SAP NetWeaver Gateway Java Library

Version 2.5.400

Copyright

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Icons in Body Text

Icon	Meaning
⚠	Caution
●	Example
●	Note
⬆	Recommendation
◇	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

What's New in this Guide

Topic	Description
Using Expand System Query Option ($expand)	New Section with an example for using the $expand query option. For more information, go to page 16.
Using Batch Request ($batch)	New Section on how to implement $batch. For more information, go to page 17.

Table of Contents

Introduction

This document describes how to use the application programming interfaces (APIs) of the SAP® NetWeaver® Gateway Java library to send requests and interpret returned responses from SAP® NetWeaver® Gateway.

Documentation Types

The following is a list of the documentation types available for SAP NetWeaver Gateway Plug-in for Eclipse:

Document Type	Explanation
SAP NetWeaver Gateway Plug-in for Eclipse	This guide provides information on how to use the framework to generate code for different applications, through supported target environments, to retrieve and interact with data from your existing SAP systems. **Target group**: Application developers, project teams, and system administrators. **Current version**: SAP NetWeaver Gateway 2.0.
SAP NetWeaver Gateway Java Library	The SAP NetWeaver Gateway Java Library provides examples for using various classes and methods to call SAP services through SAP NetWeaver Gateway **Target group**: Application developers, solution consultants, and project teams for implementations. **Current version**: SAP NetWeaver Gateway 2.0.

Target Audience

The information in this guide is intended for Java developers (Eclipse environment), who want to write client applications to interact with SAP services.

This document assumes that you are familiar with:

- Open Data protocol (OData).
- Programming in Java.

For more information about the classes and methods in the SAP NetWeaver Gateway Java library, see the documentation in Javadoc format that is provided with the library.

Constraints

You can find information about the constraints applicable to SAP NetWeaver Gateway 2.0 Developer Tools in SAP Note number 1593269.

Overview

The SAP NetWeaver Gateway Java library supplies a set of APIs and classes that handle the connectivity and processing of OData for SAP solutions on top of SAP NetWeaver Gateway.

The library parses OData content and fills the appropriate objects with values.

The library provides APIs for:

- Connecting to an SAP NetWeaver Gateway system.

  It provides APIs of HTTP client for RESTful calls to an SAP NetWeaver Gateway system, supporting HTTP GET, POST, PUT and DELETE methods.

  In addition, it supports basic authentication over secure sockets layer (SSL).

- Parsing of OData elements as well as APIs for the construction of OData format strings from OData elements encapsulated in an OdataParser class.

The library simplifies sending requests to SAP NetWeaver Gateway by providing APIs for creating the request content, sending it and parsing the response into Java entities.

In addition, it supports both ATOMPub and JSON representations of OData services.

OdataParserFactory returns a specific parser implementation according to the given representation parameter (ATOM or JSON) to the *createInstance* method.

Installing SAP NetWeaver Gateway Java Library

The SAP NetWeaver Gateway Java library is available when you install the SAP NetWeaver Gateway Plug-in for Eclipse Framework.

The library is automatically imported and added to the classpath of the specified project when using the SAP NetWeaver Gateway Proxy Generation wizard.

In addition, the library is added to the classpath of a new project when using the SAP NetWeaver Gateway Starter Application wizard.

For more information, see the guide, *SAP NetWeaver Gateway Plug-In for Eclipse.*

Getting Started

The following examples show how to use various APIs in the SAP NetWeaver Gateway Java library to interact with SAP services.

In addition, SAP has provided sample code showing how to use the Java library. You can find the samples at: wiki.sdn.sap.com/wiki/display/Snippets/SAP NetWeaver Gateway-Code Snippets

To get started, you can create a custom application using the Starter Application Generation wizard of the the SAP NetWeaver Gateway plug-in for Eclipse. This adds the library into your project.

For more information, see the guide, *SAP NetWeaver Gateway Plug-In for Eclipse.*

The following is an overview for using the library:

1. First, set the parameters for connecting to an SAP NetWeaver Gateway system in which the SAP service you want to call is available.

```
IServerConnectionParameters gateway =
new ServerConnectionParameters ("<host>", "<port>","<client_id>",
<useSSL>);
```

   Note: You must replace *<host>, <port>, <client_id>*, and <useSSL> with the appropriate values.

2. Create the authentication object. In the below example, we use basic authentication.

   An example for basic authentication:

```
IAuthentication credentials = new
UsernamePasswordCredentials("<user>","<password>");
```

   Note: You must replace *<user>, <password>* with the appropriate values.

