations.ini)
//
// [Enter Trouble Ticket]
// description=Trouble Ticket...
// context=element
// target=dname:root=Elements
// permission=manage
// type=serverscript
// operation=// @debug off \nload( "examples/ticket.fs" );
//

// Set our log category
formula.setLogCategory( "Ticket" )

// Log startup
formula.log.info( "Starting trouble ticket script" )

// Create our object which will be the "callback" from the client
// Note: this is standard javascript syntax for creating an object with named
properties/values
var callback =
{
    setTicketInfo: function( reason )
    {
       // We'll log what the user did, to simulate connecting to the ticketing
system.
       formula.log.info( "User created trouble ticket:" )
       formula.log.info( "   Element: " + element )
       formula.log.info( "   Reason:  " + reason )

       // Let's notify the user that we did what was asked.
       session.sendMessage( 'Trouble ticket created for ' + element + '.
Content:\n\n' + reason )
    },

    cancel: function()
    {
       formula.log.info( "User cancelled trouble ticket creation" )
       session.sendMessage( 'Trouble ticket cancelled for ' + element + '.' )
    }
}

// We're going to send this callback and a prompt script to the client, which will
// then cause execution to this script through remote proxy.

var clientTicketScript = "\
    // @opt -1 \
    // @debug off \
    var result = prompt( 'Enter trouble ticket information for ' + elementName +
':', \
                         'Trouble Ticket' ) \
    if( ! result ) \
        callback.cancel() \
    else \
        callback.setTicketInfo( result )\
"

//
```

```
// Now, send the dialog script to the user who invoked this script
//
// Note: this script is sent to variables, called "callback" and "elementName"
//       The callback is wrapped via a remote proxy using the
formula.util.makeRemote()
//       function.  The element name is simply the element name of the element
//       the user initiated this ticket for.
//
formula.log.info( "Sending dialog script to client" )
session.invokeScript( 'Enter Trouble Ticket',
                      clientTicketScript,
                      [ formula.util.makeRemote( callback ), element.name ],
                      [ 'callback', 'elementName' ] )
formula.log.info( "Done!" )
```

9.4 Accessing Metamodel Properties

To look up metamodel-related properties for elements, use a script that queries the supported attribute IDs. Table 9-4 lists the metamodel attributes.

Table 9-4 *Metamodel Attributes*

Attribute	Returns
metamodelPageNames	Metamodel page names for an element.
metamodelInvalidatedAttrs	State of the metamodel properties validation.
metamodelInvalidatedAttrs.Page	State of the metamodel properties validation given page name.
metamodelInvalidatedAttrIDs	Invalidated attr IDs of the metamodel properties.
metamodelInvalidatedAttrIDs.Page	Invalidated attr IDs of the metamodel properties given page name.
metamodelPageAttrIDs	Attr IDs of the metamodel properties.
metamodelPageAttrIDs.Page	Attr IDs of the metamodel properties given page name.

9.5 SCM and Scripting

The Service Configuration Manager (SCM) imports data from external sources and generates new element hierarchies in the Operations Center console. A large part of the import preparation involves mapping imported data. This section assumes familiarity with SCM functionality. Read “Using the Service Configuration Manager”in the *Operations Center 5.6 Service Modeling Guide* for more information.

When using SCM, most sites perform these two actions:

- First Stage: Relate services to applications.
- Second Stage: Relate applications to hosts.

When beginning to relate applications to hosts, the following mapping usually occurs:

```
Host A Host A (OV) Host A (Patrol) Host A (Remedy)
```

However, this format is not optimal for most users. They want to see the children of `Host A (OV), Host A (Patrol), Host A (Remedy)`. Use dynamic matching to do this:

```
[^\x2F]*=${name}.*
```

Join rules can be written using regular expression and Apache Velocity (http://velocity.apache.org).

When trying to join things with spaces, use this main concept:

```
$class=$name
```

Use the form:

```
${formula.util.encodeURL($class)}=${formula.util.encodeURL($name)}
```

Main packages for BCSM are in com.mosol.Formula.ViewBuilder Also see com.mosol.Formula.commands.ViewBuilder.java.

Some scripting applications involving SCM include scheduling SCM jobs, building core views and generating elements.

- Section 9.5.1, “Scheduling a SCM Job by Using a Script,” on page 59
- Section 9.5.2, “Scheduling Multiple SCM Jobs by Using a Script,” on page 60
- Section 9.5.3, “Generating Elements by Using a Script,” on page 61
- Section 9.5.4, “Building Core Views,” on page 62

9.5.1 Scheduling a SCM Job by Using a Script

The following script example shows running a SCM job.

NOTE: Do not run SCM jobs in parallel unless you know for certain that the SCM jobs do not affect the same elements. To be safe, use a script that runs the SCM jobs in a specific sequence, as shown in Section 9.5.2, “Scheduling Multiple SCM Jobs by Using a Script,” on page 60.

```
var logger = Packages.org.apache.log4j.Logger.getLogger("BSCM_copy_script");

function buildbscm(viewDName)
   {
     var viewElement = null;
     try {
        viewElement = formula.Root.findElement( viewDName );
     } catch( Exception ) {
     formula.log.error( "ERROR - caught exception while finding view " +
     viewDName + ": " + Exception );
     return false;
     }
     if( viewElement == null ) {
       formula.log.error("ERROR - view " + viewDName + " not found" );
       return false;
     }
     try {
       var ret = viewElement.perform( session, "ViewBuilder|Run", [], [] );
     } catch( Exception ) {
     return false;
     }
   }

logger.info('Build bscm: start ----------------------');
buildbscm('org=elements_bscm/root=Organizations');
logger.info('Build bscm: end ----------------------');
```

9.5.2 Scheduling Multiple SCM Jobs by Using a Script

The following script example shows the kicking off of SCM jobs in a specific sequence in order to remove the risk of them conflicting with each other. This is important unless you know for certain that the SCM definitions do not affect any of the same elements or hierarchies.

