opentext™

ArcSight ESM

Software Version: 7.6.4

ESM Service Layer (Web Services) Developer's Guide

Document Release Date: January 2023
Software Release Date: January 2023

Legal Notices

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Copyright Notice

Copyright 2001-2023 Open Text.

The only warranties for products and services of Open Text and its affiliates and licensors ("Open Text") are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Open Text shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Trademark Notices

"OpenText" and other Open Text trademarks and service marks are the property of Open Text or its affiliates. All other trademarks or service marks are the property of their respective owners.

Support

Contact Information

Phone	A list of phone numbers is available on the Technical Support Page: https://softwaresupport.softwaregrp.com/support-contact-information
Support Web Site	https://softwaresupport.softwaregrp.com/
ArcSight Product Documentation	https://www.microfocus.com/documentation/arcsight/

Contents

Chapter 1: Introduction

This section includes the following topics:

The ESM expose ESM functionalities as web services. By consuming the exposed web services, you can integrate ESM functionality in your own applications. The use a service-oriented architecture (SOA) that supports multiple web service clients written in different languages.

Specifically, you will have the ability to:

- Run an ESM report and feed it back to your third-party home-grown system
- Create and update cases
- Manage resource groups

The SOA approach enables ESM to support multiple options, for example:

- Developers applying Representational State Transfer (REST) principles can achieve their goals by writing scripts that send HTTP requests, and then parse the responses.
- Java developers can take advantage of the SDK to create REST clients.

SDK Installation Files

Installation files are located at

```
$ARCSIGHT_HOME/utilities/sdk
```

The SDK libaries are located at

```
$ARCSIGHT_HOME/utilities/sdk/lib
```

The SDK provides a set of tools and libraries for Java applications that consume the services in .

- SOAP clients use Simple Object Access Protocol (SOAP) XML messages to send requests to and get responses from the web server over HTTP.
- The Google Web Toolkit (GWT) provides the capability to create user interfaces, use RPC to pass Java objects between the client and the server over HTTP, and more.

For SOAP clients, you need the following web services found in the Manager's folder, `utilities/sdk/lib/`:

- `coma-infrastructure-X.X.X.release.XX.jar`: This web application contains common infrastructure classes used by ESM web services. ServiceBase and Service are the base interface and class to be extended by services in the other web applications, described next. This guide refers to this web application as `coma-infrastructure`.
- `core-ws-client-X.X.X.release.XX.jar`: This web application has client-side JAX-WS classes and provides login services (`loginService`) by returning the authentication token

(authToken) needed to begin consuming a service. The services are designed to be stateless. You will therefore pass the authentication token every time you consume a service. This guide refers to this web application as `core-service`.

- `manager-ws-client-X.X.X.release.XX.jar`: This web service provides the ESM functionality. See "ESM Resources as Web Services" below for a list of certified resources. For each `manager-ws` service, the common parent interface is ResourceDao (data access object). The base class is Resource. This guide refers to this web application as `manager-service`.

ESM Resources as Web Services

This topic provides a lookup table of available ESM services. Details about each web service and their respective web methods are described in the Javadoc files.

The following ESM services are certified for REST applications in this release:

Supported ESM Services

Web Application	Available Web Service
core-ws	LoginService
manager-ws	1. CaseService 2. GroupService 3. ReportService 4. ResourceService 5. SecurityEventService

Javadoc Documentation

> **Note:** To develop REST clients, developers only need the information from the Javadocs about ESM web services.

The Javadocs containing the descriptions of the interfaces, methods, and parameters are found under $ARCSIGHT_HOME/utilities/sdk/lib. The documentation consists of two volumes, and each volume is provided in HTML and PDF formats (X.n refers to a release version):

Javadoc Information

For descriptions on	In this format	Refer to
`core-service` client-side classes (volume 1)	HTML	`core-client-javadoc-X.X.X.release.XX.jar`
	PDF	`core-client-X.X.X.release.XX.pdf`
`manager-service` client-side classes (volume 2)	HTML	`manager-client-javadoc-X.X.X.release.XX.jar`
	PDF	`manager-client-X.X.X.release.XX.pdf`

Subsequent references to the Javadocs will say `core-client` Javadoc and `manager-client` Javadoc, respectively.

Where to Get Additional Documentation

The following OpenText ArcSight publications provide in-depth information about ESM resources:

- *ESM 101*
- *ArcSight Console User's Guide*

Get ESM documentation on the ESM documentation page.

Setting Up Your Development Environment

Following are the requirements to set up your client development environment:

- ArcSight Manager's certificate.

  All exposed ESM services are TLS/SSL-secured, therefore, import the ArcSight Manager's certificate into your development/runtime environment. The certificate option was chosen during ESM installation. It could be a temporary certificate authority (CA), a self-signed certificate, or a signed certificate from a trusted CA. Ask your ArcSight administrator about which certificate option was chosen during installation and import that certificate into your development JRE's `jre/lib/security/cacerts`.

- Additionally for SOAP developers, the web applications:

  - `coma-infrastructure.jar,`

  - `core-ws-client.jar,` and

  - `manager-ws-client.jar.`

    Include these jar files in your Java classpath.

    For REST developers, you only need the Javadocs.

To develop RESTful clients, go to .

Chapter 2: Developing REST Clients

Examples in this section are for REST application development. You can use any client-side technology, for example: URLConnection, Apache's HttpClient, or URL tunneling through any Web browser. Regardless of the client technology you choose, make sure to identify the proper methods, their arguments, and the accepted Content-Type format for your http requests.

This section includes the following topics:

> **Note:** If you are using REST clients to communicate with the ESM , check the vendor's documentation for the presentation of received data. Some clients might have different views which could show the same data differently.

Required Javadoc Documentation

To develop your ESM REST clients, refer to these two Javadoc libraries:

- `core-client-javadoc-X.X.X.release.XX.jar` for information about ESM core services such as LoginService. This Javadoc name is shortened to `core-client` Javadoc.
- `manager-client-javadoc-X.X.X.release.XX.jar` for information on ESM resource services, such as CaseService. See "Supported ESM Services" on page 6 for a complete list. This Javadoc name is shortened to `manager-client` Javadoc.

The libraries are found in `$ARCSIGHT_HOME/utilities/sdk/lib`. This directory also contains the PDF versions of the Javadocs.

Depending on your selected type of REST communication (for example, for `Apache HttpClient`), you might need additional third-party libraries. Find them in

`utilities/sdk/examples/TestKit/third-party-libs-X.X.X.release.XX.jar`

Conventions for Some Returned Values

When requesting resource services, you might get big negative numbers in place of NULL database fields. Depending on field type, returned values would be one of:

Field Type	Returned Value in Place of NULL
integer	-2147483648 (Integer.MIN_VALUE)
long	-9223372036854775808 (Long.MIN_VALUE)
double	5e-324 (Double.MIN_VALUE)

When you see these values, treat them as NULL fields.

Byte Array Representations

IP addresses are presented in different ways.

- On a UI, the values are presented as user-readable strings, for example:

IPv4 address	192.0.2.0
IPv6 address	2001:0db8:85a3:0042:1000:8a2e:0370:7334

- Internally, the same data could be stored as byte array. For example:

IPv4 address	[-64, 0, 2, 0]
IPv6 address	[32, 1, 13, -72, -123, -93, 0, 66, 16, 0, -118, 46, 3, 112, 115, 52]

- Again for the same data, the API output byte array is shown as base64 encoded string. For example:

IPv4 address	"wAACAA=="
IPv6 address	"IAENuIWjAEIQAIouA3BzNA=="

Below is an example of a data conversion using javax.xml.bind.DatatypeConverter:

```java
import javax.xml.bind.DatatypeConverter;

  private void testIpAddressConversionFor(String address) {
    IPAddress ipAddress = new IPAddress(address);
    byte[] originalBytes = ipAddress.getAddress();
    System.out.println("originalBytes=\"" + Arrays.toString(originalBytes) + "\"");

    String base64Encoded = DatatypeConverter.printBase64Binary(originalBytes);
    System.out.println("base64Encoded=\"" + base64Encoded + "\"");

    byte[] base64Decoded = DatatypeConverter.parseBase64Binary(base64Encoded);
    System.out.println("base64Decoded=\"" + Arrays.toString(base64Decoded) + "\"");

}
```

Below are examples of the IP address values resulting from the conversion. The IPv6 address examples include a complete and simplified address format:

```
IPAddress "192.0.2.0" of type IPv4
 , ivp4Long=3221225984
 , asString="192.0.2.0"
 , byteAddress=[-64, 0, 2, 0]"
 , byteAddress (64Encoded)="wAACAA=="
 , byteAddressForDb="00000000000000000000FFFFC0000200"
```

```
IPaddress "203.0.113.0" of type IPv4
 , ivp4Long=3405803776
 , asString="203.0.113.0"
 , byteAddress="[-53, 0, 113, 0]"
 , byteAddress (64Encoded)="ywBxAA=="
 , byteAddressForDb="00000000000000000000FFFFCB007100"
```

```
IPaddress "2001:0db8:85a3:0042:1000:8a2e:0370:7334" of type IPv6
 , ivp4Long=N/A
 , asString="2001:0db8:85a3:0042:1000:8a2e:0370:7334"
 , byteAddress="[32, 1, 13, -72, -123, -93, 0, 66, 16, 0, -118, 46, 3, 112, 115, 52]"
 , byteAddress (64Encoded)="IAENuIWjAEIQAIouA3BzNA=="
 , byteAddressForDb="20010DB885A3004210008A2E03707334"
```

Finding a List of Services

This section describes how to list services under the following web applications hosted by the ESM :

- core-service ("Listing Services from core-service" below)
- manager-service ("Listing Services from manager-service" on page 13)

You can get information about these web applications by typing the URL in your browser (provided later) or by opening the Javadoc for the specific web application.