   An example for X.509 authentication:

   Note: To enable the use of X.509 authentication, you must configure the use of certificates in your SAP NetWeaver Gateway host. For more information, see the section Java SE Application Security in the guide, SAP NetWeaver Gateway Plug-In for Eclipse.

```
KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

   Configure the trust store. This is needed for SSL server validation.

```
InputStream stream = new FileInputStream("<PATH_to_cacerts>");
trustStore.load(stream, null);
```

   Load the keystore using your keystore file which contains your imported personal certificate.

```
KeyStore keystore = SSLUtil.loadKeyStore(new
FileInputStream("<YOUR_.JKS_FILE_PATH>"), "<YOUR_FILE_PASSWORD>");
SSLContext sslContext = null;
```

   Create the SSL context based on the keystores, and get the SSL socket factory.

```
sslContext = SSLUtil.createSSLContext(keystore, "password", trustStore);
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
```

Initialize the X509 authentication class with the *sslSocketFactory*.

```
X509Authentication x509Authentication = new
X509Authentication(sslSocketFactory);
```

3. Initialize the REST Client based on the Gateway connection details and the appropriate authentication.

   An example for basic authentication:

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
credentials, Representation.ATOM);
```

   An example for X509 authentication:

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
x509Authentication, Representation.ATOM);
```

   Note: The option for ATOM or JSOM representation format.
   When creating the REST Client using the *RestClientFactory,* the library automatically enables protection using cross site request forgery (CSRF) tokens.

   To disable the protection, create a new custom implementation for *RestClientAbstractFactory* and register this custom implementation as the default by calling:

```
RestClientFactory.setFactory(customFactory);
```

4. Initiate a request to obtain the response from the connected SAP NetWeaver Gateway system about the SAP service.

```
IRestRequest request = new RestRequest(url);
IRestResponse response = restClient.execute(request);
```

   Where URL is the name of the SAP service to call. For example, */sap/opu/sdata/iwfnd/RMTSAMPLEFLIGHT/FlightCollection*

5. Transform the XML response into an ODataCollection object.

```
ODataCollection collecion=
ODataParserFactory.createInstance(Representation.ATOM).parseODataCollecti
on(response.getBody());
```

6. Extract the properties from the ODataCollection object.

```
ODataEntry[] entries = collecion.getEntries();
for (ODataEntry oDataEntry : entries)
{
oDataEntry.getProperties();
}
```

For more information about the classes and methods in the SAP NetWeaver Gateway Java library, see the documentation in Javadoc format that is provided with the library.

Connecting to an SAP NetWeaver Gateway System

The first API call using the Java library is to request a connection to the specific SAP NetWeaver Gateway system in which the SAP service you want resides.

Later, you use the RestService API class of the Java library to send an HTTP request for the specific SAP service.

For example, to connect to an SAP NetWeaver Gateway host with the client ID, *EX1*. You must provide the following:

- Host name: The name of the Gateway host, for instance, *example.example.sap*
- Port number: The port number of the Gateway host, for example, *55555.*
- User authentication.

Set up connection to an SAP NetWeaver Gateway system:

```
IServerConnectionParameters gateway =
new ServerConnectionParameters ("example.example.sap","55555","EX1",
<useSSL>);
```

The *ServerConnectionParameters* class provides connectivity to the specified Gateway system and user authentication.

For more information about user authentication using X509, see *Getting Started*.

Calling an SAP Service

After setting the connection and authentication using the Java library, you use the API to form an HTTP request for the specific SAP service.

For example, to call an SAP service called, *RMTSAMPLEFLIGHT*. You must provide the URI of the service.

- Initialize the REST client based on the connection details and authentication class.

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
credentials, Representation.ATOM);
```

Or use X509 authentication. For more information about X509 authentication, see *Getting Started*.

- First create a request object, and optionally add a custom header.

```
IRestRequest request = new RestRequest(url);
request.setHeader(header, value);
}
```

- Call Gateway.

```
IRestResponse response = restClient.execute(request);
```

- Parse the string XML response into an ODataCollection object.

```
ODataCollection collection =
ODataParserFactory.createInstance(Representation.ATOM).parseODataCollectio
n(response.getBody());
```

- Iterate over the collection entries..

```
ODataEntry[] entries = collecion.getEntries();
for (ODataEntry oDataEntry : entries)
{
oDataEntry.getProperties();
}
```

To call an SAP NetWeaver Gateway service, proceed as follows:

1. Indicate to the Java library to call and use the *RestClient* class to obtain the response string from the SAP service.
2. Parse the response into Java objects.
3. Call a method to send the request and receive the results.
4. Handle errors and exceptions.

