Java

The Java SDK, not part of our core offering, provides an interface for the Gigya API. The library makes it simple to integrate Gigya services in your Java project. This document is a practical step-by-step guide for programmers who wish to integrate the Gigya service into their Java project. Follow the steps below to get started, and use the Library Reference while implementing.

Library Guide

Please follow these steps to integrate this library in your Java application:

Download the SDK JAR file:

If you are upgrading from a former version, please make sure to read the SDK's Change Log. You can download the binary alone, or a package including sources and JavaDoc from here: Gigya Developer Downloads

Note: The Java SDK requires JDK1.5 and above. Please note that the GSJavaSDK.jar file is compiled using JDK 1.8 with compatibility for 1.6, but you may use the SDK's source files and compile them in a JDK 1.5 environment.

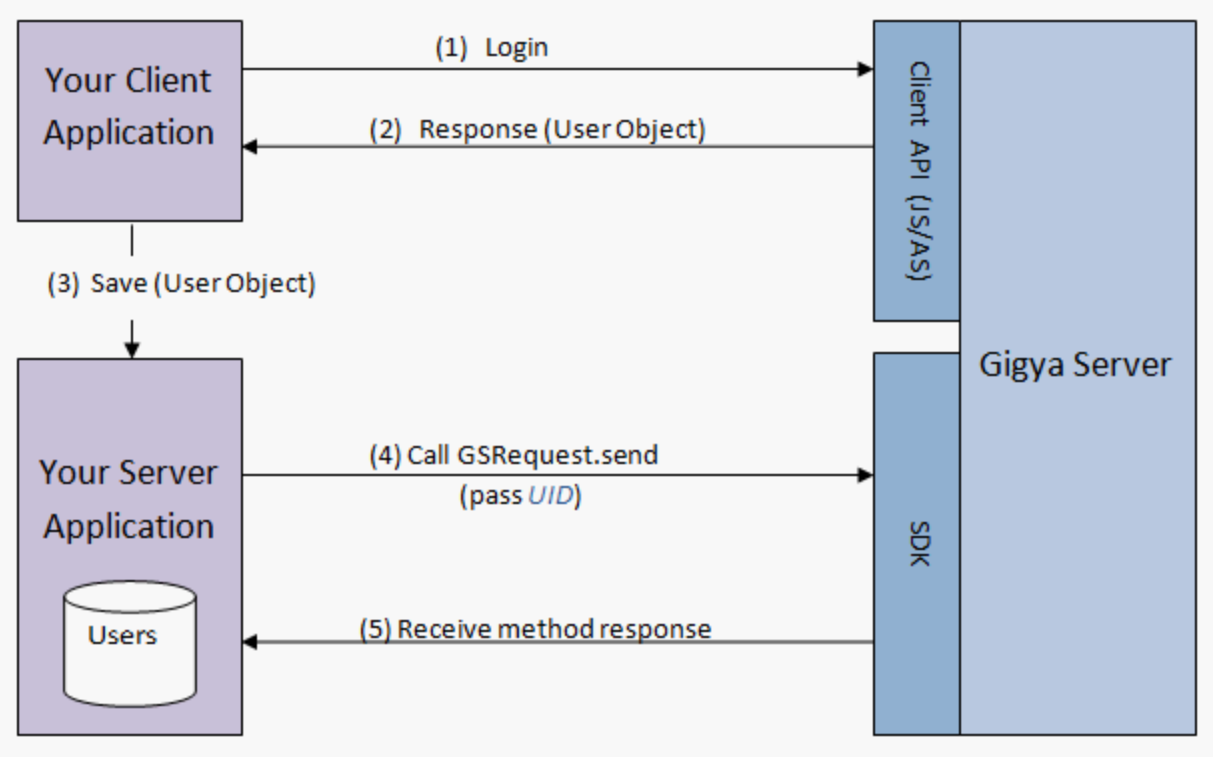
1. Please obtain Gigya's APIKey and Secret key
2. Login the User
3. Use Gigya's API - Send Requests
4. Optional - Incorporate security measures

Obtaining Gigya's API Key and Secret Key

Making API calls requires an **API Key** and a **Secret Key** that are obtained from the Dashboard section on the Gigya website. The **Secret Key** mu st be kept secret and never transmitted to an untrusted client or over insecure networks. The **API Key** and the **Secret Key** are required parameters in each request (further ahead in this document you will find guidance for sending requests).