Listing Services from core-service

You can list services either through the browser or the corresponding Javadoc.

To get a list of services from `core-service`:

Use the following URL:

https://*myhost*:8443/www/core-service/services/listServices

Replace *myhost* with the IP address of your ArcSight Manager host.

Following is an example of the List Services landing page for the `core-service` web application. Scroll down the page and locate the service of interest.

Sample Landing Page for `core-service` List Services:
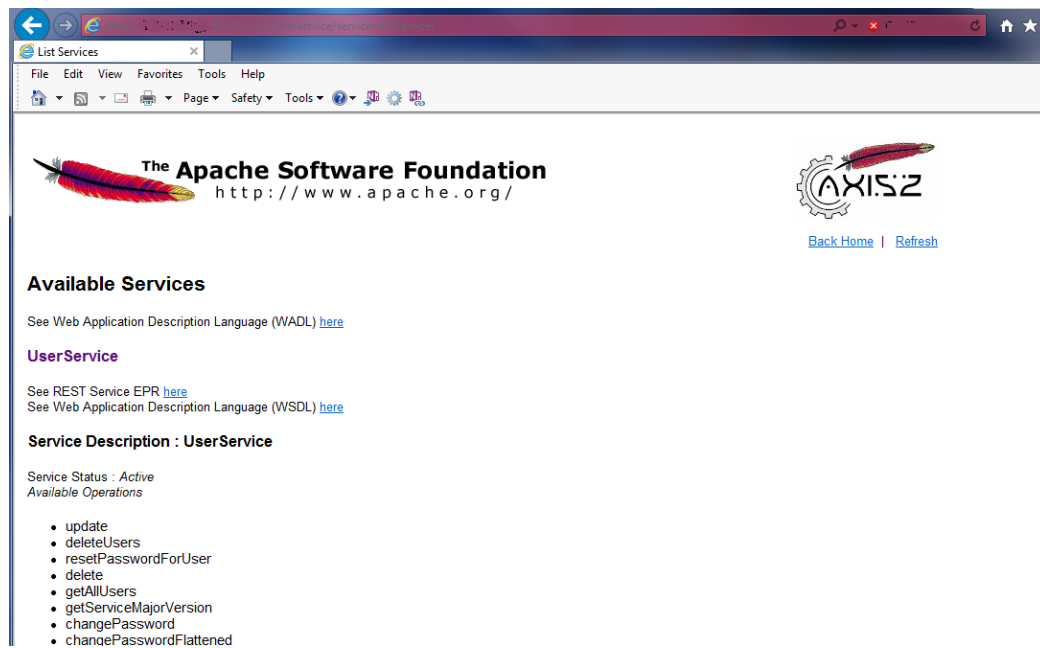
> **Note:** Although three services are listed, only `LoginService` is supported in this release.

In addition to the list of available services and their methods, the page also contains links to the web application description language (WADL) and web services description language (WSDL). The WADL and WSDL can be used for auto-generation of client classes through available third-party tools for a variety of programming languages. For details on WADLs and WSDLs, refer to informative links such as:

- WADL: https://wadl.java.net/
- WSDL: http://www.w3.org/TR/wsdl20/

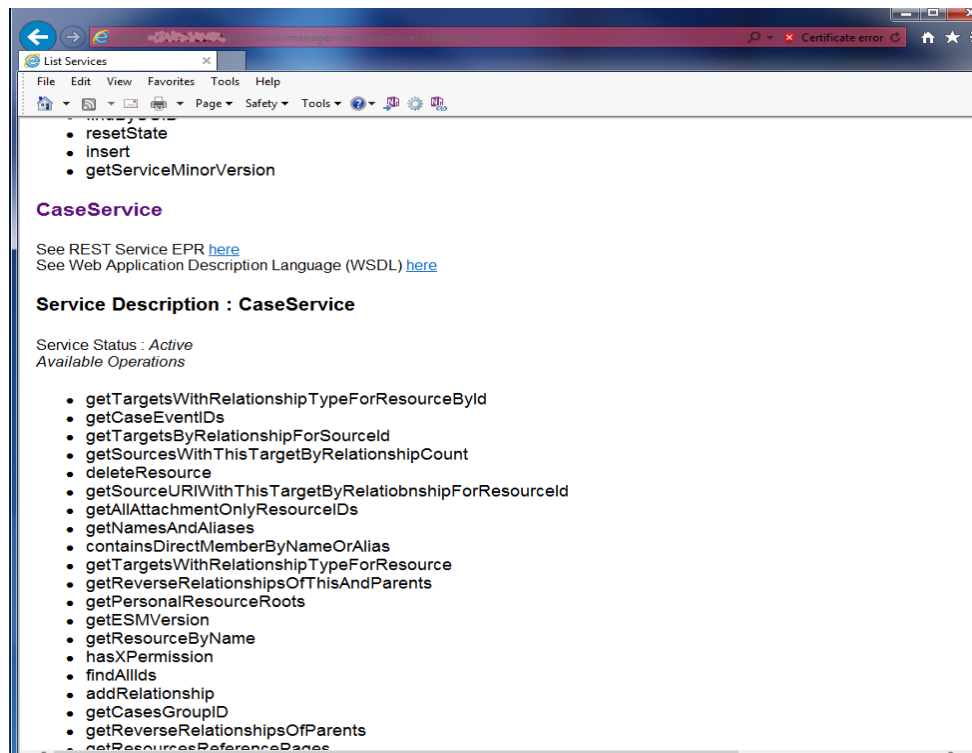**To open the core-client Javadoc for core-service:**

1. Extract `core-client-javadoc-X.X.X.release.XX.jar`.

2. Double-click `index.html`.

3. Expand the `rest` package (`com.arcsight.product.core.service.v1.rest`) to display its classes.

   The supported class (service) is displayed.

4. Select the service, in this case, `LoginServiceImpl`.

   Methods for the service are displayed, as shown:

Sample Javadoc Page for core-service:

Listing Services from manager-service

For ESM resources, get the services from the `manager-service` web application. Use the following URL:

https://*myhost*:8443/www/manager-service/services/listServices

Replace *myhost* with the IP address of your ArcSight Manager host.

Following is an example of the List Services landing page for the `manager-service` web application. Scroll down the page and locate the service of interest, as in the example showing CaseService:

Sample List Services Page for `manager-service`:

> **Note:** Multiple services are listed, but only a limited number of services are supported in this release. Refer to the list in "ESM Resources as Web Services" on page 6.

To open the manager-client Javadoc for manager-service:

1. Extract `manager-client-javadoc-X.X.X.release.XXX.jar`.

2. Double-click `index.html`.

3. Expand the `rest` package
   (`com.arcsight.product.manager.resource.service.v1.rest`) to display its classes.

   Only the supported classes (services) are displayed.

4. Select the service of interest, for example, `CaseServiceImpl`.

   Methods for the service are displayed, as shown.

Sample Javadoc Page for manager-service:

Using Proper Namespaces

Namespaces are used by server-side code to identify the proper services and attributes. This topic provides the namespaces to use in POST requests. The information in this section is provided for XML and JSON request bodies.

> **Note:** Keep the following in mind concerning namespaces:
>
> - Namespaces are unique for each service even if multiple services use the same method names.
> - The XML request body requires the full namespace as provided in "Namespaces for core-service " below and "Namespaces for manager-service " on the next page.
> - If the body contains different resources (case, user, report), make sure to use the corresponding namespace prefixes for the resources. Not all resources have namespaces. Consult the descriptions in the Javadoc for resource details, including namespace information.

Namespaces for core-service

The following table lists the JSON and XML namespaces for the service in core-service. Only the supported service is shown.

core-service Namespaces

Service	JSON	XML prefix	Full XML namespace name
LoginService	log	ns3	xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/ loginService/"

Namespaces for manager-service

The following table lists the JSON and XML namespaces for services in `manager-service`. Only the supported services are shown.

manager-service Namespaces

Service	JSON	XML prefix	Full XML namespace name
CaseService	cas	ns4	xmnls:ns4="http//ws.v1.service.resource .manager.product.arcsight.com/caseService/"
GroupService	gro	ns15	xmlns:ns15="http://ws.v1.service.resource. .manager.product.arcsight.com/groupService/"
ReportService		ns22	xmlns:ns22="http://ws.v1.service.manager. product.arcsight.com/reportService/"
ResourceService		ns23	xmlns:ns23="http://ws.v1.service.manager. product.arcsight.com/resourceService/"
SecurityEventService		n25	xmlns:ns25="http://ws.v1.service.manager. product.arcsight.com/securityEventService/"

XML Example

The following example shows a Case resource by ID request for XML:

```
<?xml version="1.0"?>

<ns4:getResourceById
xmlns:ns4="http://ws.v1.service.resource.manager.product.arcsight.com/caseService/">
    <ns4:authToken>i2plmQocO9Gqz9x2dReLBFoSdbkMtntWtqiUunO8CZs.</ns4:authToken>
     <ns4:resourceId>774c7vkQBABCAGdbEeuV1Aw==</ns4:resourceId>
</ns4:getResourceById>
```

JSON Example

The following shows an example for JSON Content-Type:

```
{
  "cas.addCaseEvents" : {
    "cas.authToken" : "value",
    "cas.caseID" : "value",
    "cas.eventIDs" : [ "a1", "a2", "a3" ]
  }
}
```

Preparing the URL

Prepare the request URL using the pattern
`<PROTOCOL>://<HOST>:<PORT>/www/<MODULE>/rest/<SERVICE>/<METHOD>`, for example:

https://192.0.2.0:8443/www/manager-service/rest/DashboardService/getDashboardIfNewer

Identifying the Supported Content-Types in Requests

In preparing your request body, identify the proper Content-Type format. There are several available types; ESM web services support the following Content-Types to use in your request body:

- `application/x-www-form-urlencoded`
- `application/xml`
- `application/json`

Content-Types accepted by ESM is described in the client-side Javadocs for every web-service method. For example, in the Javadoc, the Content-Type for CaseService's `GetPersonalGroup` web method is indicated as `x-www-form-urlencoded`:

POST

```
Consumes({"application/x-www-form-urlencoded"})
URL: https://HOST:PORT/www/manager-service/rest/CaseService/getPersonalGroup
Normal response code: 200
```

After you specify the Content-Type, you then prepare the request body. One way to prepare a JSON or XML presentation for the resource is to start by requesting that resource using some get method. This way, you receive correct representation of the resource from ESM. Another option is to use JSON prototypes from the `manager-client` Javadoc, which contains prototypes for all model classes describing resources of different types, as presented in "Preparing the Request Body" on page 20.