```
// This script uses a "dnames" array to define the elements containing
// the SCM job definitions. Instead of using the schedule in the SCM job,
// it uses the "Administration > Jobs" Time Management to run the SCM jobs
// in the order you specify.

// Edit the dnames variables to specify the specific dnames in the order you
// require them to be built.

// For more advanced functionality (ie; requires more scripting), you can turn on
// adapters just before a job runs, interrogate the adapter to see if it is done
// "starting". Once it is done, you can kick off the SCM job and when complete,
// you can stop the adapter.


formula.log.info( "Starting build of SCM jobs..." );

var dnames = new Array()

dnames[0] = "Application=Application+A+thru+C/layout_organic=my+Applications/
root=Generational+Models/root=Services"
dnames[1] = "SCM job name using DName"

// Add dname instances above as necessary.

/* no changes required below here */
/* function to kick of SCM jobs based on the dnames above */

function buildView( viewDName )
{
    var viewElement = null;

// find the element based on the dname

    try {
        viewElement = formula.Root.findElement( viewDName );
    } catch( Exception ) {

        formula.log.error( "ERROR - caught exception while finding view " +
viewDname + ": " + Exception );

        return false;
    }

// if element can't be found, issue an error to log file

    if( viewElement == null ) {

        formula.log.error("ERROR - view " + viewDName + " not found" );
        return false;
    }


//if the element is found, kick of SCM job via the perform() method.

    try {

        var ret = viewElement.perform( session, "ViewBuilder|Run", [], [] );

    } catch( Exception ) {
```

```
        return false;
    }
}

/* for loop to run through each dname and kick off SCM job */

for( var i=0; i<dnames.length; i++ )

{
    formula.log.info( "Starting SCM job for:  " + dnames[i] );
    buildView( dnames[i] );
}

formula.log.info( "Finished running all SCM jobs" );
```

9.5.3 Generating Elements by Using a Script

The scripts used to generate elements vary, but it is important to add the following line to the end of the script, or else all elements generated by the script are removed at the end of the SCM job.

```
bscm.addGeneratedElement( postScriptGeneratedElement )
```

For example:

```
// Example 'Post Element Generation Script' to build element trees under generated
elements that look like:
//    gen_container=Versions/router=NetRouter+One/gen_container=Installed+Routers/
[generated element]

// Create an element tree.
var isSourceChild = true; // Make children contribute condition.
var routerName = 'NetRouter One'; // Will need to URL encode this for spaces or
other special characters.
var child1 = createChildElement( element /* the generated element */,
'gen_container=Installed+Routers', isSourceChild );
var child2 = createChildElement( child1, 'router=' + formula.util.encodeURL(
routerName ), isSourceChild );
var child3 = createChildElement( child2, 'gen_container=Versions', isSourceChild );

// Function to create and return a child on a parent.
function createChildElement( parentElement, childKey , isSourceChild ) {
   // Get the child (or create it if it does not exist).
   var child = server.getElement( childKey + '/' + parentElement.dname );
   if ( isSourceChild ) {
      // This child should contribute condition to the parent.
      bscm.addStaticMatches( parentElement, [ child ], false /*
displaySourceElementsAsChildren */ );
   }
   // Prevent SCM from removing the element.
   bscm.addGeneratedElement( child );
   // Return the element.
   return child;
}
```

9.5.4 Building Core Views

```
//
// This script initiates the build of core views in the following order:
// Production Views --> _Common Views --> GRNs
// Production Views --> _Common Views --> Sybase Hosts
// Production Views --> _Common Views --> Consolidated Host List
//

load("custom/msAdapterUtils.fs");

var errHdr = "MONITOR_FAILED_TO_BUILD_VIEW: ";

function buildCoreViews()
{
    var adapterList = new Array();
    var runtimeAlarmDName;
    var viewDName;
    var viewObject;
    var status;

    formula.log.info("\n\n***************** Building Production Views --> _Common
Views --> GRNs: STARTED");
    runtimeAlarmDName = "EBI-Element=Adapter+Runtime+Information/
DEF_ESP_APPS2=AD_ESP_APPS2/root=Elements";
    adapterList[0] = new MSAdapterObject("AD_ESP_APPS2", runtimeAlarmDName,
RUN_COMPLETED, ADAPTER_STARTED, false);
    adapterList[1] = new MSAdapterObject(NETCOOL_ADAPTER_NAME, "", "",
NETCOOL_ADAPTER_STARTED, false);

    viewDName = "org=GRNs/org=_Common+Views/org=Production+Views/
root=Organizations";
    viewObject = new MSViewObject(viewDName, adapterList);

    status = viewObject.buildView();
    if( status == false )
    {
        formula.log.error( errHdr + "core view " + viewDName + " build failed,
aborting the entire view build process")
        return false;
    }
    formula.log.info("\n***************** Building Production Views --> _Common
Views --> GRNs: COMPLETED\n");
    formula.log.info("Sleeping for 10 seconds before starting next view build...");
    java.lang.Thread.sleep( 10000 );


    formula.log.info("\n\n***************** Building Production Views --> _Common
Views --> Sybase Hosts: STARTED");

    runtimeAlarmDName = "EBI-Element=Adapter+Runtime+Information/
DEF_ESP_SYBASE_INFRA=AD_ESP_SYBASE_INFRA/root=Elements";

    adapterList.length=0;
    adapterList[0] = new MSAdapterObject("AD_ESP_SYBASE_INFRA", runtimeAlarmDName,
RUN_COMPLETED, ADAPTER_STARTED, true);
    adapterList[1] = new MSAdapterObject(NETCOOL_ADAPTER_NAME, "", "",
NETCOOL_ADAPTER_STARTED, false);

    viewDName = "org=Sybase+Hosts/org=_Common+Views/org=Production+Views/
root=Organizations";
    viewObject = new MSViewObject(viewDName, adapterList);

    status = viewObject.buildView();
    if( status == false )
    {
        formula.log.error( errHdr + "core view " + viewDName + " build failed,
aborting the entire view build process")
        return false;
```