Logging in the User

The first interaction with Gigya must always be logging in. If the user is not logged in, you cannot access their social profile nor perform social activities, such as setting their status. Sending requests requires an identified Gigya user (the identification of whom is performed using the **UID** p arameter) with an **active session**. A user session is created when a user logs in via the Gigya service. Log users in through your client application using our JavaScript Web SDK methods: socialize.login, socialize.notifyLogin, or using our ready made Login Add-on.



To learn more about the login process, see Social Login.

Sending a Request

After you have logged in the user, you may use the GSRequest class to access the user profile and perform various activities. This is implemented using GSRequest's `send` method. The following code sends a request to set the current user's status to "I feel great":

```
// Define the API-Key and Secret key (the keys can be obtained from your
site setup page on Gigya's website).
final String apiKey = "PUT-YOUR-APIKEY-HERE";
final String secretKey = "PUT-YOUR-SECRET-KEY-HERE";

// Step 1 - Defining the request
String method = "socialize.setStatus";
GSRequest request = new GSRequest(apiKey, secretKey, method);

// Step 2 - Adding parameters
request.setParam("uid", "PUT-UID-HERE");  // set the "uid" parameter to
user's ID
request.setParam("status", "I feel great");  // set the "status" parameter
to "I feel great"

// Step 3 - Sending the request
GSResponse response = request.send();

// Step 4 - handling the request's response.
if(response.getErrorCode()==0)
{   // SUCCESS! response status = OK
    System.out.println("Success in setStatus operation.");
}
else
{  // Error
    System.out.println("Got error on setStatus: " + response.getLog());
}
```

Step 1: Defining the Request

Create a GSRequest instance:

```
string method = "socialize.setStatus";
GSRequest request = new GSRequest(apiKey, secretKey, method);
```

The parameters of the GSRequest constructor are:

1. apiKey
2. secretKey
   Note: Read above about obtaining both of these keys from Gigya's site.
3. method - the Gigya API method to call, including namespace. For example: 'socialize.getUserInfo'. Please refer to the REST API reference for the list of available methods.

Step 2: Adding Parameters

After creating the GSRequest object, use the setParam method to add parameters to the request:

```
request.setParam("param1", "value1");
request.setParam("param2", "value2");
request.setParam("param3", "value3");
...
```

When a parameter is a complex object, use the GSObject class. See example in the Appendix below.

Note: In the REST API reference you may find the list of available Gigya API methods and the list of parameters per method.

Step 3: Sending the Request

Execute GSRequest 's `send` method:

```
GSResponse response = request.send();
```

The method returns a GSResponse object, which is handled in the next step.

Note: By default, requests to the Gigya API are sent using the "us1.gigya.com" domain. If your site has been set up to use another of Gigya's data centers, you must specify that the request should be sent to that specific data center and add the following line of code before calling the Send method:

```
request.setAPIDomain("<Data_Center>");
```

Where **<Data_Center>** is:

- **`us1.gigya.com`** - For the US data center.
- **`eu1.gigya.com`** - For the European data center.
- **`au1.gigya.com`** - For the Australian data center.
- **`ru1.gigya.com`** - For the Russian data center.
- **`cn1.gigya-api.cn`** - For the Chinese data center.

If you are not sure of your site's data center, see Finding Your Data Center.

See the GSRequest documentation for more information.

Step 4: Handling the Response

Use the GSResponse object to check the status of the response, and to receive response data:

```
if(response.getErrorCode()==0)
{    // SUCCESS! response status = OK
 System.out.println("Success in setStatus operation.");
}
else
{  // Error
 System.out.println("Got error on setStatus: " + response.getLog());
}
```

The GSResponse object includes data fields. For each request method, the response data fields are different. Please refer to the Gigya REST API reference for the list of response data fields per method.
For example - handling a socialize.getUserInfo response:

The response of 'socialize.getUserInfo' includes a 'user' object.

```
// Sending 'socialize.getUserInfo' request
GSRequest request = new GSRequest(apiKey, secretKey,
"socialize.getUserInfo", false);
request.setParam("uid", "PUT-UID-HERE");  // set the "uid" parameter to
user's ID
GSResponse response = request.send();

// Handle 'getUserInfo' response
if (response.getErrorCode() == 0)
{
 // SUCCESS! response status = OK
 String nickname = response.getString("nickname", "");
 int age = response.getInt("age", 0);
 System.out.println("User name: " + nickname + "; The user's age: " + age
);
}
else
{
 System.out.println("Got error on getUserInfo: " + response.getLog());
}
```

Optional - Incorporating Security Measures

Validating Signatures

> Signature validation is only necessary and supported when validating the signature of a response that was received on the client side and then passed to the server. Server-to-server calls do not contain the **UIDSignature** or **signatureTimestamp** properties in the response.