Selecting the Content-Type of the Response

As with requests, responses can come in different Content-Type formats. By default, the response is in XML. You can request your data to be returned in JSON by following these instructions:

1. Add **&alt=json** to the URL when submitting the request through the Web browser:

   ```
   https://<HOST>:8443/www/manager-service/rest/CaseService/getCaseEventIDs?
   caseID=774c7vkQBABCAGdbEeuV1Aw%3D%3D&alt=json&authToken=
   ```

2. Add **accept** in the header:

   ```
   Map<String, String> requestProperties = new HashMap<String, String>();
   requestProperties.put("accept", "application/json");
   String resultJson = executor.sendGet(getProtocol(), getServerName(), getPort(),
       webService.getModuleName(), webService.getServiceName(), webServiceMethod, requestData, requestProperties);
   ```

The above example adds the standard HTTP header **accept** to the HTTP request with the value, either **application/json** or **application/xml**. Depending on your technology of choice, you will have different approaches, as described in the next section.

Using Additional Headers in Requests

The following example shows how to specify additional headers for **HttpURLConnection** and **Apache HttpClient**. The header is first stored into an auxiliary map (in the example, the map is called `requestProperties`). Use the same approach when you need to add more than one header.

Example Request for HttpURLConnection:

```
package com.arcsight.web.httpclients;

public class URLConnectionExecutorClient implements ExecutorClient {

HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    for (Map.Entry<String, String> nextParam : requestProperties.entrySet()) {
        connection.setRequestProperty(nextParam.getKey(), nextParam.getValue());
    }
```

Example Request for Apache HttpClient:

```
package com.arcsight.web.httpclients;

public class HttpExecutorClient implements ExecutorClient {

    HttpPost request = new HttpPost(urlstr);
    for (Map.Entry<String, String> nextParam : requestProperties.entrySet()) {
request.addHeader(nextParam.getKey(), nextParam.getValue());
    }
```

Returned data

Following is a partial example of the returned data in JSON:

```
{
   "cas.getResourceByIdResponse": {
     "cas.return": {
       "attributeInitializationInProgress": false,
       "createdTime": {
         "day": 13,
         "hour": 18,
         "milliSecond": 720,
         "minute": 36,
         "month": 2,
         "second": 30,
         "timezoneID": "US/Pacific",
         "year": 2014
       },
       "createdTimestamp": 1394760990720,
       "creatorName": "admin",
       "deprecated": false,
       "disabled": false,
       "inCache": false,
       "inactive": false,
       "initialized": true,
       "isAdditionalLoaded": false,
       "localID": 30064771073,
       "modificationCount": 11,
       "modifiedTime": {
         "day": 25,
         "hour": 14,
         "milliSecond": 206,
         "minute": 30,
         "month": 2,
         "second": 39,
```

```
        "timezoneID": "US/Pacific",
        "year": 2014

        ...
    }
  }
}
```

Preparing the Request Body

This topic describes some rules to guide you in preparing the request body.

Topics in this section:

-
-
-

Using Simple Parameters Passed in x-www-form-urlencoded Format

For POST commands with `x-www-form-urlencoded` Content-Type, the body of the HTTP message sent to the server is one big query string, where name-value pairs are separated by ampersand (**&**), and names are separated from values by the equal symbol (**=**).

Make sure the web service method accepts POSTs with `x-www-form-urlencoded` Content-Type. You can verify this by checking the method definition in the Javadoc, which should look something like this:

```
POST    Consumes({"application/x-www-form-urlencoded"})
    URL: https://HOST:PORT/www/manager-
service/rest/CaseService/getPersonalGroup
    Normal response code: 200
```

For such cases, provide parameters in the form of simple URL-encoded name-value pairs, as in `authToken` and `caseID` below.

```
HashMap<String, String> params = new HashMap<String, String>();params.put
("authToken", authToken);
params.put("caseID", caseId);

boolean first = true;StringBuffer sb = new StringBuffer();
for (Map.Entry<String, String> param : params.entrySet()) {
    String keystr = URLEncoder.encode(param.getKey(), "UTF-8");
    String paramstr = URLEncoder.encode(param.getValue(), "UTF-8");
```

```
    if ( first ) { first = false;
    } else {
        sb.append("&");
    }
    sb.append(keystr).append("=").append(paramstr);
}
String paramStr = sb.toString();
```

In the above example, the string `paramStr` at the end of the code block constitutes the request body you want to send.

If all method parameters are simple Java objects or primitives, the correspondent web service method might also accept GET requests with all parameters added directly into URL. Use the standard format for preparing such URL:

```
<PROTOCOL>://<HOST>:<PORT>/www/<MODULE>/<SERVICE>/<METHOD>?parameters
```

where parameters are key-value pairs separated by ampersand (&) with URL-encoded values. For example:

```
https://<HOST>:8443/www/core-
service/rest/LoginService/login?login=admin&password=password
https://<HOST>:8443/www/manager-
service/rest/CaseService/getCaseEventIDs?caseID=774c7vkQBABCAGdbEeuV1Aw%3D%3D&
alt=json&authToken=
```

> ✔ **Tip:** Do not use `GET` to send sensitive information like credentials. Instead, use a similar `POST` request where passed parameters are included in the request body and protected by the SSL protocol.

Using Complex Parameters

When parameters of a web service method are complex Java objects, then most likely the method will be declared with one wrapper class containing all parameters, for example:

```
@Path("getDashboardIfNewer")
@POST
@Consumes({"application/xml","text/xml","application/json"})
public
com.arcsight.product.manager.resource.service.v1.axis2.jaxws.DashboardServiceG
etDashboardIfNewerResponse
getDashboardIfNewer
(com.arcsight.product.manager.resource.service.v1.axis2.jaxws.DashboardService
GetDashboardIfNewer value)
...
```

This topic provides guidelines on how to prepare the request body for such requests:

- Preparing the Request Body Manually
- Using Client-Side SDK Web Applications

Preparing the Request Body Manually

The procedures described in this section will be rarely used, but provided for your reference. In most cases, it would be sufficient to adjust the JSON prototype for the request body (see Step 3: If the method parameters are complex Java objects:).

You need the Javadocs for this procedure. The specific example used here calls for the `manager-client` Javadoc. See "Javadoc Documentation" on page 6.

Step 1: Obtain the namespaces for XML requests

For XML requests, obtain the namespace assigned to the service ("Using Proper Namespaces" on page 15). Start the request body with the method's name as defined in `Post` annotation, starting with namespace's prefix. While the prefix is just an abbreviation to shorten the request body, you must define the prefix by pointing to the required namespace in the request. The abbreviated namespace is declared in the request body, shown in the following example as **ns7**:

```
<ns7:getDashboardIfNewer
xmlns:ns7
="http://ws.v1.service.resource.manager.product.arcsight.com/dashboardServic
e/">
     //
</ns7:getDashboardIfNewer>
```

Step 2: Add XML elements for method parameters

Add the XML elements for the method parameters. Consult the Javadoc for the definitions of method parameters. Also refer to "Using Proper Namespaces" on page 15.

All simple Java objects (such as String or Boolean) or primitives go with the service namespace:

```
<ns7:authToken>mNxMd5Lpy0NgPwI3cTE_4op952fme1Nb3ZsyznGArzA.</ns7:authToken>
```

Following is a similar block for JSON requests (**das** is the JSON namespace for DashboardService). Use the JSON namespace exactly as it was defined in ESM without any additional declarations in the request body:

```
"das.authToken" : "mNxMd5Lpy0NgPwI3cTE_4op952fme1Nb3ZsyznGArzA.",
```

Step 3: If the method parameters are complex Java objects:

Step 3a: Analyze the definitions

Analyze their definitions and note all @XmlElements, because you will add XML or JSON blocks for those members. For example, class DashboardServiceGetDashboardIfNewer is defined as

```
public class DashboardServiceGetDashboardIfNewer {

    @XmlElement(name = "authToken", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/dashboardServic
e/")
    private String authToken;
    @XmlElement(name = "currentSignature", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/dashboardServic
e/")
    private
com.arcsight.product.manager.resource.service.v1.model.ResourceSignature
currentSignature;
```

Step 3b: Specify values for the parameters

To specify values for the parameters, use proper namespaces and names (for example, authToken and currentSignature) as they were declared in the @XmlElement annotation. Depending on your Content-Type, you would have one of the following:

- **Prepared request body in XML (partial):**

```
<ns7:getDashboardIfNewer
xmlns:ns7="http://ws.v1.service.resource.manager.product.arcsight.com/dash
boardService/">
    <ns7:authToken>mNxMd5Lpy0NgPwI3cTE_
4op952fme1Nb3ZsyznGArzA.</ns7:authToken>
    <ns7:currentSignature>

        // .. to be completed below
    </ns7:currentSignature>
</ns7:getDashboardIfNewer>
```

- **Prepared request body in JSON (partial):**

```
{
"das.getDashboardIfNewer" : {
    "das.authToken" : "mNxMd5Lpy0NgPwI3cTE_4op952fme1Nb3ZsyznGArzA.",
      "das.currentSignature" : {
         // .. to be completed below
      }
    }
  }
```

> ✓ **Tip:** To bypass Step 3a: Analyze the definitions and Step 3b: Specify values for the parameters use the JSON prototype from the `manager-client` Javadoc. Even so, you still need to provide blocks for nested resources, described next.