```
    }
    formula.log.info("\n***************** Building Production Views --> _Common
Views --> Sybase Hosts: COMPLETED\n");
    formula.log.info("Sleeping for 10 seconds before starting next view build...");
    java.lang.Thread.sleep( 10000 );


    formula.log.info("\n\n***************** Building Production Views --> _Common
Views --> Consolidated Host List: STARTED");

    runtimeAlarmDName = "EBI-Element=Adapter+Runtime+Information/
DEF_ESP_CHL3=AD_ESP_CHL3/root=Elements";
    adapterList.length=0;
    adapterList[0] = new MSAdapterObject("AD_ESP_CHL3", runtimeAlarmDName,
RUN_COMPLETED, ADAPTER_STARTED, true);
    adapterList[1] = new MSAdapterObject(NETCOOL_ADAPTER_NAME, "", "",
NETCOOL_ADAPTER_STARTED, false);

    viewDName = "org=Consolidated+Host+List/org=_Common+Views/
org=Production+Views/root=Organizations";
    viewObject = new MSViewObject(viewDName, adapterList);

    status = viewObject.buildView();
    if( status == false )
    {
        formula.log.error( errHdr + "core view " + viewDName + " build failed,
aborting the entire view build process")
        return false;
    }
    formula.log.info("\n***************** Building Production Views --> _Common
Views --> Consolidated Host List: COMPLETED\n");
    formula.log.info("Sleeping for 10 seconds before starting next view build...");
    java.lang.Thread.sleep( 10000 );

    return true;
}


var server = formula.Administration.findElement('formulaServer=Server' );
// pause updates to Elements.ini
server.perform( session, 'Element|PausePersistence', [],[true] );

var result = buildCoreViews();

formula.log.info("Triggering a FULL GC if possible...");
java.lang.System.gc();

// resume updates to Elements.ini
server.perform( session, 'Element|PausePersistence', [],[false] );

if( result )
{
    formula.log.info( "Core view builds are done, wait for 3 minutes before
starting state view builds." );
    java.lang.Thread.sleep( 180000 );
    var stateViewDName = "Job=Build+State+Views/jobs=Jobs/
timeManagement=Time+Management/root=Administration";
    var jobElement = formula.Root.findElement( stateViewDName );
    if( jobElement.isJobRunning() )
    {
        formula.log.info( "State View refresh job is already running.")
    }
    else
    {
        try {
            formula.log.info( "Enabling job..." );
```

```
            jobElement.perform( session, 'Enable', [], [] );
        } catch( exception ) {
            formula.log.info("Caught exception when enabling job: " + exception);
        }

        try {
            formula.log.info( "Starting job..." );
            jobElement.perform( session, 'Run', [], [] );
        } catch( exception ) {
            formula.log.error( errHdr + "Caught exception when starting job: " +
exception );
        }
    }
}
```

9.6 Miscellaneous Scripting Functions

Table 9-5 provides an overview of various scripting functions:

Table 9-5 *Scripting Functions*

Function	Description
info(message)	Displays a message dialog box with an informational icon. This is part of the standard browser JavaScript extensions, so it is familiar to those who have already used JavaScript.
alert(message)	Displays a message dialog box with a warning/error icon. This is part of the standard browser JavaScript extensions, so it is familiar to those who have already used JavaScript.
confirm(message)	Displays a message dialog box with a question icon, with *Yes/No* button choices. This is part of the standard browser JavaScript extensions, so it is familiar to those who have already used JavaScript. This function returns True if the user clicks the *Yes* button.

prompt(prompt|prompts, title, multiline|multilines)	Similar to confirm(), this prompts the user for information. The prompt argument is a label for a text field where the user enters information. Optionally, the first argument can be an array of strings, which sets up multiple input values.
	
	The title is optional, and denotes the title of the dialog box to display.
	
	The final variable is also optional, and denotes whether or not the prompt text fields are multiple lines or single line (True/False).
	
	The return value is either the string the user entered, or null/undefined. If there are multiple prompts used, it is an array of strings.
	
	Examples:
	
	`var result = prompt( 'Enter some information', 'Formula')`
	`print( 'You entered: ' + result )`
	
	`var results = prompt( [ 'Enter 1', 'Enter 2' ], 'Formula', [ true, false ] )`

fload(scriptName)	Loads a script from the script repository. The script repository is in the formula database/scripts directory. For example:
	
	`load( 'util/login' ) // Loads the login utility.`
	
	`login()`

Function	Description
@	Loads a script from the script repository. The script repository is in the formula database/scripts directory. Use it when loading the script is the only command being issued and it is not embedded within a script. For example, when creating a custom menu operation or loading a script when an adapter starts (Script.onStarted in adapter properties). For example: `@/login.fs // Loads the login utility.`
writeln/write	Write output to standard output. Similar to the standard JavaScript print function.
in/out/err	Standard input, output, and error streams, exposed as variables.
Args	The arguments to the script, as an array of strings. If the script is a command line script, these are the arguments the user entered for the script. If the script is an operation, and the user was prompted for data, this is the data the user entered.

10 Command Line Scripting: The fscript Utility

Use the Operations Center scripting command line environment named fscript to interactively execute or batch-execute script commands. Ensure that the `/OperationsCenter_install_path/bin` directory is in the system path to run these examples.

10.1 Starting fscript

To start the fscript utility, enter `fscript` at the system command line:

```
$ fscript
js>
```

Interactively enter script commands at this prompt. Table 10-1 the additional functions that are available to assist with interactive scripts.

Table 10-1 *Interactive Script Functions*

Script Function	Description
list(obj)	List each normal property of the object.
listAll(obj)	List all properties of the object, even hidden ones.
describe(obj)	Print debugging or declaration of the object.
quit()	Exit the fscript environment.

10.2 Invoking a Script by File Name

To use the fscript utility to batch-execute a script, invoke the `fscript` command using the `-f` parameter:

