SETTING UP YOUR JAVA DEVELOPER ENVIRONMENT

Summary

Configure your local dev environment for integrating with Salesforce using Java.

This tipsheet describes how to set up your local environment so that you can start using Salesforce APIs, such as SOAP API or REST API.

📝 **Note:** If you're setting up a local environment to develop Salesforce applications using Apex and custom Metadata API components, take a look at the Salesforce Extensions for Visual Studio Code.

This tipsheet focuses on tools and configurations you'll need to set up your local development system. It assumes you already have a working Salesforce organization with the "API Enabled" permission. API is enabled by default on Developer Edition, Enterprise Edition, Unlimited Edition, and Performance Edition organizations.

If you are not already a member of the Lightning Platform developer community, go to `developer.salesforce.com/signup` and follow the instructions for signing up for a Developer Edition organization. Even if you already have Enterprise Edition, Unlimited Edition, or Performance Edition, use Developer Edition for developing, staging, and testing your solutions against sample data to protect your organization's live data. This is especially true for applications that insert, update, or delete data (as opposed to simply reading data).

If you have a Salesforce organization you can use for development but need to set up a sandbox for development and testing, see Deploy Enhancements from Sandboxes in Salesforce Help.

Installing Java

You need the Java Developer Kit (JDK) version 8.0 or later to use Salesforce APIs. Java is a robust, cross-platform, widely used language that integrates well with Salesforce.

To install the JDK, you need a Windows, Mac OS X, or Linux system that has internet access. Depending on your system, you might also need administrator level access to install the JDK.

📝 **Note:** If you think you already have the JDK installed, use the steps listed in **Verifying your JDK install** to verify your version of Java. Most versions of Mac OS X and Linux come pre-installed with a version of the JDK.

The JDK is a development kit required to build Java applications. The JDK includes the Java Runtime Environment (JRE) which is required to run Java applications.

1. Navigate to http://www.oracle.com/technetwork/java/javase/downloads/index.html in your browser on your local system. Download the latest version of the JDK for your operating system. Make sure you are downloading the JDK, and not the JRE.

2. On Windows, double-click the installer executable and follow the steps to install the JDK and the included JRE to your local machine. On Mac OS X, open the .dmg file and double-click the installer package. On Linux, if you downloaded an .rpm file, in a command prompt window type `rpm —ivh` *`jdk install rpm file`*. If you downloaded a .tar file, extract the files from the tar archive and copy to a location of your choice.

3. Add the JDK executables to your path.

   a. On Windows, click **Start** > **Control Panel** > **System and Security** > **System** > **Advanced system settings**. Click **Environment Variables** and find the `PATH` variable in System variables.

Add the location of the `bin` folder of the JDK installation path to the end of your path value. Your path might look something like:

`%SystemRoot%\system32;%SystemRoot%;C:\Program Files\Java\jdk1.8.0_162_x64\bin`. Click **Ok** to apply the changes.

b. On Mac OS X or Linux, you must update your `$PATH` environment variable. On Mac OS X, you can also use the `java_home` command to set your Java paths.

Verifying your JDK install

To verify your JDK install, in a command prompt window type `java —version`. You should see something like:

```
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 225.162-b12, mixed mode)
```

You can also verify that the Java compiler was properly installed by typing `javac —version` in a command prompt window. The output should look something like:

```
javac 1.80_162
```

If you get an error indicating that either java or javac is an unknown executable, your installation might have failed, or you might not have set your path environment as described in Step 3.

Installing Eclipse

Eclipse is an integrated development environment (IDE) for Java development.

Eclipse requires a Java runtime environment to run.

While Eclipse is not required to develop integration applications for Salesforce, install Eclipse if you want an easy to use IDE that works with Salesforce.

1. Navigate to http://www.eclipse.org/downloads in your browser. Download "Eclipse IDE for Java Developers." Choose either the 32–bit version or the 64–bit version, depending on the version of the JDK you have installed.

   > ⊘ Important: If you have Java versions in addition to Java 8 installed, specify a JDK 1.8 version in your `eclipse.ini` file. For details, see Specifying the JVM in the Eclipse Foundation's `eclipse.ini` documentation.

2. Un-archive the downloaded file to a location of your choice. Eclipse does not have a special installation application.

3. Launch the Eclipse executable in the `eclipse` folder you just un-archived. On Windows, this is `eclipse.exe`, on Mac OS X, this is `Eclipse.app`, and on Linux this is `eclipse`. Eclipse will ask for the location of a new eclipse workspace. Click **Ok** to accept the default workspace location.