Step 3c: Add details

Add details for nested objects, if any. Each time, consult the description of the corresponding Java object. For this example, we need to add the description of `currentSignature`, which represents the class `ResourceSignature`. So go to the definition of `ResourceSignature` class:

```java
@Model
public class ResourceSignature implements Serializable {
    private static final long serialVersionUID = 1L;

    private String id;
    private long modificationCount;
    // ... truncated
```

Here, we don't have any JAXB annotations, so add the values to the request body using names of corresponding class members. That gives the final request body for XML as:

```xml
<ns7:getDashboardIfNewer
xmlns:ns7="http://ws.v1.service.resource.manager.product.arcsight.com/dashboardService/">
<ns7:authToken>mNxMd5Lpy0NgPwI3cTE_4op952fme1Nb3ZsyznGArzA.</ns7:authToken>
<ns7:currentSignature>
<id>FztgORycBABDy1m0rly+irg==</id>
<modificationCount>0</modificationCount>
</ns7:currentSignature>
```

```xml
</ns7:getDashboardIfNewer>
```

Following is a similar request in JSON format:

```json
{
"das.getDashboardIfNewer" : {
        "das.authToken" : "mNxMd5Lpy0NgPwI3cTE_4op952fme1Nb3ZsyznGArzA.",
        "das.currentSignature" : {
            "id" : "FztgORycBABDy1m0rly+irg==",
            "modificationCount" : 0
        }
    }
}
```

Another way to find JSON prototypes for Model classes like ResourceSignature is by using the corresponding information in the Javadoc.

Look for the link `JSON prototype` in the class definition, for example:

```
Class Case

java.lang.Object
    com.arcsight.product.manager.resource.service.v1.model.Resource
     com.arcsight.product.manager.resource.service.v1.model.Case

public class Case extends Resource implements java.io.Serializable
This represents a Case resource

Author:
    Seva Yakhontov
    JSON prototype

See also:Serialized Form
```

Using Client-Side SDK Web Applications

Even though you can develop RESTful clients without any additional libraries, you might find the ESM SDK libaries designed for SOAP clients useful when working with complex parameters.

Here's how to prepare the request body with help of the SDK:

1. Include into your project the provided SDK web application:

   ```
   utilities/sdk/lib/manager-ws-client-X.X.X.release.XX.jar
   ```

2. In your Java application, use the corresponding request object from the SDK. Set all members that you want to be included into the request:

   ```
   com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceUpdateGroup requestObject
       = new com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceUpdateGroup();
   requestObject.setAuthToken(authToken);
   requestObject.setAddList(Arrays.asList(new String[] {"add1", "add2"} ));
   requestObject.setGroup(testGroup);
   requestObject.setRemoveList(Arrays.asList(new String[] {"remove1", "remove2"} ));
   ```

3. Use third-party libraries (such as Jettison from http://jettison.codehaus.org/) to convert the object to JSON. These libraries automatically convert the request into JSON representation with regard to JAXB annotations in the corresponding Java classes. For this example, you would get:

   ```
   {
   "gro.updateGroup": {
   "gro.authToken": "PRVBHWp2ZdVjX728lmP1fYWLG3TK-eiIJYc03w_nIlE.",
   "gro.group": {
   "attributeInitializationInProgress": false,
   "createdTimestamp": -1,
   "deprecated": false,
   "disabled": false,
   "inCache": true,
   "inactive": false,
   "initialized": false,
   ```

```
"isAdditionalLoaded": false,
"localID": -1,
"modificationCount": 0,
"modifiedTimestamp": -1,
"name": "testGroup",
"state": 0,
"type": 0,
"containedResourceType": 0,
"subGroupCount": 0,
"virtual": false
},
"gro.addList": [
"add1",
"add2"
],
"gro.removeList": [
"remove1",
"remove2"
]
}
}
```

Use the converted format as your JSON request body.

Special Case: Specifying a Collection of Values

To specify set of similar values (a collection) in the request, add as many similar blocks as the number of these values for XML; or use array syntax for JSON. For example, we will use the web service method `getSecurityEvents`. This method returns presentations of the SecutityEvents for all specified IDs. This method has only one parameter of the type `SecurityEventsServiceGetSecurityEvents`, a Java class that accepts a collection of IDs.

```
public class SecurityEventServiceGetSecurityEvents {
    @XmlElement(name = "authToken", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/")
    private String authToken;
    @XmlElement(name = "ids", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/")
    private List<Long> ids;
    @XmlElement(name = "timeField", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/")
    private com.arcsight.product.manager.resource.service.v1.model.event.ArcField timeField;
    @XmlElement(name = "startMillis", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/")
    private long startMillis;
    @XmlElement(name = "endMillis", namespace =
"http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/")
    private long endMillis;
...
```

XML

If you then need to request representations for three `SecurityEvents`, add the three `ids` elements as shown:

```
<?xml version="1.0"?>
<ns25:getSecurityEvents
xmlns:ns25="http://ws.v1.service.resource.manager.product.arcsight.com/securityEventService/">
     <ns25:authToken>n3j_hj9o8e4OqFJAjikpAAHVRW9vv3f3PvYPIB5sgS8.</ns25:authToken>
     <ns25:ids>1270001</ns25:ids>
     <ns25:ids>1270002</ns25:ids>
     <ns25:ids>1270003</ns25:ids>
     <ns25:startMillis>-1</ns25:startMillis>
     <ns25:endMillis>-1</ns25:endMillis>
</ns25:getSecurityEvents>
```

JSON

For JSON, you would use:

```
{
  "sev.getSecurityEvents" : {
    "sev.authToken" : "n3j_hj9o8e4OqFJAjikpAAHVRW9vv3f3PvYPIB5sgS8.",
    "sev.ids" : [1270001, 1270002, 1270003],
    "sev.startMillis" : "-1",
    "sev.endMillis" : "-1",
  }
}
```

Sending HTTP Requests

This section provides examples on sending HTTP requests. The provided examples use auxilliary classes and interfaces that are not required. Your client-side application can be designed differently; the examples here just illustrate the usage of standard Java URLConnection and Apache HttpClient for the ESM APIs.

Topics in this section:

About Common Classes

This section provides the following optional examples on the following common classes. These are for information only, you are not required to use them.

- CommandResult Class
- ExecutorClient Class

CommandResult Class

The CommandResult class describes the results from the HTTP response that comes back. This class also includes statusCode and error message (if any), which allows other classes to handle accordingly.

```
public class CommandResult {
    /**
    * Status code returned by Web-server for the corresponding ServletRequest
    */
  private final int statusCode;
    /**
    * Returned response
    */
  private final String response;
    /**
    * Returned error description (if any)
    */
  private final String error;

  // truncated
}
```

ExecutorClient Class

The ExecutorClient class contains the definitions of various clients that send HTTP requests:

```
public interface ExecutorClient {
    /**
    * Submits GET request to the specified <code>urlstr</code>, reads the
response,
    * and returns it to the caller.
    *
    * @param urlstr destination URL (e.g.
    *https://<HOST>:8443/www/core-service/rest/LoginService/login)
    * @param requestProperties key/value entries to set request headers
    * @return response description of response returned by the server
```

```
      */
   public CommandResult sendGet(String urlstr, Map<String, String>
requestProperties)
   throws IOException;
```

```
    /**
    * Submits POST request to the specified <code>urlstr</code>, reads the
response,
    * and returns it to the caller.
    *
    * @param urlstr destination URL (e.g.
    https://<HOST>:8443/www/core-service/rest/LoginService/login)
    * @param requestProperties key/value entries to set request headers
    * @param requestBody string presentation of request body
    * @param contentType MIME type of request body
    * @return response description of response returned by the server
    */
   public CommandResult sendPost(String urlstr, Map<String, String>
requestProperties,
             String requestBody, String contentType) throws IOException;
}
```

Sending HTTP Requests with URLConnection

The examples uses the URLConnectionExecutorClient class that implements
ExecutorClient. Each step is followed by code samples.