```
$ fscript -f /OperationsCenter_install_path/database/scripts/util/adapters.fs

Enter web server host [reason] :
Enter web server port [80]      :
Enter your Formula(r) account userid   : admin
Enter your Formula(r) account password : formula


Adapters:
   0 CIC on qa2sun0: stopped
   1 Eve(tm) on reason: stopped
   2 Formula(r) on hiro (unsecured): stopped
   3 Formula(r) on yt: stopped
   4 IT Masters MasterCell(tm) on rufus: stopped
   5 Icon Library: stopped
   6 Max on rufus: stopped
   7 NetIQ on qaintel3: stopped
   8 NetView(r) on qasun1: stopped
   9 NetView(r) on taz: stopped
   10 Netcool(r) on twee: stopped
   11 OVO on fishhead: stopped
   12 OpenView on bogey: stopped
   13 OpenView on fishbowl: stopped
   14 OpenView on qasun: stopped
   15 PATROL(r) on bogey: stopped
   16 PATROL(r) on fishbowl: stopped
   17 PATROL(r) on hiro: stopped
   18 PATROL(r) on qaintel0: stopped
   19 PATROL(r) on qasun: stopped
   20 PATROL(r) on reason: stopped
   21 PATROL(r) on twee: stopped
   22 PEM on raven: stopped
   23 SPECTRUM(r) on twee: stopped
   24 Script 3DNS: stopped
   25 Script Test 1: stopped
   26 Script Test 2: stopped
   27 Script Test 3: stopped
   28 Script Test 4: stopped
   29 Script Test 5: stopped
   30 Script Test 6: stopped
   31 Script Test 7: stopped
   32 Script Test 8: stopped
   33 Script Test 9: stopped
   34 Script(tm): stopped
   35 T/EC(r)+ on qaintel1: stopped
   36 T/EC(r)+ on rubberneck: stopped
   37 TNG on qant: stopped
   38 Tivoli T/EC(r) on reason, i: Ready for T/EC events
   39 Tivoli T/EC(r) on reason, ii: stopped
   40 Tivoli T/EC(r) on reason, iii: stopped
   41 Tivoli T/EC(r) on reason, iv: stopped
   42 US:EVE:PROD: stopped

Adapter:
```

10.3 Invoking a Script by Module Name

To execute a script in the script repository (located under */OperationsCenter_install_path/*`database/scripts`), enter the module name with the `-m` parameter:

```
# fscript -m util/forceoff

Usage:
    forceoff [ username | Group:groupname | username:sessionid ] [ message ]

Examples:
    forceoff joeuser
    forceoff Group:users
    forceoff admin:42 "Your session was terminated so we can perform system
maintenance."
```

10.4 Invoking a Script with Arguments

Invoking a script many times requires supplying arguments to parameterize the script.

To send arguments to the script (the arguments appear as the `args` array variable), use the `-A` parameter, followed by arguments to be supplied to the script:

```
# fscript -m util/forceoff -A admin

Enter web server host [reason] :
Enter web server port [80]      :
Enter your Formula(r) account userid   : admin
Enter your Formula(r) account password : formula
Found target element: Default Administrator (admin)

Result of forceoff operation:

Warning. You are not able to logout your own session.

There were no sessions active with the given account admin
```

10.5 Using the Interactive Option

To run a script and then return control to the fscript prompt, use the interactive option (`-i`):

```
# fscript -m util/formula -i

Enter web server host [reason] :
Enter web server port [80]      :
Enter your Formula(r) account userid   : admin
Enter your Formula(r) account password : formula
js> list( formula.Administration['formulaServer=Server'].properties )
[object Object]:
   Object Heap Used Memory (KB)
   Adapters
   Auditable
   Formula Server Start Timestamp
   Condition
   Algorithm
   Total Alarms
   Formula Server Has Been Up For
   Element
   Object Heap Free Memory (KB)
   Algorithm Parameters
   Total Element Classes
   Object Heap Allocated Memory (KB)
   Last Reported
   Sessions
   Total Elements
js>
```

10.6 Exiting fscript

The `quit()` method is used to exit fscript with an optional exit code. It is available in all fscript invocations regardless of whether it is interactive or not.

Pass a numeric value to the method to control the exit code of the fscript call. If script does not provide an exit code, the fscript exit code defaults to `0`. In the case of an unhandled exception, the exit code is `-1`.

To exit the fscript prompt, issue `quit()` as a command:

```
js> quit(exitCode)
#
```

Where, *exitCode* is an optional exit code.

11 Usage Scenarios

This section presents several usage scenarios for scripting. They vary from database connectivity, to interacting with a user, to creating sophisticated user interfaces:

11.1 Use Case: Opening a Connection to a JDBC Database

In general, NOC Script can call any Java-based library. There are two ways to access a Java package:

- Use the full package name when accessing Java classes and interfaces.
- Use the `importPackage()` mechanism in JavaScript.

The following example shows how to open a connection to a JDBC* database (in this case a Microsoft* SQL Server* database) using full package names:

```
try
{
   java.lang.Class.forName( 'com.inet.tds.TdsDriver' ) // Get the driver into
memory.
   var props = new java.util.Properties()
   props.setProperty( 'user', 'formula' )
   props.setProperty( 'password', 'sesame' )
   var url = 'jdbc:inetdae:qaintel0:1433'
   var conn = java.sql.DriverManager.getConnection( url, props )
}
catch( Exception )
{
   writeln( 'Could not get connection: ' + Exception )
}
The following example shows how to use importPackage:
importPackage( java.sql )
importPackage( java.util )
importPackage( java.lang )