4. Dismiss the welcome page by closing the welcome page window. You are now in the Eclipse workbench, ready to create a new Java-based Salesforce integration project.

Picking a Path Based on Which API You Use

The next steps for setting up your development environment depend on which Salesforce API you want to use.

To use SOAP API or CRUD-based Metadata API, or any other WSDL-based Salesforce API, complete the steps in the following tasks.

- Installing the Web Services Connector (WSDL-Based APIs) on page 3
- Download Developer WSDL Files (WSDL-Based APIs) on page 3
- Generating Java Stub Files (WSDL-Based APIs) on page 4
- Verify the WSDL Environment (WSDL-Based APIs) on page 4

To use REST API, Bulk API, Chatter API, or any other REST-based Salesforce API, complete the steps in the following tasks.

- Installing HttpClient and JSON Frameworks (REST-Based APIs) on page 6
- Setting Up Connected App Access (REST-Based APIs) on page 6
- Verify the REST Environment (REST-Based APIs) on page 6

Tooling API provides both SOAP and REST-based interfaces, so depending on your needs, you can set up your environment by using one of the paths above.

Streaming API requires installing additional Java frameworks for supporting push technology. See Example: Subscribe to and Replay Events Using a Java Client (EMP Connector) in the *Force.com Streaming API Developer's Guide*.

Installing the Web Services Connector (WSDL-Based APIs)

The Lightning Platform Web Services Connector (WSC) is a high-performance runtime framework that makes using WSDL-based Salesforce APIs easier.

To use the WSC framework you'll need a working install of the Java JDK.

1. Navigate to https://mvnrepository.com/artifact/com.force.api/force-wsc in your browser and download the WSC pre-built .jar file that matches the API version of Salesforce you're using. If you can't find a pre-built version of WSC that works with the API version you're using, you can build the .jar file from source. Navigate to https://github.com/forcedotcom/wsc and follow the instructions on "Building WSC."

2. Save the WSC .jar file in a known location. You'll use it to generate stub files with the WSDLs from your Salesforce organization.

3. Depending on the version of WSC you are using, you may need to also download additional frameworks. Download the following frameworks and extract and copy the framework .jar files to a location you'll remember.

   - Rhino JavaScript framework, available at https://developer.mozilla.org/en-US/docs/Rhino/Download_Rhino
   - StringTemplate engine framework, available at https://www.stringtemplate.org/download.html

Download Developer WSDL Files (WSDL-Based APIs)

Salesforce Web Services Definition Language (WSDL) files provide API details that you use in your developer environment to make API calls.

To download WSDL files directly from your Salesforce organization:

1. Log in to your Salesforce developer organization in your browser.

2. From Setup, enter *API* in the `Quick Find` box, then select **API**.

3. Download the appropriate WSDL files for the API you want to use.

   a. If you want to use SOAP API you'll need either the Enterprise or Partner WSDL. See Choosing a WSDL in the *SOAP API Developer Guide* to determine which WSDL to download.

   b. If you want to use Metadata API you'll need the Metadata WSDL. To login and authenticate with Salesforce you'll also need either the Enterprise or Partner WSDL.

   c. If you want to use Tooling API you'll need the Tooling WSDL. To login and authenticate with Salesforce you'll also need either the Enterprise or Partner WSDL.

Generating Java Stub Files (WSDL-Based APIs)

To use WSDL-based Salesforce APIs with Java, you need to generate .jar stub files that you can use in your Java projects.

You'll need the WSC .jar file to generate stub files. You'll also need the appropriate WSDL files for the API you plan to use.

1. Open a command prompt window and navigate to the location where your WSDL and WSC .jar files are.

2. Generate the Java stub for the WSDL by using the following command in a command prompt window: `java -classpath` **path to WSC jar**/**WSC jar filename** `com.sforce.ws.tools.wsdlc` **path to WSDL**/**WSDL filename path to output stub jar and filename**. You might need to also include additional .jar files that WSC needs, such as Rhino or StringTemplate, in the `classpath` list, separated by semi-colons (on Windows) or colons (on Mac/Linux). See Installing the Web Services Connector (WSDL-Based APIs) for more information on Rhino and StringTemplate.

An example Windows command for generating the stub .jar file "enterprise_stub.jar" using the API version 29.0 WSC and the Enterprise WSDL might look something like this:

```
java -classpath \testWorkspace\wsc\force-wsc-29.0.0.jar;
\testWorkspace\rhino1_7R4\js.jar;
\testWorkspace\stringTemplate\ST-4.0.7.jar;
\jdk\jdk1.7.0_17\lib\tools.jar
  com.sforce.ws.tools.wsdlc
  \testWorkspace\wsdl\enterprise.wsdl
  \testWorkspace\stub\enterprise_stub.jar
```

Note that this example includes the Rhino and StringTemplate dependent .jar files in the `classpath`.