To GET Data

This example submits the GET request to the specified URL (urlstr), reads the response, and
returns the response to the caller.

```
/**
 * @param urlstr destination URL (for example
 * https://<HOST>:8443/www/core-service/rest/LoginService/login)
 * @param requestProperties key/value entries to set request headers
 * @return response description of response returned by the server
 */
public CommandResult sendGet(String urlstr, Map<String, String>
requestProperties)
throws IOException {
    // 1. Prepare the connection
    URL url = new URL(urlstr);
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();

    // 2. Set optional headers (for example to request data in JSON format)
    for (Map.Entry<String, String> nextParam : requestProperties.entrySet()) {
```

```
        connection.setRequestProperty(nextParam.getKey(), nextParam.getValue
());
    }
    // 3. Automatically submit the request by reading the statusCode/response
    int responseCode = connection.getResponseCode();
    String responseStr = "";
    String errMsg = "";
    try {
        if ( HttpURLConnection.HTTP_NO_CONTENT != responseCode ) {
            BufferedReader in = new BufferedReader(new
            InputStreamReader(connection.getInputStream()));
            String line;
            StringBuffer sb = new StringBuffer();
                while ((line = in.readLine()) != null) {
                sb.append(line);
            }
            responseStr = sb.toString();
            in.close();
        }
    } catch (IOException e) {
        // Read errMsg from connection.getErrorStream();
        throw e;
    }
    return new CommandResult(responseCode, responseStr, errMsg);
}
```

To POST Data

This example submits the POST request to the specified URL (urlstr), reads the response, and
returns the response to the caller.

```
/**
 * @param urlstr destination URL (for example
 * https://<HOST>:8443/www/core-service/rest/LoginService/login)
 * @param requestProperties key/value entries to set request headers
 * @param requestBody string presentation of request body
 * @param contentType MIME type of request body
 * @return response description of response returned by the server
 */
```

```
public CommandResult sendPost(String urlstr, Map<String, String>
requestProperties,
        String requestBody, String contentType) throws IOException {

    // 1. Prepare the connection
    URL url = new URL(urlstr);
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    connection.setRequestMethod("POST");
```

```
    connection.setDoOutput(true);
    connection.setRequestProperty("Content-Type", contentType);
    // 2. Set optional headers (e.g. to request data in JSON format)
    for (Map.Entry<String, String> nextParam : requestProperties.entrySet())
{
        connection.setRequestProperty(nextParam.getKey(), nextParam.getValue
());
    }
    // 3. Submit the request
    BufferedWriter writer = new BufferedWriter(new
        OutputStreamWriter(connection.getOutputStream()));
    writer.write(requestBody);
    writer.flush();
    writer.close();
    // 4. Read the response
    String responseStr = "";
    String errMsg = "";
    int responseCode = connection.getResponseCode();
```

```
    try {
       if ( HttpURLConnection.HTTP_NO_CONTENT != responseCode ) {
          BufferedReader inreader = new BufferedReader(new
               InputStreamReader(connection.getInputStream()));
          StringBuffer sb = new StringBuffer();
          String line;
          while ((line = inreader.readLine()) != null) {
               sb.append(line);
          }
          responseStr = sb.toString();
       }
    } catch (IOException e) {
       // Read errMsg from connection.getErrorStream();
       throw e;
    }
    return new CommandResult(responseCode, responseStr, errMsg);
}
```

Sending HTTP Requests with Apache HttpClient

The following example uses the `HTTPExecutorClient` class that implements `ExecutorClient`.
Each step is followed by code samples.

To GET Data

This example submits the GET request to the specified URL (urlstr), reads the response, and returns the response to the caller.

```
/**
 * @param urlstr destination URL (for example
 * https://<HOST>:8443/www/core-service/rest/LoginService/login)
 * @param requestProperties key/value entries to set request headers
 * @return response description of response returned by the server
 * */
public CommandResult sendGet(String urlstr, Map<String, String>
requestProperties)
throws IOException {
    HttpClient client = new DefaultHttpClient();
    // 1. Prepare the connection
    HttpGet request = new HttpGet(urlstr);
    // 2. Set optional headers (for example to request data in JSON format)
    for (Map.Entry<String, String> nextParam : requestProperties.entrySet()) {
        request.addHeader(nextParam.getKey(), nextParam.getValue());
    }
    // 3. Submit the request
    HttpResponse response = client.execute(request);
    // 4. Read the response
    String responseStr = "";
    String errMsg = "";
    int responseCode = response.getStatusLine().getStatusCode();
    try {
        if ( HttpStatus.SC_OK !=  responseCode ) {
            BufferedReader inreader = new BufferedReader(new InputStreamReader
(response.getEntity().getContent()));
            StringBuffer sb = new StringBuffer();
            String line;
            while ((line = inreader.readLine()) != null) {
                sb.append(line);
            }
            String result = sb.toString();
            inreader.close();
        }
    } catch (IOException e) {
        // Read errMsg from connection.getErrorStream();
```

```
        throw e;
    }
    return new CommandResult(responseCode, responseStr, errMsg);
}
```

To POST Data

To POST data, start with the same code as the previous GET example, "To GET Data" on the previous page, then replace step 1 with:

```
...
// 1. Prepare connection
HttpPost request = new HttpPost(urlstr);
request.setEntity(requestData.getHttpEntity());
// 2. Set optional headers (e.g. to request data in JSON format)
...
```

Parsing the HTTP Response

Depending on **Accept** HTTP header, you receive back either the XML (the default) or the JSON (with application/json **accept** header) representation of the requested data. The response can be parsed using your favorite parsing library, but it is easier to use the provided SDK classes, as demonstrated in the following sections.

Topics in this section:

- "Using Jettison" below
- "Using Jackson" on page 35

Using Jettison

This example uses a call, getResourceById, placed for GroupService. According to the Javadoc, such call is performed by a method of the GroupServiceImpl class, specifically by:

```
@Path("getResourceById")
@POST
@Consumes({"application/x-www-form-urlencoded"})
public

com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceGetRe
sourceByIdResponse
```

```
    getResourceByIdPost(@DefaultValue("") @FormParam("authToken") String
authToken,
```

```
        @FormParam("resourceId") java.lang.String resourceId,
```

```
        @Context HttpHeaders hh)
```

```
    throws com.arcsight.tools.coma.service.ServiceException,
```

```
        com.arcsight.coma.bridge.AuthorizationException,
```

```
        com.arcsight.coma.bridge.AuthenticationException
```

The example shows that the method returns an instance of the class
GroupServiceGetResourceByIdResponse. It means that returned JSON/XML would be a
presentation of an object of the class GroupServiceGetResourceByIdResponse.

The information is enough for Jettison to re-create an instance of that class by the returned
JSON response. Once the instance is re-created, you can access members of that instance
without the need to parse the JSON response.

To use Jettison:

Add the ArcSight SDK libraries (utilities/sdk/lib) to your project and add code to your
client application similar to the following:

```java
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.stream.XMLStreamReader;
```

```java
import org.codehaus.jettison.json.JSONObject;
import org.codehaus.jettison.mapped.MappedNamespaceConvention;
import org.codehaus.jettison.mapped.MappedXMLStreamReader;
```

```java
// ArcSight SDK classes
import com.arcsight.product.manager.resource.service.v1.model.Group;
import
com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceGetRe
sourceByIdResponse;
```

```java
    String responseBody = ... // that's what you get back from HTTP Request
    Class claz = GroupServiceGetResourceByIdResponse.class;
```

```java
    // 1. Provide Jettison with JAXBContext
    JSONObject obj = new JSONObject(responseBody);
    JAXBContext jaxbContext = JAXBContext.newInstance(claz);
    Unmarshaller marshaller = jaxbContext.createUnmarshaller();
```

```java
    // 2. Provide information about used namespaces
    Configuration config = new Configuration();
    Map<String, String> xmlToJsonNamespaces = new HashMap<String, String>();
    //   namespaceUri =
    //
"http://ws.v1.service.resource.manager.product.arcsight.com/groupService/";
    //   namespacePrefix = "gro";
    xmlToJsonNamespaces.put(namespaceUri, namespacePrefix);
    xmlToJsonNamespaces.put("http://www.w3.org/2001/XMLSchema-instance",
```

```
"xsi");
    config.setXmlToJsonNamespaces(xmlToJsonNamespaces);
    MappedNamespaceConvention con = new MappedNamespaceConvention(config);
```

```
    // 3. Set up a reader and let it create the object of the specified class
    XMLStreamReader reader = new MappedXMLStreamReader(obj, con);
    GroupServiceGetResourceByIdResponse groupResponse
            = (GroupServiceGetResourceByIdResponse)marshaller.unmarshal
(reader);
    // 4. Retrieve a member of response class Group
    Group group = groupResponse.getReturn();
```

From here, your client application can operate on the group object directly without any JSON parsing.

> ✔ **Tip:** Download the required Jettison libraries from http://jettison.codehaus.org/.

Using Jackson

You can use two versions: Jackson 1.x from http://jettison.codehaus.org/ and Jackson 2.x from https://github.com/FasterXML/jackson.

Start by identifying SDK classes for the object returned by the service call. The example used here is the same as for Jettison, described in "Using Jettison" on page 33. It starts with adding the OpenText-provided ArcSight SDK classes:

```
// ArcSight SDK classes
```

```
import com.arcsight.product.manager.resource.service.v1.model.Group;
```

```
import
com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceGetRe
sourceByIdResponse;
```

```
    String responseBody = ... // that's what you get back from HTTP Request
    Class claz = GroupServiceGetResourceByIdResponse.class;
```

```
    ObjectMapper mapper = new ObjectMapper();
    // optional mapper configuration (see example below)
    // mapper.setPropertyNamingStrategy(new WsapiNameStrategy());
```

```
    InputStream is = new ByteArrayInputStream(responseBody.getBytes("UTF-8"));
    GroupServiceGetResourceByIdResponse groupResponse
            = (GroupServiceGetResourceByIdResponse)mapper.readValue(is,
claz);
```

```
    // Retrieve a member of response class Group
    Group group = groupResponse.getReturn()
```

Observe that the code is smaller than for Jettison. However, you might need to add mapper configuration for nonstandard cases.

Dealing with Uppercase Field Names

Field names in uppercase are used as is by Jettison (as part of Jersey) so that ESM will have the exact letter case in JSON responses.