try
{
   Class.forName( 'com.inet.tds.TdsDriver' ) // Get the driver into memory.
   var props = new Properties()
   props.setProperty( 'user', 'formula' )
   props.setProperty( 'password', 'sesame' )
   var url = 'jdbc:inetdae:qaintel0:1433'
   var conn = DriverManager.getConnection( url, props )
}
catch( Exception )
{
   writeln( 'Could not get connection: ' + Exception )
}
```

11.2 Use Case: Gathering Information from the User for Script Invocation on a Server

In many cases, before an operation executed by a server script can perform an action, it is necessary to gather information by interacting with the user. NOC Script fully supports this type of interaction by:

- Allowing a server-side script to send a script to the client session that invoked it
- Gathering information from the user
- Issuing a “call back” to the server script, supplying the user input to the originating server-side script

Consider the example where a trouble ticket is created for an element that has an outage. The user must supply information to “cut” the ticket, such as a summary of the problem, before the server-side script logic can create a trouble ticket.

A simple prompt() invocation suffices for gathering the summary information from the user. However, invoking the script on the client side raises the issue of applying to the information some logic on the server, which might be the only connectivity resource that can interact with the trouble ticket system. Thus, it might be better to originate the script on the server side. (There is another way to solve this problem, as shown in Section 11.3, “Use Case: Invoking a Server-Side Script from a Client Script,” on page 74.)

The following example, illustrated in detail, describes how to create a server-side script to resolve the problem:

11.2.1 Configuring the Script

To configure this script, copy the following `operations.ini` section to the operations.ini file that resides in the `/`*`OperationsCenter_install_path`*`/database/shadowed` directory:

```
[Enter Trouble Ticket]
description=Trouble Ticket...
context=element
target=dname:root=Elements
permission=manage
type=serverscript
operation=@examples/ticket
```

This operation denotes a right-click menu command that can be invoked on any element that resides in the Elements top-level tree in Operations Center. It is a server-side script, which simply delegates invocation to a script named ticket that resides in the `/`*`OperationsCenter_install_path`*`/database/scripts/examples` directory.

11.2.2 Script File Content

The following is the contents of the ticket script file:

```
////////////////////////////////////////////////////////
// Operation definitions to add to an element: (paste into Operations.ini)
//
// [Enter Trouble Ticket]
// description=Trouble Ticket...
// context=element
// target=dname:root=Elements
// permission=manage
// type=serverscript
// operation=// @debug off \nload( "examples/ticket" );
//

formula.log.info( "Here!" )

// Set our log category
formula.setLogCategory( "Ticket" )

// Log startup
formula.log.info( "Starting trouble ticket script" )

// Create our object which will be the "callback" from the client
// Note: this is standard javascript syntax for creating an object with named
properties/values
var callback =
{
    setTicketInfo: function( reason )
    {
       // We'll log what the user did, to simulate connecting to the ticketing
system.
       formula.log.info( "User created trouble ticket:" )
       formula.log.info( "   Element: " + element )
       formula.log.info( "   Reason: " + reason )

       // Let's notify the user that we did what was asked.
       session.sendMessage( 'Trouble ticket created for ' + element + '.
Content:\n\n' + reason )
    },

    cancel: function()
    {
       formula.log.info( "User cancelled trouble ticket creation" )
       session.sendMessage( 'Trouble ticket cancelled for ' + element + '.' )
    }
}

// We're going to send this callback and a prompt script to the client, which will
// then cause execution to this script through remote proxy.

var clientTicketScript = "\
    // @opt -1 \
    // @debug off \
    var result = prompt( 'Enter trouble ticket information for ' + elementName +
':', \
                         'Trouble Ticket' ) \
    if( ! result ) \
        callback.cancel() \
    else \
        callback.setTicketInfo( result )\
"

//
```

```
// Now, send the dialog script to the user who invoked this script
//
// Note: this script is sent to variables, called "callback" and "elementName"
//       The callback is wrapped via a remote proxy using the
formula.util.makeRemote()
//       function. The element name is simply the element name of the element
//       the user initiated this ticket for.
//
formula.log.info( "Sending dialog script to client" )
session.invokeScript( 'Enter Trouble Ticket',
                      clientTicketScript,
                      [ formula.util.makeRemote( callback ), element.name ],
                      [ 'callback', 'elementName' ] )
formula.log.info( "Done!" )
```

11.2.3 Notes About the Script

Several notes about this script:

- The script declares a “callback” object in JavaScript with two properties that are functions which either: 1) accept the data entered by the user or 2) cancel the operation.
- The actual client script is embedded as a textual string in this script. See this declaration in the section that declares the value of clientTicketScript. If the client-side script is large, remove it from this script and place it in the repository. Refer to the script using only a load() statement or the @ symbol notation.
- The script to execute against the session is invoked using the invokeScript method, which takes the parameters of the script name, the script itself, and optional parameters to send to the script. These are exposed as variables when the session executes the script.
- Prepare the value of the *callback* variable for remote invocation using the formula.util.makeRemote() function, which turns a JavaScript object into an entity that can be remote, suitable for distributed function invocations.
- The client-side script calls the *callback* variable remotely, which results in sending data back to the originating server-side script, or calling the cancel method.
- Both callback functions send a message to the originating session, displaying a message to the user about what happened.

11.3 Use Case: Invoking a Server-Side Script from a Client Script

Sometimes a client-side script needs to invoke a script that normally runs on the Operations Center server. In this scenario, the client might or might not need to pass information to the server-side script.

To invoke a script that normally runs on the Operations Center server:

- Create an operation and assign it to an element in the Operations Center Administration user interface. Consider making this script hidden, because it might not be useful to call unless programmatic input is supplied. This can be done by modifying the operations.ini and adding `hidden=true` to the script.
- The client-side script can then find the element that has the operation defined, hard-coding the element dname to look up the element, if necessary. The perform() method is then called on this element, passing the appropriate parameters.

A few examples:

```
// Find the "server" element.
var server = formula.Administration.findElement( 'formulaServer=Server' )

// Acknowledge an alarm.
element.perform( session, 'Ack', [alarm], [])
```

In the above example, `Ack` is the operation name, the first `[]` argument is an array of alarms, and the second `[]` argument would pass operation arguments if they were required.