Verify the WSDL Environment (WSDL-Based APIs)

You can verify your developer environment with a simple Java test application in Eclipse.

You should have the JDK, Eclipse, and WSC installed, and have generated the Java stub .jar files for the WSDL files that you need to use. You'll need the stub .jar file for either the Enterprise or Partner WSDL to follow the verification steps.

1. Run Eclipse. Click **File** > **New** > **Java Project** and name the project `SF-WSC-Test`.

2. Add the WSC and stub .jar files to your project. Click **Project** > **Properties** > **Java Build Path** > **Libraries**, click **Add External JARs**, select the WSC, and stub .jar files, and click **OK**.

3. Add a new folder to the `src` folder by right-clicking `src`, then select **New** > **Folder** and use `wsc` as the folder name.

4. Create a new class by right-clicking wsc and selecting **New** > **Class**. Name the class `Main`.

5. Replace the code Eclipse generates for `Main.java` as described in the following section.

Use the following simple login example code for your Main.java class. Replace `YOUR DEVORG USERNAME` with your developer organization username, and replace `YOUR DEVORG PASSWORD AND SECURITY TOKEN` with your developer organization password appended with your security token. If you did not set a security token in your organization, just provide your password. A GitHub Gist of this code is available here: https://gist.github.com/anonymous/78864d2c4ccfe4e983ef.

```java
package wsc;

import com.sforce.soap.enterprise.Connector;
import com.sforce.soap.enterprise.EnterpriseConnection;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class Main {
    static final String USERNAME = "YOUR DEVORG USERNAME";
    static final String PASSWORD = "YOUR DEVORG PASSWORD AND SECURITY
 TOKEN";
    static EnterpriseConnection connection;

    public static void main(String[] args) {

        ConnectorConfig config = new ConnectorConfig();
        config.setUsername(USERNAME);
        config.setPassword(PASSWORD);

        try {

            connection = Connector.newConnection(config);

            // display some current settings
            System.out.println("Auth EndPoint:
"+config.getAuthEndpoint());
            System.out.println("Service EndPoint:
"+config.getServiceEndpoint());
            System.out.println("Username: "+config.getUsername());
            System.out.println("SessionId: "+config.getSessionId());

        } catch (ConnectionException e1) {
            e1.printStackTrace();
        }
    }
}
```

The following example output shows a typical successful run of this code.

```
Auth EndPoint: https://login.salesforce.com/services/Soap/c/27.0
Service EndPoint:
https://yourInstance.salesforce.com/services/Soap/c/27.0/00DU0000000L5f0
```

```
Username: testuser@testorg.com
SessionId: 00DU0000000Q5f0!ARoAQDjpkH.NReBp_vBLZ124aDbgYM_v7so9ciUu
```

If the verification Java project runs and displays output that matches your organization, your developer environment is set up and you can start developing Java applications that integrate with Salesforce.

Installing HttpClient and JSON Frameworks (REST-Based APIs)

To access REST resources, you'll need to install HttpClient and JSON frameworks. HttpClient lets you access HTTP resources. The JSON framework lets you generate and parse JSON request and response data.

You'll need to have the JDK installed on your local system to use the HttpClient and JSON frameworks.

1. Navigate to http://hc.apache.org/downloads.cgi in your browser and download the binary archive of the latest "GA" version of HttpClient. Un-archive the downloaded file and move the directory to a location you'll remember.

2. Navigate to http://mvnrepository.com/artifact/org.json/json in your browser and download the latest binary .jar file. Copy this .jar file to a location you'll remember.

Setting Up Connected App Access (REST-Based APIs)

Because Salesforce REST APIs use OAuth authentication, create a connected app to integrate your application with Salesforce.

A connected app integrates an application with Salesforce using APIs. Connected apps use standard SAML and OAuth protocols to authenticate, provide single sign-on, and provide tokens for use with Salesforce APIs. In addition to standard OAuth capabilities, connected apps allow Salesforce admins to set various security policies and have explicit control over who can use the corresponding apps.

Specify basic information about your app. See Configure Basic Connected App Settings in *Salesforce Help*.

Next, provide OAuth settings. See See Enable OAuth Settings for API Integration in *Salesforce Help*.

For more information, see Create a Connected App in *Salesforce Help*.