For example, if your Java class has this member:

```
/** URI reference to this resource */
private String URI;
```

Then the returned JSON response would contain this entry:

```
"URI": "/All Cases/All Cases/Personal/admin's Cases",
```

Unfortunately, processing in Jackson would fail with this exception:

```
Caused by: org.codehaus.jackson.map.exc.UnrecognizedPropertyException:
Unrecognized field "URI" ...
...
```

This exception happens because Jackson expects field names in JSON to be in lowercase or camelStyle. There are several approaches to instruct Jackson how to map object names to JSON fields. Following is the simplest approach that creates an implementation of a class and use it with the Jackson mapper:

```
public static class WsapiNameStrategy extends PropertyNamingStrategy {

        private final Map<String, String> mapping;

public WsapiNameStrategy() {

            mapping = new HashMap<String, String>();
mapping.put("uri", "URI");
}

@Override

        public String nameForSetterMethod(MapperConfig<?> config,
AnnotatedMethod method, String defaultName) {
if (mapping.containsKey(defaultName)) {
return mapping.get(defaultName);
}
return super.nameForSetterMethod(config, method, defaultName);
}
}
```

Namespace Support

As described elsewhere in this guide, the ESM returns data in XML by default, but data can be returned in different formats specified on your requests. One available option is JSON (http://json.org/). For both formats (XML or JSON), the uses namespaces to distinguish similar-looking blocks of data that belong to resources and data that belong to services of different types.

Namespaces are a natural part of XML syntax, but there is no similar standard concept in JSON. To provide similar functionality, different JSON parsers use different solutions. Not all parsers have functionality analogous to XML namespaces.

To work with JSON, the uses jettison (http://jettison.codehaus.org/), which supports XML namespaces by using additional prefixes.

For example, the XML element

```
xmlns:ns7="http://ws.v1.service.resource.manager.product.arcsight.com/dashboar
dService/"
...
```

```
<ns7:currentSignature>
```

```
    <id>FztgORycBABDy1m0rly+irg==</id>
```

```
    <modificationCount>0</modificationCount>
```

```
</ns7:currentSignature>
```

will be represented in JSON as

```
"das.currentSignature" : {
     "id" : "FztgORycBABDy1m0rly+irg==",
     "modificationCount" : 0
}
```

Here, the prefix das is associated with XML namespace

```
xmlns:ns7="http://ws.v1.service.resource.manager.product.arcsight.com/dashboar
dService/"
```

To be able to convert XML to JSON and back, Jettison uses additional mappings between JSON namespaces (das, as in the example) and standard XML namespaces. The mappings used by Jettison from the are provided in the table in "Using Proper Namespaces" on page 15).

With that in mind, to parse the response, you need:

- the JSON presentation of the ESM resource, and
- a class corresponding to that data.

With these two pieces, a parser (Jettison or Jackson) will prepare an object of the specified class populated with the data from JSON. You can then access the parsed data by simply calling get methods on that Java object. If you use Jettison to parse a response, you do not need anything else–Jettison uses the metadata from the class definition to perform parsing. For other parsers like Jackson, you would need to provide additional classes, Marshaller and Unmarshaller, that will help Jackson properly handle namespace prefixes. This approach uses the NAMESPACE_PREFIX_MAPPER property.

Examples: Using ESM Services

This topic provides examples of consuming web services from core-service and manager-service web applications.

Topics in this section:

- "Using the LoginService from core-service" below
- "Using Services from manager-service" on page 40

See also "Using the webservices-test Web Application" on page 51 for more examples available from the test kit.

Using the LoginService from core-service

The LoginService is part of the core-service web application. This web application includes logging in and out. Any communication with ESM starts with a login request and ends with a logout request. After your session is authenticated, you get back an authentication token. As long as the session is valid, use the same token for any additional web service calls to ESM.

Topics in this section:

- "Login Request" on the next page
- "Logout Request" on the next page

Login Request

To submit a login request:

```
https://<HOST>:8443/www/core-
service/rest/LoginService/login?login=admin&password=password
```

Response

A simple response would be

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:loginResponse
    xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
    xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
    xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/">

    <ns3:return>83dFc8zbveUgdfhqtVcO6stRxDVL5v1wflAcdIls6gg.

    </ns3:return>

</ns3:loginResponse>
```

If you submit a login request with the additional **&alt=json** (because you expect the response in JSON), the expected response would be

```
{
"log.loginResponse": {
"log.return": "R89tt0pTLXGguADK7-_uHF3s68n0MINSoI2GoHC8OxY."
}
}
```

In both responses, you will get back the authentication token that you will pass as the first argument to any other web service call.

Logout Request

Web service logout calls expect the previously-received authentication token.

```
import com.arcsight.web.media.RequestData;
import com.arcsight.web.media.UrlEncodedFormData;
import com.arcsight.web.httpclients.ExecutorClient;

    HashMap<String, String> params = new HashMap<String, String>();
    params.put("authToken", authToken);
    RequestData requestData = new UrlEncodedFormData(params);
```

```
    httpClient.sendGet(protocol, hostname, port, "core-service",
        "LoginService", "logout", requestData);
```

If ESM fails to perform your request, you get an exception. If you get back the status code 204 (No Content), this means there is no response body.

Using Services from manager-service

This section provides examples for web services in manager-service and also includes additional technniques you can apply.

Prior to using the services described here, it is assumed you have the authentication token from LoginService required to log in and log out of the service. If the examples tell you to enter credentials, refer to "Using the LoginService from core-service" on page 38 for details.

Topics in this section:

CaseService

This example is a simple Java application that logs in to ESM, gets the description of a case resource with the specified ID, and then updates the case by changing its state from QUEUED to CLOSED. Find the referred classes in the ESM installation under utilities/sdk/examples/CaseExample.

```java
package main.java.com.arcsight.demo.TestClient;

import java.io.ByteArrayInputStream;
import java.util.HashMap;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
```

```java
public class TestCaseRest {
    public static void main(String[] args) throws Exception {
        ArcSightRestClient.trustAll();
```

```java
        // Specify your ESM host and credentials here
        String host = "localhost";
        String username = "PROVIDE_HERE";
        String password = "PROVIDE_HERE";
        // Specify ID of the Case Resource to be used in this example
        String caseId = "7B7wtYzEBABCAGyLqkEdj+g==";

        // Login through REST
        HashMap<String, String> params = new HashMap<String, String>();
        params.put("login", username);
        params.put("password", password);
        String xml = ArcSightRestClient.getRestXml(host, 8443,
                "core-service", "LoginService", "login", params);

        // Parse the XML for token
        DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
        DocumentBuilder b = f.newDocumentBuilder();
        Document d = b.parse(new ByteArrayInputStream(xml.getBytes()));
        Node node = d.getDocumentElement().getChildNodes().item(0);
        String token = node.getTextContent();
        if (token == null) {
            System.out.println("Failed to login");
            System.out.println(xml);
            return;
        }
        // Search case resource
        params = new HashMap<String, String>();
        params.put("authToken", token);
        params.put("caseID", caseId);
        String resXml = ArcSightRestClient.getRestXml(host, 8443,
                "manager-service", "CaseService", "getCaseEventIDs", params);

        params = new HashMap<String, String>();
        params.put("authToken", token);
        params.put("resourceId", caseId);
        // Alternatively you could use "postRestXml" method here
        String caseXml = ArcSightRestClient.getRestXml(host, 8443,
                "manager-service", "CaseService", "getResourceById", params);

        int startIndex, endIndex;
        // extract xml namespace headers
        startIndex = caseXml.indexOf("xmlns");
        endIndex = caseXml.indexOf("><ns3:return>");
        String xmlNameSpace = caseXml.substring(startIndex, endIndex);

        // extract case body
        startIndex = caseXml.indexOf("<ns3:return>") + 12;
```

```
            endIndex = caseXml.indexOf("</ns3:return>");
            String caseBodyXml = caseXml.substring(startIndex, endIndex);
            // remove <referenceString>
            startIndex = caseBodyXml.indexOf("<referenceString>");
            endIndex = caseBodyXml.indexOf("</referenceString>") + 18;
            caseBodyXml = caseBodyXml.substring(0, startIndex) +
    caseBodyXml.substring(endIndex);

            // toggle stage of QUEUED and CLOSED
            String oldStage = "<stage>QUEUED</stage>";
            String newStage = "<stage>CLOSED</stage>";
            if (caseBodyXml.indexOf(oldStage) >= 0) {
                caseBodyXml = caseBodyXml.replace(oldStage, newStage);
            } else {
                caseBodyXml = caseBodyXml.replace(newStage, oldStage);
            }

            String caseUpdateXml = "<?xml version=\"1.0\" encoding=\"UTF-8\"
    standalone=\"yes\"?>";
            caseUpdateXml += "<ns3:update " + xmlNameSpace + ">";
            caseUpdateXml += "<ns3:authToken>" + token + "</ns3:authToken>";
            caseUpdateXml += "<ns3:resource>" + caseBodyXml + "</ns3:resource>";
            caseUpdateXml += "</ns3:update>";

            params = new HashMap<String, String>();
            params.put("value", caseUpdateXml);
            String updateXml = ArcSightRestClient.postRestXml(host, 8443,
                    "manager-service", "CaseService", "update", params);

            String updatedXml = caseXml.replace("UNCLASSIFIED", "");
    }
}
```

GroupService

This example demonstrates how to add a new subgroup for storing case resources to an existing group with the specified ID. ESM SDK classes are used to prepare a complex JSON request body.

```
import java.io.StringWriter;
import java.util.HashMap;
import java.util.Map;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.stream.XMLStreamWriter;
```

```
import org.codehaus.jettison.mapped.Configuration;
import org.codehaus.jettison.mapped.MappedNamespaceConvention;
import org.codehaus.jettison.mapped.MappedXMLStreamWriter;

import com.arcsight.web.rest.CommandResult;
import com.arcsight.web.rest.RequestDataException;
import com.arcsight.web.rest.httpclients.ExecutorClient;
import com.arcsight.web.rest.httpclients.URLConnectionExecutorClient;
import com.arcsight.web.rest.media.JsonRequestData;
import com.arcsight.web.rest.media.RequestData;
```

```
public class GroupExample extends ArcSightRestClient {

...