The Gigya service supports a mechanism to verify the authenticity of the Gigya processes, to prevent fraud. When Gigya sends you information about a user, your server needs to know that it is actually coming from Gigya. For this reason, Gigya attaches a cryptographic signature to the responses that include user information. We highly recommend validating the signature. The SigUtils class is a utility class for generating and validating signatures.

For example, Gigya signs the socialize.getUserInfo method response. The following code validates the signature received with the 'socialize.getUserInfo' method response:

```
// Handle 'getUserInfo' response
if (response.getErrorCode() == 0)
{
 // SUCCESS! response status = OK
 // Validate the signature
 bool valid = SigUtils.validateUserSignature(response.getString("UID", ""),
response.getString("signatureTimestamp", ""),
   secretKey, response.getString("UIDSignature", ""));

 if (valid)
  System.out.println("signature is valid");
 else
  System.out.println("Fraud!!!");
}
```

The parameters of the `validateUserSignature` method are:

1. UID - the user's unique ID
2. signatureTimestamp - The GMT time of the response in UNIX time format (i.e. the number of seconds since Jan. 1st 1970). The method validates that the timestamp is within five minutes of the current time on your server.
3. secretKey - The key to verification is your partner's "**Secret Key**". Your secret key (provided in BASE64 encoding) is located at the bottom of the Dashboard section on Gigya's website (Read more above).
4. UIDSignature - the cryptographic signature.

All the parameters, with the exception of the *secretKey*, should be taken from the 'User' object received with the 'getUserInfo' method response. The method returns a Boolean value, signifying if the signature is valid or not.

In a similar fashion, when using the 'getFriendsInfo' method, The method response include a collection of 'Friend' objects. Each Friend object will be signed with a cryptographic signature. To verify the signature of a friend object, please use the `validateFriendSignature` method.

Sending Requests over HTTPS

To send requests to the Gigya service using SSL, set the request to use HTTPS:

When creating a GSRequest object, set the *useHTTPS* Boolean parameter to be true.

```
boolean useHTTPS = true;            // send the request over HTTPS
GSRequest request = new GSRequest(apiKey, secretKey, method, useHTTPS);
```

Appendix - Publish User Action Example

The following code sample sends a request to publish a user action to the newsfeed stream on all the connected providers that support this feature.
The socialize.publishUserAction method has a complex parameter called userAction, which defines the user action data to be published. To define the userAction parameter create a GSObject object and fill it with data. There are two ways to fill the GSObject with data; you can either use the put method or construct the GSObject with a JSON string, as shown in the two examples below:

Option A - Using GSObject's put Method

```
// Defining the userAction parameter
GSObject userAction = new GSObject();
userAction.put("title", "This is my title");
userAction.put("userMessage", "This is my user message");
userAction.put("description", "This is my description");
userAction.put("linkBack", "http://google.com");

GSArray mediaItems = new GSArray();
mediaItems.add(new GSObject("{\"src\":\"http://www.f2h.co.il/logo.jpg\",
\"href\":\"http://www.f2h.co.il\",\"type\":\"image\"}"));
userAction.put("mediaItems", mediaItems);

// Sending 'socialize.publishUserAction' request
GSRequest request = new GSRequest("PUT-YOUR-APIKEY-HERE",
"PUT-YOUR-SECRET-KEY-HERE", "socialize.publishUserAction");
request.setParam("userAction", userAction); // set the "userAction"
parameter
request.setParam("uid", "PUT-UID-HERE");  // set the "uid" parameter to
user's ID

// Sending 'socialize.publishUserAction' request
GSResponse response = request.send();
```

Option B - Using a JSON String

```
// Defining the userAction parameter
GSObject userAction = new GSObject("{\"title\":\"This is my title\",
\"userMessage\":\"This is a user message\",
     \"description\":\"This is a description\",
\"linkBack\":\"http://google.com\",
     \"mediaItems\":[ {\"src\":\"http://www.f2h.co.il/logo.jpg\",
\"href\":\"http://www.f2h.co.il\",\"type\":\"image\"}]}");

// Sending 'socialize.publishUserAction' request
GSRequest request = new GSRequest("PUT-YOUR-APIKEY-HERE",
"PUT-YOUR-SECRET-KEY-HERE", "socialize.publishUserAction");
request.setParam("userAction", userAction); // set the "userAction"
parameter
request.setParam("uid", "PUT-UID-HERE");  // set the "uid" parameter to
user's ID

// Sending 'socialize.publishUserAction' request
GSResponse response = request.send();
```

To learn more about publishing user actions, see Advanced Sharing.