The following is an example for updating data in your SAP backend system through SAP NetWeaver Gateway:

```
/*
* Code example for updating data (PUT) through SAP NetWeaver Gateway
* @param entry entry with updated data
* @throws MarshalerException
* @throws RestClientException
*/
   private static void updateExample(ODataEntry entry) throws
   MarshalerException, RestClientException
   {
//update some data
   entry.putProperty("PASSNAME", "John Smith");
//Convert to XML
   String body =
   ODataParserFactory.createInstance(Representation.ATOM).format(entry);

// build the request
IRestRequest request = new RestRequest(url);
request.setMethod(Method.PUT);
request.setBody(body);
```

The following is an example for creating new data in your SAP backend system through SAP NetWeaver Gateway:

```
/*
* Example code for CREATING new data (POST) through SAP NetWeaver Gateway in
* the SAP backend system.
*/
private static ODataEntry createExample() throws MarshalerException,
RestClientException, ParserException
{

//Create HashMap object
HashMap<String, String> propertiesHashMap = new HashMap<String, String>();

//Add key value pairs to HashMap
  propertiesHashMap.put("carrid","AA");
  propertiesHashMap.put("connid","0017");
  propertiesHashMap.put("fldate","20120125");
  propertiesHashMap.put("CUSTOMID","00004617");
  propertiesHashMap.put("COUNTER","00000000");
  propertiesHashMap.put("AGENCYNUM","00000325");
  propertiesHashMap.put("PASSNAME","Joe Smith");

// Create an entry
ODataEntry atomEntry = createAtomEntry(propertiesHashMap);

//Convert to xml string
String body = ODataParserFactory.createInstance(Representation.ATOM).format(atomEntry);

// build the request
IRestRequest request = new RestRequest(url);
request.setMethod(Method.POST);
request.setBody(body);

//Call Gateway
IRestResponse response = client.execute(request);
String responseString = response.getBody();
System.out.println(responseString);

//Parse the response XML string from Gateway
ODataEntry newEntry =
ODataParserFactory.createInstance(Representation.ATOM).parseODataEntry(responseString);
return newEntry;

}
```

Accessing Metadata Attributes of OData Entities

For accessing metadata attributes of a feed or an entry you first need to cast the OData objects into their respective Atom specific implementations.

Below is an example, for accessing the metadata of the properties and links of some entry:

```
ODataCollection parsedODataCollection =
ODataParserFactory.createInstance(Representation.ATOM).parseODataCollection(
feed xml);

ODataEntry[] entries = parsedODataCollection.getEntries();

for (ODataEntry entry : entries)

{

       ODataProperty[] properties = entry.getProperties();

       for (ODataProperty property : properties)

       {

              ODataPropertyImpl propertyImpl = (ODataPropertyImpl)property;

              propertyImpl.getName();

              propertyImpl.getType();

              // or get attribute by name

              propertyImpl.getAttribute("m:type");

       }


ODataLink[] links = entry.getLinks();

       for (ODataLink link : links)

       {

              AtomLink atomLink = (AtomLink)link;

              atomLink.getType();

              atomLink.getSapSemantics();

              atomLink.getAttribute("key");

       }

}
```

For accessing service metadata (attributes, annotations, and so on) of the EDMX object:

```
Edmx edmx =
ODataParserFactory.createInstance(Representation.ATOM).parseEdmx(xml);

List<Property> properties = EdmxUtilities.getProperties("entityTypeName",
edmx);

for (Property property : properties)

{

       property.getSapLabel();

       property.getName();

       property.getType();

// or get attribute by name

       property.getAttribute("Nullable");

}
```

Notice the SAP annotations that appear in the EDMX document and the service document.

Moreover, once you cast to the Atom representation (AtomFeed, AtomEntry, AtomLink, or Edmx, Property, and so on), you can always take the scenic route in the XML structure.

Below is an example getting the properties of some entity types in the EDMX, you can also use:

```
edmx.getEdmxDataServices().getSchemas()
schema.getEntityTypes())
Property[] properties = entityType.getProperties();
```

Adding Custom Request and Response Interceptors

The request and response interceptors allow you to implement your own authentication intercepting implementation.