    /**
     * Login and perform all created tests.
     * @throws Exception
     */
    public void runTests(ExecutorClient httpClient) throws Exception {

      String authToken = login(httpClient);
      try {
          int groupType = 7; // CASE
          createNewGroup(httpClient, authToken, "New Test Group for Cases",
getGroupId(), groupType);

      }  finally {
          if ( null != authToken ) {
              logout(httpClient, authToken);
          }
      }
      logMsg(this, "runTests", "COMPLETED for httpClient= " + httpClient);
  }
  /**
     * Creates a new group resource with the name <code>groupName</code> under
existing Group with ID <code>parentGroupId</code>.
     *
     * @param httpClient client to communicate with the server
     * @param authToken authentication token for the session
     * @param groupName Name of new Group resource
     * @param parentGroupId ID of the parent Group
     * @param groupType identifier of new Group type
     * @throws Exception is any error happens during the execution of the
method
     */
  private void createNewGroup(ExecutorClient httpClient, String authToken,
```

```java
String groupName,
     String parentGroupId, int groupType) throws Exception {

     final String methodName = "runTests";

     // 1. Prepare the request the form of a Java object, not JSON or XML string

     //     a. Create an object for a new Group
     com.arcsight.product.manager.resource.service.v1.model.Group testGroup
          = new com.arcsight.product.manager.resource.service.v1.model.Group
();
     testGroup.setName(groupName);
     testGroup.setContainedResourceType(groupType);

     //     b. Create and populate Java object containing request data
     com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupService
InsertResource requestObject
     = new
com.arcsight.product.manager.resource.service.v1.axis2.jaxws.GroupServiceInser
tResource();
     requestObject.setAuthToken(authToken);
     requestObject.setParentId(parentGroupId);
     requestObject.setResource(testGroup);

     // 2. Using JSON parser convert JAX-WS objects into JSON
     String namespaceUri =
"http://ws.v1.service.resource.manager.product.arcsight.com/groupService/";
     String shortNamespace = "gro";
     String requestBody = getJsonRequest(requestObject,
          namespaceUri, shortNamespace);
     debugMsg(this, methodName, "1. requestBody: " + requestBody);

     // 3. Wrap JSON into the internal object and send the request to the server
     RequestData requestData = new JsonRequestData(requestBody,
"application/json");
     CommandResult result = httpClient.sendPost(getProtocol(), getServerName
(), getPort(),
          "manager-service", "GroupService", "insertResource", requestData);
     debugMsg(this, methodName, "2. status: " + result.getStatusCode());

     // 4. Report the result or error
     if ( result.isSuccess() ) {
         debugMsg(this, methodName, "3. New Group: " + result.getResponse());
     } else {
         debugMsg(this, methodName, "3. Failure: " + result.getError());
     }

}
```

```java
  /**
   * Converts Java object into JSON presentation.
   *
   * @param requestObject Object to convert to JSON
   * @param namespaceUri namespace associated with the service
   * @param namespacePrefix JSON namespace defined for the service (see
jettison)
   * @return JSON presentation for the specified Java object
   * @throws RequestDataException
   */
  private static String getJsonRequest(final Object requestObject,
          final String namespaceUri, final String namespacePrefix) throws
RequestDataException {

      StringWriter writer = new StringWriter();
      JAXBContext jaxbContext;
      try {
          jaxbContext = JAXBContext.newInstance(requestObject.getClass());

          Configuration config = new Configuration();
          Map<String, String> xmlToJsonNamespaces = new HashMap<String,
String>(1);
          xmlToJsonNamespaces.put(namespaceUri, namespacePrefix);
          xmlToJsonNamespaces.put("http://www.w3.org/2001/XMLSchema-instance",
"xsi");
          config.setXmlToJsonNamespaces(xmlToJsonNamespaces);
          MappedNamespaceConvention con = new MappedNamespaceConvention
(config);
          XMLStreamWriter xmlStreamWriter = new MappedXMLStreamWriter(con,
writer);
          Marshaller marshaller = jaxbContext.createMarshaller();
          marshaller.marshal(requestObject, xmlStreamWriter);
          return writer.toString();
      } catch (JAXBException e) {
          throw new RequestDataException("Failed to convert requestObject to
JSON: " +  e, e);
      }
  }

  ...
}
```

ReportService

This example demonstrates a way to request the XML description of the report resource and print its value in one of the fields (URI field is used here). Other methods of ReportService

allow you to create a new report resource, delete, and modify existing resources. Note that this service does not provide the ability to run reports. Running reports are done through the ArchiveReportService, which is not supported in for ESM 6.8c or later.

```java
private void runTests(ExecutorClient httpClient, String authToken) throws
Exception {

    final String methodName = "runTests";

    // get description of existing Report
    String serviceMethodName = "getResourceById";
    HashMap<String, String> params = new HashMap<String, String>();
    params.put("authToken", authToken);
    params.put("resourceId", reportId);
    RequestData requestData = new NameValuePairRequestData(params);
    CommandResult commandResult = httpClient.sendGet(getProtocol(),
getServerName(), getPort(),
        "manager-service", "ReportService", serviceMethodName, requestData);
    String resXml = commandResult.getResponse();
    //debugMsg(this, methodName, "1.GET " + serviceMethodName + ": " +
resXml);

    int startIndex, endIndex;

    // extract Resource
    startIndex = resXml.indexOf("<ns22:return>") + 13;
    endIndex = resXml.indexOf("</ns22:return>");
    String reportResource = resXml.substring(startIndex, endIndex);
    //debugMsg(this, methodName, "2. reportResource: " + reportResource);

    // extract <URI>
    startIndex = resXml.indexOf("<URI>") + 5;
    endIndex = resXml.indexOf("</URI>");
    String uri = resXml.substring(startIndex, endIndex);
    debugMsg(this, methodName, "3. URI: " + uri);
}
```

ResourceService

In this example, you will use ResourceService to list all paths to the root resource from the resource with the given ID. You will see that each path corresponds to a different representation of the resource within the ESM framework hierarchy.

```java
private void runTests(ExecutorClient httpClient, String authToken) throws
Exception {

    final String methodName = "runTests";
```

```
    // get description of existing Resource
    String serviceMethodName = "getAllPathsToRoot";
    HashMap<String, String> params = new HashMap<String, String>();
    params.put("authToken", authToken);
    params.put("resourceId", resourceId);
    RequestData requestData = new NameValuePairRequestData(params);
        CommandResult commandResult = httpClient.sendGet(getProtocol(),
getServerName(), getPort(),
        "manager-service", "ResourceService", serviceMethodName, requestData);
    String resXml = commandResult.getResponse();
    //debugMsg(this, methodName, "1.GET " + serviceMethodName + ": " +
resXml);

    /*
    * Multiple paths to root will be returned in the form of array like below
    *
<ns23:return>01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0X
T-
k80YBABCA5fPgKSmjNg==/06tGQ60cBABCAEoIkbYC9AQ==/7iINeGUgBABCAF5RSwU2eOg==</ns2
3:return>
```

```
<ns23:return>01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0X
T-
k80YBABCA5fPgKSmjNg==/0wu10okcBABCAFs59orlj6g==/06tGQ60cBABCAEoIkbYC9AQ==/7iIN
eGUgBABCAF5RSwU2eOg==</ns23:return>
```

```
<ns23:return>01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0X
T-
k80YBABCA5fPgKSmjNg==/0wu10okcBABCAFs59orlj6g==/7iINeGUgBABCAF5RSwU2eOg==</ns2
3:return>
```

```
    */

    int startIndex=0;
    int endIndex=0;

    // Extract paths to root
    int numPaths = 0;
    while ( true ) {
        startIndex = resXml.indexOf("<ns23:return>", endIndex) + 13;

        if ( startIndex < 0 || startIndex < endIndex) {
            break;
        }
        endIndex = resXml.indexOf("</ns23:return>", startIndex);
        if ( endIndex < 0 ) {
            break;
        }
        String allPathsToRoot = resXml.substring(startIndex, endIndex);
```

```
        numPaths++;
        debugMsg(this, methodName, "" + numPaths + ". allPathsToRoot: " +
allPathsToRoot);
    }

    debugMsg(this, methodName, "Number of paths to root: " + numPaths);
}
```

Here is an example of a truncated output:

```
18:23:38,024 DEBUG ResourceExample:217 - Received Login XML = <?xml
version="1.0" encoding="UTF-8" standalone="yes"?><ns3:loginResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/"><ns3:r
eturn>znDHbIXpdtdjaq1NeKOhTFWzK7AAjSDgsgDefPYEW20.</ns3:return></ns3:loginResp
onse>
```

```
18:23:38,068 DEBUG ResourceExample:217 - Successful login
```

```
18:23:38,073 DEBUG URLConnectionExecutorClient:64 - GET
urlstr=https://HOST:8443/www/manager-
service/rest/ResourceService/getAllPathsToRoot?resourceId=7iINeGUgBABCAF5RSwU2
eOg%3D%3D&authToken=znDHbIXpdtdjaq1NeKOhTFWzK7AAjSDgsgDefPYEW20.
```

```
18:23:38,087 DEBUG ResourceExample:217 - 1. allPathsToRoot:
01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0XT-
k80YBABCA5fPgKSmjNg==/06tGQ60cBABCAEoIkbYC9AQ==/7iINeGUgBABCAF5RSwU2eOg==
```

```
18:23:38,087 DEBUG ResourceExample:217 - 2. allPathsToRoot:
01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0XT-
k80YBABCA5fPgKSmjNg==/0wu1OokcBABCAFs59orlj6g==/06tGQ60cBABCAEoIkbYC9AQ==/7iIN
eGUgBABCAF5RSwU2eOg==
```

```
18:23:38,088 DEBUG ResourceExample:217 - 3. allPathsToRoot:
01000100010001007/01000100017777777/0AFPLmPsAABCAHBFLqlR1uw==/0XT-
k80YBABCA5fPgKSmjNg==/0wu1OokcBABCAFs59orlj6g==/7iINeGUgBABCAF5RSwU2eOg==
```

```
18:23:38,088 DEBUG ResourceExample:217 - Number of paths to root: 3
```

SecurityEventService

The example starts with a login and then requests several events by ID. The returned data could be requested in XML (shown in the example) or in JSON. With regard to the requested format, the returned data could be parsed to get the values of the desired event fields.