```
// Call custom script-based operation to do something interesting for us.
server.perform( session, 'Do That', [], [ 'One Argument', 'Another Argument' ] )
```

In this last example, `Do That` is the operation name, the first `[]` argument is an array of alarms which we left as an empty array since this is not an alarm-based operation, and lastly the second `[]` argument are optional operation arguments that are passed as a string array. The arguments appear in the target operation as the predefined script variable `args`, where `args[0]` and `args[1]` correspond to `One Argument` and `Another Argument`.

11.4 Use Case: Creating User Interface Scripts with Java and JFC/Swing

Use JavaScript to create sophisticated Java user interfaces in JFC/Swing. There is a caveat: declarations are always untyped, so do not declare variables. For example:

- Do not do this:

  ```
  Packages.javax.swing.JButton button = new Packages.javax.swing.JButton( "OK" )
  ```

- Instead, do this:

  ```
  var button = new Packages.javax.swing.JButton( "OK" )
  ```

  Listeners added to GUI components use a slightly different syntax.

There are no GUI designers for JFC/Swing as embedded within JavaScript. However, some GUI designers, such as NetBeans*, work well with JavaScript.

In fact, JavaScript is so adept at creating JFC/Swing user interfaces that Operations Center creates many user interfaces in JavaScript, because it allows dynamically generated code for management-system-specific tasks.

To minimize confusion, Operations Center has attempted to make its JavaScript appear unlike Java, using single-quote (') strings, and no semicolons in JavaScript statements. However, there is nothing wrong with making your code look exactly like Java. For example:

```
new java.awt.Frame( "A Frame Window" );
```

This code compiles in Java and works in JavaScript, with no modifications.

11.4.1 Syntax for Bean-Listener Patterns

Those familiar with JFC/Swing have probably seen or used inner-class declarations for setting up a listener to a component. JavaScript supports a similar declarative syntax for creating listeners to JFC/Swing and AWT components.

An example in Java:

```
final frame = new java.awt.Frame( "A Frame Window" );
frame.addWindowListener
(
   new java.awt.event.WindowAdapter()
   {
      public void windowClosing( java.awt.event.WindowEvent evt )
      {
         frame.setVisible( false );
      }
   }
);
```

The same example in JavaScript:

```
var frame = new java.awt.Frame( "A Frame Window" );
frame.addWindowListener
(
   new java.awt.event.WindowAdapter()
   {
      windowClosing: function( evt )
      {
         frame.setVisible( false );
      }
   }
)
```

Note the similarity with Java: there is still a declaration of an inner class, derived from java.awt.event.WindowAdapter(). However, the declaration uses inline notation of a property of the JavaScript object, namely the windowClosing property, which is itself a function taking one argument.

You can manufacture similar listeners for any JFC/Swing or AWT component that allows for listener registration. However, options are not limited to these interfaces. It is possible to create inner class declarations for almost anything that can be done in Java.

11.4.2 Utilizing NetBeans

NetBeans is an open source project found at http://www.netbeans.org (http://www.netbeans.org), where it can be freely downloaded.

An attractive feature of NetBeans is creating user interfaces with the GUI design tool. Copy or paste these interfaces into a JavaScript script with little or no modifications. As long as the interactivity portion of the user interface design is left out of the NetBeans form, it is possible to create user interfaces for use in NOC Script.

As an example:

1. Consider the following form declaration, taken directly from a user interface create in NetBeans. This user interface is the *Change Password* dialog in Operations Center:

```
/*
 * ChangePasswordDialog.java
 *
 * Created on September 6, 2000, 12:01 PM
 */

package proto;

/**
 *
 * @author  kwester
 * @version
 */
public class ChangePasswordDialog extends javax.swing.JDialog {

    /** Creates new form ChangePasswordDialog */
    public ChangePasswordDialog(java.awt.Frame parent,boolean modal) {
        super (parent, modal);
        initComponents ();
        pack ();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is


     * always regenerated by the FormEditor.
     */
   private void initComponents()//GEN-BEGIN:initComponents
   {
      java.awt.GridBagConstraints gridBagConstraints;

      clientPanel = new javax.swing.JPanel();
      buttonPanel = new javax.swing.JPanel();
      OKButton = new javax.swing.JButton();
      CancelButton = new javax.swing.JButton();
      specifyLabel = new javax.swing.JLabel();
      oldPasswordLabel = new javax.swing.JLabel();
      oldPasswordField = new javax.swing.JPasswordField();
      newPasswordLabel = new javax.swing.JLabel();
      newPasswordField = new javax.swing.JPasswordField();
      newPasswordAgainLabel = new javax.swing.JLabel();
      newPasswordAgainField = new javax.swing.JPasswordField();

      clientPanel.setLayout(new java.awt.GridBagLayout());

      buttonPanel.setLayout(new java.awt.GridLayout(1, 2, 4, 4));

      OKButton.setText("OK");
      buttonPanel.add(OKButton);

      CancelButton.setText("Cancel");
      buttonPanel.add(CancelButton);

      gridBagConstraints = new java.awt.GridBagConstraints();
      gridBagConstraints.gridx = 0;
      gridBagConstraints.gridy = 7;
      gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
      gridBagConstraints.ipadx = 30;
      gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
      gridBagConstraints.weightx = 1.0;
      gridBagConstraints.insets = new java.awt.Insets(24, 0, 8, 8);
      clientPanel.add(buttonPanel, gridBagConstraints);
```

```
        specifyLabel.setText("Enter new password:");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.gridwidth = 4;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.insets = new java.awt.Insets(8, 8, 24, 8);
        clientPanel.add(specifyLabel, gridBagConstraints);

        oldPasswordLabel.setText("Old password:");

    oldPasswordLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
        clientPanel.add(oldPasswordLabel, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 80;
        gridBagConstraints.ipady = 4;

        gridBagConstraints.weightx = 1.0;      gridBagConstraints.insets = new
    java.awt.Insets(4, 0, 0, 24);
        clientPanel.add(oldPasswordField, gridBagConstraints);

        newPasswordLabel.setText("New password:");

    newPasswordLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
        clientPanel.add(newPasswordLabel, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 80;
        gridBagConstraints.ipady = 4;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(4, 0, 0, 24);
        clientPanel.add(newPasswordField, gridBagConstraints);

        newPasswordAgainLabel.setText("New password (again):");

    newPasswordAgainLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT)
    ;
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 5;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
        clientPanel.add(newPasswordAgainLabel, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 5;
```