The following is sequence for implementing and using the interceptors:

- Creating the interceptor: Create an implementation of the respective interface (*IRestRequestInterceptor* or *IRestResponseInterceptor* or both).
  Add your logic in the *onRequest* or *onResponse* methods.

```
public class CustomRequestInterceptor implements
IRestRequestInterceptor

…

@Override

public void onRequest(IRestRequest restRequest, IRestClient restClient)

{

restRequest.setHeader(customKey, customValue);

}
```

- Registering the interceptor: After getting an instance of the *IRestClient* from the *RestClientFactory* register the client to use your custom interceptor.

```
restClient.registerRequestInterceptor(CustomRequestInterceptor);
```

Removing Request and Response Interceptors

You can disable certain interceptors provided with the default implementation of the *RestClientAbstractFactory*.

To do so:

- Create a new custom implementation for RestClientAbstractFactory

```
public class CustomFactory implements RestClientAbstractFactory

…

IRestClient restClient = new …

restClient.registerResponseInterceptor(…);
```

- Register this custom implementation as the default by calling:

```
RestClientFactory.setFactory(customFactory);
```

Creating Your Custom Authentication Implementation

You can use your custom authentication implementation with your application.

To do so:

- Create a new implementation of *IRestclient* that extends *RestClientBase* to get the interceptors mechanism working, and to implement *IRestclient*.
  Add handling of the custom authentication implementation.

```
public class CustomAuthentication implements IAuthentication

public class CustomRestClient extends RestClientBase implements
IRestClient

//Handle authentication in the client
```

- Create a new custom implementation for *RestClientAbstractFactory*, instantiated with the new custom implementation of *IAuthentication* and register the interceptors in the *createInstance* method.

```
public class CustomFactory implements RestClientAbstractFactory

…

IRestClient restClient = new
CustomRestClient(IServerConnectionParameters serverConnectionDetails,
IAuthentication CustomAuthentication)

restClient.registerResponseInterceptor(…);
```

- Register the custom implementation as the default by calling:

```
RestClientFactory.setFactory(CustomFactory);
```

Using Expand System Query Option ($expand)

The *$expand* system query option allows you to represent entities of an Entity Type instance or Entity Set, and include actions that are associated with the entities in the expand expression.

Below is an example code for getting the expanded entries in a navigation property:

```
IRestRequest request = new RestRequest("URL?$expand=<Navigation Property>");
IRestResponse response = client.execute(request);
```

- Example for ATOM publishing:

```
ODataEntry parseODataEntry =
ODataParserFactory.createInstance(Representation.ATOM).parseODataEntry(respo
nse.getBody());

ODataEntry expandData;

      for (ODataLink oDataLink : parseODataEntry.getLinks())
      {

       if (oDataLink.getTitle().equalsIgnoreCase("<Navigation Property>"))
       {

           InlineData inlineData = oDataLink.getInlineData();

       if (inlineData != null)
       {

           expandData = inlineData.getEntry();

        }

       }

      }
```

- Example for JSON:

```
ODataEntry parseODataEntry =
ODataParserFactory.createInstance(Representation.JSON).parseODataEntry(response.ge
tBody());

    ODataEntry expandData;

    ODataProperty property = parseODataEntry.getProperty("<Navigation Property>");

    if (property != null)

     {

      expandData = ODataEntryFactory.createEntry(Representation.JSON, "", "");
      ODataProperty[] childProperties = property.getChildProperties();

      for (ODataProperty oDataProperty : childProperties)

      {

        expandData.putProperty(oDataProperty);

     }
    }
```

Using Batch Request ($Batch)

You can create a BATCH request for a given URL suffixed with **/$batch**, that holds a list of operations.

An operation can be a read or a ChangeSet request. Requests can be added to a ChangeSet and to a Batch request given the matching API.

For a Batch request there is a matching Batch response, that holds a list of responses.

A response can be an **IRestResponse** (for a read request) or a **ChangeSetResponse** (for a ChangeSet).

Batch Request as MIME Multipart Message

The OData batch request is represented as a Multipart MIME v1.0 message, a standard format that allows the representation of multiple parts, each of which can have a different content type, within a single request.

Multipart message should include a unique boundary indicator in its header, for example, "boundary=frontier".

Each part is started with the boundary indicator, -frontier, and ends with it as well.

Part can be a retrieve request or a changeset (set of create, update, delete) operations request. So each retrieve or changeset request must be separated by the boundary indicator, as mentioned above.