Verify the REST Environment (REST-Based APIs)

You can verify your developer environment with a simple Java test application in Eclipse.

You should have the JDK, Eclipse, and the HttpClient and JSON frameworks installed.

1. Run Eclipse. Click **File** > **New** > **Java Project** and name the project "SF-REST-Test."

2. Click **Project** > **Properties** > **Java Build Path** > **Libraries** and click **Add External JARs**. Add the HttpClient .jar files: `httpclient`, `httpcore`, `commons-codec`, and `commons-logging` (the .jar files will have version information in the filenames). Add the JSON .jar file, which might also have a version number in the .jar filename.

3. Add a new folder to the `src` folder by right-clicking `src`, then select **New** > **Folder** and use `sfdc_rest` as the folder name.

4. Create a new class by right-clicking `sfdc_rest` and selecting **New** > **Class**. Name the class `Main`.

5. Replace the code Eclipse generates for `Main.java` as described in the following section.

Use the following simple login example code for your Main.java class. Replace YOUR DEVORG USERNAME with your developer organization username, and replace YOUR DEVORG PASSWORD + SECURITY TOKEN with your developer organization password appended with your security token. If you did not set a security token in your organization, just provide your password. Replace YOUR OAUTH CONSUMER KEY with the consumer key from your development organization's connected app. Replace YOUR OAUTH CONSUMER SECRET with the consumer secret from your development organization's connected app. A GitHub Gist of this code is available here: https://gist.github.com/anonymous/fcb1bc36ef50c0efbeb5.

```java
package sfdc_rest;

import java.io.IOException;

import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.util.EntityUtils;
import org.apache.http.client.ClientProtocolException;

import org.json.JSONObject;
import org.json.JSONTokener;
import org.json.JSONException;

public class Main {

    static final String USERNAME     = "YOUR DEVORG USERNAME";
    static final String PASSWORD     = "YOUR DEVORG PASSWORD + SECURITY
 TOKEN";
    static final String LOGINURL     = "https://login.salesforce.com";

    static final String GRANTSERVICE =
"/services/oauth2/token?grant_type=password";
    static final String CLIENTID     = "YOUR OAUTH CONSUMER KEY";
    static final String CLIENTSECRET = "YOUR OAUTH CONSUMER SECRET";

    public static void main(String[] args) {

        DefaultHttpClient httpclient = new DefaultHttpClient();

        // Assemble the login request URL
        String loginURL = LOGINURL +
                          GRANTSERVICE +
                          "&client_id=" + CLIENTID +
                          "&client_secret=" + CLIENTSECRET +
                          "&username=" + USERNAME +
                          "&password=" + PASSWORD;

        // Login requests must be POSTs
        HttpPost httpPost = new HttpPost(loginURL);
        HttpResponse response = null;

        try {
            // Execute the login POST request
```

```
            response = httpclient.execute(httpPost);
        } catch (ClientProtocolException cpException) {
            // Handle protocol exception
        } catch (IOException ioException) {
            // Handle system IO exception
        }

        // verify response is HTTP OK
        final int statusCode =
response.getStatusLine().getStatusCode();
        if (statusCode != HttpStatus.SC_OK) {
            System.out.println("Error authenticating to Force.com:
"+statusCode);
            // Error is in EntityUtils.toString(response.getEntity())

            return;
        }

        String getResult = null;
        try {
            getResult = EntityUtils.toString(response.getEntity());
        } catch (IOException ioException) {
            // Handle system IO exception
        }
        JSONObject jsonObject = null;
        String loginAccessToken = null;
        String loginInstanceUrl = null;
        try {
            jsonObject = (JSONObject) new
JSONTokener(getResult).nextValue();
            loginAccessToken = jsonObject.getString("access_token");
            loginInstanceUrl = jsonObject.getString("instance_url");
        } catch (JSONException jsonException) {
            // Handle JSON exception
        }
        System.out.println(response.getStatusLine());
        System.out.println("Successful login");
        System.out.println("  instance URL: "+loginInstanceUrl);
        System.out.println("  access token/session ID:
"+loginAccessToken);

        // release connection
        httpPost.releaseConnection();
    }
}
```

The following example output shows a typical successful run of this code.

```
HTTP/1.1 200 OK
Successful login
  instance URL: https://yourInstance.salesforce.com
  access token/session ID:
00DU0000000L5SPxa1XFi0rwB16YCQ.Xyv2nKiCT8iIN9_nkKQJ3UUf
```

8

If the verification Java project runs and displays output that matches your organization, your developer environment is now set up and you can start developing Java applications that integrate with Salesforce REST APIs.