```
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 80;
        gridBagConstraints.ipady = 4;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(4, 0, 0, 24);
        clientPanel.add(newPasswordAgainField, gridBagConstraints);

        getContentPane().add(clientPanel, java.awt.BorderLayout.CENTER);

    }//GEN-END:initComponents

     /** Closes the dialog */
     /**
     * @param args the command line arguments
     */
     public static void main (String args[]) {
         new ChangePasswordDialog (new javax.swing.JFrame (), true).show ();
     }


    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton OKButton;
    private javax.swing.JLabel newPasswordLabel;
    private javax.swing.JLabel specifyLabel;
    private javax.swing.JPasswordField newPasswordField;
    private javax.swing.JLabel newPasswordAgainLabel;
    private javax.swing.JButton CancelButton;
    private javax.swing.JPasswordField newPasswordAgainField;
    private javax.swing.JPanel clientPanel;
    private javax.swing.JLabel oldPasswordLabel;
    private javax.swing.JPasswordField oldPasswordField;
    // End of variables declaration//GEN-END:variables

}
```

2 Copy or paste the section of code highlighted in bold into a NOC Script script with only no modifications. This enables taking a script, which results in assembling the clientPanel variable in the script, and adding it to the content pane of a JDialog:

```
// Create an owner frame.
frame = new javax.swing.JFrame( 'Hidden' )

// Create a dialog that holds the clientPanel.
dialog = new javax.swing.JDialog( frame, 'Change Password', true )
dialog.getContentPane().add( clientPanel )
dialog.pack()
formula.util.center( dialog )
```

3 Place the following line at the top of the script, to declare the `javax` variable:

```
javax = Packages.javax
```

4 After creating the dialog, create a few listeners to components of the dialog, to ensure interaction with the user:

```
// Add a window listener to automatically close.
dialog.addWindowListener
(
   new java.awt.event.WindowAdapter()
  {
    windowClosing: function( evt )
    {
      dialog.setVisible( false )
    }
  }
)

// Add an OK listener.
OKButton.addActionListener
(
   new java.awt.event.ActionListener()
  {
    actionPerformed: function( evt )
    {
      info( 'You pressed OK' )
    }
  }
)

// Add a Cancel listener.
CancelButton.addActionListener
(
   new java.awt.event.ActionListener()
  {
    actionPerformed: function( evt )
    {
      alert( 'You pressed Cancel' )
    }
  }
)
```

5 The resulting script:

```
// Forward declaration of "javax", since it isn't a predefined package name for
JS
javax = Packages.javax

// BEGIN PASTE FROM NETBEANS

      clientPanel = new javax.swing.JPanel();
      buttonPanel = new javax.swing.JPanel();
      OKButton = new javax.swing.JButton();
      CancelButton = new javax.swing.JButton();
      specifyLabel = new javax.swing.JLabel();
      oldPasswordLabel = new javax.swing.JLabel();
      oldPasswordField = new javax.swing.JPasswordField();
      newPasswordLabel = new javax.swing.JLabel();
      newPasswordField = new javax.swing.JPasswordField();
      newPasswordAgainLabel = new javax.swing.JLabel();
      newPasswordAgainField = new javax.swing.JPasswordField();

      clientPanel.setLayout(new java.awt.GridBagLayout());

      buttonPanel.setLayout(new java.awt.GridLayout(1, 2, 4, 4));

      OKButton.setText("OK");
      buttonPanel.add(OKButton);

      CancelButton.setText("Cancel");
      buttonPanel.add(CancelButton);
```

```
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 7;
        gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
        gridBagConstraints.ipadx = 30;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(24, 0, 8, 8);
        clientPanel.add(buttonPanel, gridBagConstraints);

        specifyLabel.setText("Enter new password:");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.gridwidth = 4;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.insets = new java.awt.Insets(8, 8, 24, 8);
        clientPanel.add(specifyLabel, gridBagConstraints);

        oldPasswordLabel.setText("Old password:");

    oldPasswordLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
        clientPanel.add(oldPasswordLabel, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 80;
        gridBagConstraints.ipady = 4;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(4, 0, 0, 24);
        clientPanel.add(oldPasswordField, gridBagConstraints);

        newPasswordLabel.setText("New password:");

    newPasswordLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
        gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
        clientPanel.add(newPasswordLabel, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 80;
        gridBagConstraints.ipady = 4;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(4, 0, 0, 24);
        clientPanel.add(newPasswordField, gridBagConstraints);

        newPasswordAgainLabel.setText("New password (again):");

    newPasswordAgainLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT)
    ;
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
```

```
      gridBagConstraints.gridy = 5;
      gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
      gridBagConstraints.insets = new java.awt.Insets(4, 24, 0, 8);
      clientPanel.add(newPasswordAgainLabel, gridBagConstraints);

      gridBagConstraints = new java.awt.GridBagConstraints();
      gridBagConstraints.gridx = 1;
      gridBagConstraints.gridy = 5;
      gridBagConstraints.gridwidth = 2;
      gridBagConstraints.gridheight = 2;
      gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
      gridBagConstraints.ipadx = 80;
      gridBagConstraints.ipady = 4;
      gridBagConstraints.weightx = 1.0;
      gridBagConstraints.insets = new java.awt.Insets(4, 0, 0, 24);
      clientPanel.add(newPasswordAgainField, gridBagConstraints);

// END PASTE FROM NETBEANS

// Create an owner frame.
frame = new javax.swing.JFrame( 'Hidden' )