Retrieve operation is regarded as a single part whereas changeset is a multipart message itself within the main multipart message.

Processing a Batch Request

The order of requests in a multipart message (retrieve or changeset) is significant and will be executed by the service accordingly.

The order of operations within changeset is not significant. A service may process them in any order.

Note: Referencing requests in a changeset, using the Content-ID is not supported SAP NetWeaver Gateway.

Batch Request Response Body

Service gets the multipart message and return 202 HTTP code when succcessful. Errors use 404 and other HTTP errors in case of malformed batch request.

Response is also a multipart message with the same order as the requests. Response for each request is the same it is outside a batch request.

Below is an example of how to use a Batch request that creates an entity, and gets the response:

```
private IRestClient client;

private ServerConnectionParameters connectionParameters;

 connectionParameters = new ServerConnectionParameters("GW HOST", "GW PORT",
"CLIENT ID", false);

client = RestClientFactory.createInstance(connectionParameters, new
UsernamePasswordCredentials("USERNAME", "PASSWORD"), Representation.ATOM);

 BatchRequest batchRequest = new
BatchRequest("/sap/opu/odata/IWFND/RMTSAMPLEFLIGHT/$batch");

IRestRequest create = new RestRequest(Method.POST,
"TravelagencyCollection");
```

```
create.setContentType("application/atom+xml");
create.setBody("some body here");
ChangeSet changeSet = new ChangeSet();
changeSet.addRequest(create);
 client.execute(tempBatchRequest);
IRestResponse response = client.execute(batchRequest);
 BatchResponse br = (BatchResponse) response;
List<IRestResponse> responses = br.getResponses();
```

Reference

The following is an example using the SAP NetWeaver Gateway Java library:

- Gateway connection details.

```
IServerConnectionParameters gateway =
new ServerConnectionParameters ("<host>", "<port>", "<client_id>",
<useSSL>);
```

- Create the authentication object.
  In the below example, we use basic authentication.

```
IAuthentication credentials = new
UsernamePasswordCredentials("<user>","<password>");
```

Note: You must replace *<user>, <password>* with the appropriate values.

An example for X.509 authentication:

Note: To enable the use of X.509 authentication, you must configure the use of certificates in your SAP NetWeaver Gateway host. For more information, see the section Java SE Application Security in the guide, SAP NetWeaver Gateway Plug-In for Eclipse.

```
KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

Import the (cacerts) certificate file from the certificate authority (CA), from your file system.

```
InputStream stream = new FileInputStream("<PATH_to_cacerts>");
trustStore.load(stream, null);
```

Load the keystore using your keystore file which contains your certificate from the file system.

```
KeyStore keystore = SSLUtil.loadKeyStore(new
FileInputStream("<YOUR_.JKS_FILE_PATH>"), "<YOUR_FILE_PASSWORD>");
SSLContext sslContext = null;
```

Create the SSL context based on the keystores, and get the SSL socket factory.

```
sslContext = SSLUtil.createSSLContext(keystore, "password", trustStore);
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
```

Initialize the X509 authentication class with the *sslSocketFactory.*

```
X509Authentication x509Authentication = new
X509Authentication(sslSocketFactory);
```

- Initialize the REST Client based on the Gateway connection details and the appropriate authentication.

An example for basic authentication:

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
credentials, Representation.ATOM);
```

An example for X509 authentication:

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
x509Authentication);
```

Note: When creating the REST Client using the *RestClientFactory,* the library automatically enables protection using cross site request forgery (CSRF) tokens.

To disable the protection, create a new custom implementation for *RestClientAbstractFactory* and register this custom implementation as the default by calling:

```
RestClientFactory.setFactory(customFactory);
```

- Initializing the REST client based on the connection details.

```
IRestClient restClient = RestClientFactory.createInstance(gateway,
credentials, Representation.ATOM);
```

Below is an example for using the header API to add custom headers to the REST Client that would be added to requests.

Note that only the last statement is the actual API call.

```
IRestRequest request = new RestRequest(url);
request.setHeader(header, value);
}
```

- Calling Gateway.

```
IRestResponse response = restClient.execute(request);
```

- Parsing the string XML response into an ODataCollection object.

```
ODataCollection collection =
ODataParserFactory.createInstance(Representation.ATOM).parseODataCollection(
Response.getBody());
```

For more information about the classes and methods in the SAP NetWeaver Gateway Java library, see the documentation in Javadoc format that is provided with the library.