```
package com.arcsight.tests.rest;
```

```java
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.arcsight.web.rest.CommandResult;
import com.arcsight.web.rest.httpclients.ExecutorClient;

public class SecurityEventExample extends BaseExample {

    public long[] getEventIds() {
        return new long[] { 1L, 2L, 3L };
    }

    public void test(ExecutorClient httpClient) {
        String authToken = login(httpClient);
        try {
            getSecurityEvents(httpClient, authToken, getEventIds());
        } finally {
            if ( null != authToken ) {
                logout(httpClient, authToken);
            }
        }
    }

    private void getSecurityEvents(final ExecutorClient executor, final
String authToken, final long[] eventIds)
    throws IOException {

        // See manager-client Javadoc for JSON prototype for request body
        String requestBodyFormat = "{\n"
        +  "  \"sev.getSecurityEvents\" : {\n"
        +     "     \"sev.authToken\" : \"%s\",\n"
        +     "     \"sev.ids\" : [ \"%s\" ],\n"
```

```java
        //+     "     \"sev.timeField\" : { \"JSON for the nested object\" },\n"
        +     "     \"sev.startMillis\" : \"%s\",\n"
        +     "     \"sev.endMillis\" : \"%s\"\n"
        +     "  }\n"
        +     "}";

        // make sure that your Events within this timeframe. Use "-1" for
unlimited range
        final long startMillis = -1L;
        final long endMillis = -1L;

        // simplified way to prepare request body
```

```java
        String requestBody = String.format(requestBodyFormat,
            authToken, Arrays.toString(eventIds), startMillis, endMillis);

        // Header to request response in XML format
        Map<String, String> requestProperties = new HashMap<String, String>();
        requestProperties.put("accept", "application/xml");

        // submit the request using one of the clients (HttpURLConnection or
apache HttpClient)
        String urlstr =
"https://HOST:PORT/Context/SecurityEventService/getSecurityEvents";
        CommandResult result = executor.sendPost(urlstr, requestProperties,
                requestBody, "application/json");

        // Get the description of the Events in the requested format
(XML/JSON)
        String responseBody = result.getResponse();

        // Parse the response to retrieve the fields with the required data
        /*
        <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
        <ns25:getSecurityEventsResponse

xmlns:ns2="http://ws.v1.service.resource.manager.product.arcsight.com/activeLi
stService/"
xmlns:ns3="http://ws.v1.service.resource.manager.product.arcsight.com/archiveR
eportService/"
xmlns:ns4="http://ws.v1.service.resource.manager.product.arcsight.com/caseServ
ice/"
            ...
xmlns:ns25="http://ws.v1.service.resource.manager.product.arcsight.com/securit
yEventService/"
            ...
        <ns25:return>
            <agent>
                <mutable>true</mutable>
                <address>265716344</address>
                <assetId>401Id4kUBABCAXKvda6GBuw==</assetId>
                <assetLocalId>-9223372036854775808</assetLocalId>
                <hostName>192.0.2.0</hostName>
                <macAddress>-9223372036854775808</macAddress>
                <translatedAddress>-
9223372036854775808</translatedAddress>
                <zone>
                    <externalID>Hewlett-Packard Company</externalID>
                    <id>MX8HU5fsAABCCV7v-GNArfg==</id>
                    ...
        </ns25:return>
```

```
            <ns25:return>
                <agent>
                    <mutable>true</mutable>
                    <address>265716344</address>
                    <assetId>401Id4kUBABCAXKvda6GBuw==</assetId>
                    ...
                <ttl>10</ttl>
                <type>BASE</type>
            </ns25:return>
        </ns25:getSecurityEventsResponse>
        */
    }
}
```

Using the webservices-test Web Application

The `webservices-test` web application includes examples and their usages. The web application, along with other libraries and Javadoc reference, are in

```
ARCSIGHT_HOME$/utilities/sdk/examples/TestKit/
```

Topics in this section:

- "Configuration" below
- "Using the Examples" on page 53

Configuration

Credentials to Use

All examples are executed for the following default settings:

```
{ protocol:https, servername:localhost, port:8443, username:user,
password:mypassword }
```

If you want to run the examples with different values for some of these properties, change the system property or environment variable. For example, to run an example for a different host, use the syntax

```
java -Dservername=192.0.2.0 -Dusername=testUser   -cp webservices-test-
1.7.0.release.5.jar:lib/*  com.arcsight.tests.rest.QueryViewerExample
czjbMSicBABCdSZMaQR6puQ==
```

Log Settings

The `webservices-test` web application uses standard `log4j` package to report messages. The test web application comes with default `log4j` settings described in the file, `log4j.properties`.

```
#
# Appenders.
#
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=webservices-test.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy/MM/dd HH:mm:ss} %5p %c
{1}:%L - %m%n
#
# Default configuration.
#
log4j.rootLogger=info, stdout, file
#
# ArcSight classes
#
log4j.logger.com.arcsight=debug
#
# Spring.
#
# Default log level.
log4j.logger.org.springframework=info
log4j.logger.com.sun.xml.ws=debug
log4j.logger.org.apache.axis2.jaxws.marshaller.impl.alt.MethodMarshallerUtils=
debug
```

To change log settings:

Specify the alternative location of the file with log4j configuration using the standard property, `log4j.configuration`:

```
java -Dlog4j.configuration=file:/opt/arcsight/webservices-
test/mylog4j.properties  -Dservername=192.0.2.0 -Dusername=admin    -cp
```

```
webservices-test-X.X.X.release.XXX.jar:lib/*
com.arcsight.tests.rest.QueryViewerExample czjbMSicBABCdSZMaQR6puQ==
```

Using the Examples

This section describes the required libraries and the procedures to try the examples in the test web application.

Sources and Build Artifacts

Build artifacts from `utilities/sdk/examples/TestKit` include:

Utilities	Description
`thirdparty-libs-X.X.X.release.XX.jar`	A complete set of all required third party libraries
`webservices-test-X.X.X.release.XX.jar`	Main library with ready-to-run examples
`webservices-test-javadoc-X.X.X.release.XX.jar`	Javadoc descriptions for the provided examples
`webservices-test-sources-X.X.X.release.XX.jar`	Source classes for the provided examples

Examples in `webservices-test-X.X.X.release.XX.jar` are ready to use. Try them against some ESM instances.

Running the Examples

To execute examples, you need a running ESM instance and a limited list of third-party libraries. All required third party libraries are available in `thirdparty-libs-X.X.X.release.XX.jar`. See if their versions work for you. Replace if required.

1. Go to `utilities/sdk/examples/TestKit` and extract all jar files from `thirdparty-libs-X.X.X.release.XX.jar` into a new subfolder, `lib`.

2. Check the list of available examples and their usages by running this command (use semicolon separators on Windows):

    ```
    java -cp webservices-test-X.X.X.release.XX.jar:lib/*
    com.arcsight.tests.WebServicesApi
    ```

    The resulting output is similar to the following:

    ```
    Welcome to ESM Web-Services API examples suite.
    ```

    ```
    Existing examples:
    ```

    ```
      1. CaseExample      ... REST based example for accessing Case resources
      2. QueryViewerExam ... REST based example for accessing QueryViewer
    resources
      3. LoginServiceTes ... Testing LoginService service using REST
    ```

```
   4. ActiveListServi ... Testing Active List service using REST
   5. ArchiveReportSe ... REST based example for accessing ArchiveReport
resources
   6. CaseServiceTest ... REST based example for accessing Case resources
   7. DashboardServic ... Testing Dashboard service using REST
   8. GroupServiceTes ... Testing GroupService service using REST
   9. QueryServiceTes ... REST based example for accessing ArcSightQuery
resources
   10.QueryViewerServ ... REST based example for accessing QueryViewer
resources
   11.SecurityEventSe ... Testing SecurityEventService service using REST
   12.ResourceService ... REST based example for accessing resources of the
general type
   13.UserResourceSer ... REST based example for accessing User resources
```

3. Each example has its own usage notes, like this one for UserResourceServiceTest. Follow the usage notes and run the examples for resources in your ESM instance.

```
Usage:
1. com.arcsight.tests.rest.unittests.UserResourceServiceTest:
java [OPTIONS] -cp webservices-test-1.7.0.jar:lib/*
       com.arcsight.tests.rest.unittests.UserResourceServiceTest userId
where
        userId is ID of the User Resource (ResourceType for User is
NUMBER          1)
```

```
       and OPTIONS could be used to set system properties any like
       -Dservername=<HOST>  -Dlog4j.debug
Hint: change log4j by pointing to your config as          -
Dlog4j.configuration=file:/opt/arcsight/webservices-test/1.7.0/
       log4j.properties
```

```
       ...
       ...
       ...

Options:
**************
  protocol .............. to specify protocol if it's different from "https"
  servername ............ to specify ESM hostname if it differs from
                          "localhost"
  port .................. to specify port if it's different from "8443"
  username .............. to specify ESM username (default is
                          "admin")
  password .............. to specify ESM password (default is "password")
  webservices.smoke.tests.only ... to exercise only single GET for every
                          method used in unittests
```

Publication Status

Released:

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this computer, click the link above and an email window opens with the following information in the subject line:

Feedback on ESM Service Layer (Web Services) Developer's Guide (ESM 7.6.4)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to Documentation-Feedback@microfocus.com.

We appreciate your feedback!