// Create a dialog that holds the clientPanel.
dialog = new javax.swing.JDialog( frame, 'Change Password', true )
dialog.getContentPane().add( clientPanel )
dialog.pack()
formula.util.center( dialog )

// Add a window listener to automatically close.
dialog.addWindowListener
(
   new java.awt.event.WindowAdapter()
  {
    windowClosing: function( evt )
    {
      dialog.setVisible( false )
    }
  }
)

// Add an OK listener.
OKButton.addActionListener
(
   new java.awt.event.ActionListener()
  {
    actionPerformed: function( evt )
    {
      info( 'You pressed OK' )
    }
  }
)

// Add a Cancel listener.
CancelButton.addActionListener
(
   new java.awt.event.ActionListener()
  {
    actionPerformed: function( evt )
    {
      alert( 'You pressed Cancel' )
    }
  }
)

// Show the dialog
dialog.setVisible( true )
```

A The Script Library

A number of scripts are provided in the `/OperationsCenter_install_path/database/scripts` directory. Table A-1 describes the functions of these scripts. See Chapter 2, “Creating and Debugging Scripts,” on page 11 for information about customizing scripts or adding your own to the Script Library.

Table A-1 *Default Scripts Provided in the Script Library*

Script Name	Function
/commands/suppress.fs	Configures the Suppression/Acknowledgement command for an adapter. For details on this command, see *Configuring Suppression and Acknowledgement* in the *Operations Center 5.6 Server Configuration Guide*.
/commands/impactreport.fs	A starting point script example that requires configuration and customizing. Creates a new log category and logs a message for every alarm that impacts the selected service model and for each impacted element. Elements reported on can be filtered by specifying a class filter.
/examples/jfcexample.fs	Example of creating user interface scripts with JFC/Swing.
/examples/ticket.fs	Adds a command on the element operations menu to create a trouble ticket. See Section 11.2, “Use Case: Gathering Information from the User for Script Invocation on a Server,” on page 72 for details.
/jdbc/select.fs	Makes a simple connection and gets a list of tables in the database.
/mail/Action_MailElementAnd AlarmInformation.fs	Used in an automation action that e-mails element and alarm information.
/mail/Action_MailElementInformation.fs	Used in an automation action that e-mails element information.
/mail/mail.fs	Companion script supporting automation mail actions.
/mail/mailalarminfo.fs	Companion script supporting alarm information capture and mailing.
/mail/mailelement.fs	Companion script supporting element information capture and mailing.
/pager/Action_PageElement.fs	Used in an automation action to send element information via a paging gateway.
/servlets/dump.fs	Generates dynamic and static content for a Web browser.
/servlets/login.fs	Stores some of the login logic used to access Operations Center. Used with the dump.fs script.
/servlets/navigator.fs	Contains information used to navigate objects in the system. Used with the dump.fs script.
/tests/array.fs	Creates sample array.
/tests/createorgs.fs	Creates organizations dynamically while the Operations Center server is running.

Script Name	Function
/tests/elementfun.fs	Performs a number of element functions: ◆ Simple comparison function for string values. ◆ Simple function to write an element. ◆ Get a simple element by traversal. ◆ Get a complex element by chaining lookup. ◆ Find an element. ◆ Get a simple organization. ◆ Perform some simple property lookups.
/tests/elementproperties.fs	Display element properties.
/tests/frame.fs	Creates a frame.
/tests/hello.fs	Adds the word “Hello” as an element menu command.
/tests/runtimeerror.fs	Creates sample run time error.
/tests/syntaxerror.fs	Creates sample syntax error.
/util/Action_PostAlarmToTec.fs	Used in an automation action to post alarm information to T/EC.
/util/Action_PostToTec.fs	Used in an automation action to post an event message to T/EC.
/util/adapters.fs	A utility to interactively query and manage Operations Center adapters.
/util/exportacls.fs	Exports the access manager to XML.
/util/f2fhelp.fs	Start/stop/status the named f2f adapter on the named server.
/util/fastmatch.fs	A utility to match a given named regular expression with cached compilation for repetitive matching.
/util/forceoff.fs	A utility to terminate Operations Center sessions for a user or group of users.
/util/formula.fs	Loads and logs in to Operations Center.
/util/image2go.fs	Imports an image file and converts it to the Operations Center GO format.
/util/importacls.fs	Imports the access manager to XML.
/util/login.fs	Starts the browser without showing the main user interface, and allows for command line interaction with the Operations Center server.
/util/orb.fs	Initializes the ORB.
/util/orgs.fs	Enables dynamic organization manipulation. Creates a new organization element within Operations Center, modifies an existing organization or deletes an organization.

Script Name	Function
/util/pause_op.fs	Executes a pop-up a window in the browser which states `YOU ARE IN PAUSE MODE` after the Alarms view window has been in Pause mode for an amount of time that exceeds the preset period. Put both the pause_op and pauseStop_op scripts into the operations store in Operations Center and insert them as client scripts against the Enterprise object. Refer to the comment block at the top of pause_op.fs for more information.
/util/pauseStop_op.fs	Removes the pause_op.fs objects and stops the alarm pause monitoring.
/util/showimpacted.fs	
/util/syscheck.fs	Measures memory in Operations Center and outputs to the formula.trc file.
/util/syscheckstop.fs	Stops the syscheck function.
/util/testorgs.fs	Demonstrates how to use orgs.fs.
/util/urlhelp.fs	Obtains the contents of a given URL string, opens the stream and pulls data from it. Prints what was obtained from the stream and closes the stream.
/util/viewbuilderExporter.fs	Exports the organizations at and below the active element. This script places the output into the `/`*`OperationsCenter_install_path`*`/bin` directory. The file name is `viewbuilderout.xml`. This script places a CSV output file also into the `/`*`OperationsCenter_install_path`*`/bin` directory as `viewbuilderout.csv`.
/util/wall.fs	A utility to send a message to a user or group of users from Operations Center.
/util/template/login.fs	Starts the browser without displaying the main user interface and allows for command line interaction with the Operations Center server.