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 Force.com IDE.

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 Force.com 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 Developer Environments in the *Application Lifecycle Guide* for instructions on creating a developer sandbox.

Installing Java

You'll need the Java Developer Kit (JDK) version 5.0 or better to use Salesforce APIs. Java is a robust, cross-platform, widely used language that integrates well with Salesforce.

To install the JDK, you'll 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 might 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 on 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\jdk1.7.0_15\bin`. Click **Ok** to apply the changes.

    b. On Mac OS X or Linux, you'll need to 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 "1.7.0_15"
Java(TM) SE Runtime Environment (build 1.7.0_15-b03)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
```

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

```
javac 1.7.0_15
```

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.

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.

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

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: Java Client in the *Force.com Streaming API Developer's Guide*.

Installing the Web Services Connector (WSDL-Based APIs)

The Force.com 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 http://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 http://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's 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, you'll need to 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.

To create a connected app, open the New Connected App wizard.

- In Salesforce Classic, from Setup, enter `Apps` in the `Quick Find` box, then select **App**. Under Connected Apps, click **New**.

- In Lightning Experience, you use the App Manager to create connected apps. From Setup, enter `App` in the `Quick Find` box, then select **App Manager**. (**1**) Click **New Connected App**. (**2**)

Next, specify basic information about your app.

1. Enter the connected app name. This name is displayed in the App Manager and on its App Launcher tile.

   📝 Note: The connected app name must be unique for the connected apps in your org. You can reuse the name of a deleted connected app if the connected app was created using the Spring '14 release or later.

2. Enter the API name used when referring to your app from a program. It defaults to a version of the name without spaces. Only letters, numbers, and underscores are allowed, so if the original app name contains any other characters, edit the default name.

3. Enter the contact email for Salesforce to use when contacting you or your support team. This address isn't given to Salesforce admins who install the app.

4. Enter the contact phone for Salesforce to use in case we need to contact you. This number isn't given to Salesforce admins who install the app.

5. Enter a logo image URL to display your logo on the App Launcher tile. It also appears on the consent page that users see when authenticating. The URL must use HTTPS. Use a GIF, JPG, or PNG file format

and a file size that's preferably under 20 MB, but, at most 100 KB. We resize the image to 128 pixels by 128 pixels, so be sure that you like how it looks. If you don't supply a logo, Salesforce generates one for you using the app's initials.

- You can upload your own logo image by clicking **Upload logo image**. Select an image from your local file system that meets the size requirements for the logo. When your upload is successful, the URL to the logo appears in the `Logo Image URL` field. Otherwise, make sure that the logo meets the size requirements.

- You can upload your own image by clicking **Upload logo image**. Select an image from your local file system. When your upload is successful, the URL to the logo appears in the `Logo Image URL` field.

- You can also select a logo from the samples provided by Salesforce by clicking **Choose one of our sample logos**. The logos include ones for Salesforce apps, third-party apps, and standards bodies. Click the logo you want, and then copy and paste the URL into the `Logo Image URL` field.

- You can use a logo hosted publicly on Salesforce servers by uploading an image as a document using the Documents tab. View the image to get the URL, and then enter the URL into the `Logo Image URL` field.

6. Enter an icon URL to display a logo on the OAuth approval page that users see when they first use your app. Use an icon that's 16 pixels high and wide and on a white background.

   You can select an icon from the samples provided by Salesforce. Click **Choose one of our sample logos**. Click the icon you want, and then copy and paste the displayed URL into the `Icon URL` field.

7. If there is a Web page with more information about your app, provide an info URL.

8. Enter a description, up to 256 characters, to display on the connected app's App Launcher tile. If you don't supply a description, just the name appears on the tile.

Next, provide OAuth settings by selecting **Enable OAuth Settings** and providing the following information.

1. Enter the callback URL (endpoint) that Salesforce calls back to your application during OAuth. It's the OAuth `redirect_uri`. Depending on which OAuth flow you use, the URL is typically the one that a user's browser is redirected to after successful authentication. Because this URL is used for some OAuth flows to pass an access token, the URL must use secure HTTPS or a custom URI scheme. If you enter multiple callback URLs, at run time Salesforce matches the callback URL value specified by the app with one of the values in `Callback URL`. It must match one of the values to pass validation.

2. If you're using the JWT OAuth flow, select **Use Digital Signatures**. If the app uses a certificate, click **Choose File** and select the certificate file.

3. Add all supported OAuth scopes to Selected OAuth Scopes. These scopes refer to permissions given by the user running the connected app. The OAuth token name is in parentheses.

   Access and manage your Chatter feed (chatter_api)
   Allows access to Chatter REST API resources only.

   Access and manage your data (api)
   Allows access to the logged-in user's account using APIs, such as REST API and Bulk API. This value also includes `chatter_api`, which allows access to Chatter REST API resources.

   Access your basic information (id, profile, email, address, phone)
   Allows access to the Identity URL service.

7

Access custom permissions (custom_permissions)
> Allows access to the custom permissions in an org associated with the connected app. It shows whether the current user has each permission enabled.

Allow access to your unique identifier (openid)
> Allows access to the logged-in user's unique identifier for OpenID Connect apps.

Full access (full)
> Allows access to the logged-in user's data, and encompasses all other scopes. `full` doesn't return a refresh token. You must explicitly request the `refresh_token` scope to get one.

Perform requests on your behalf at any time (refresh_token, offline_access)
> Allows a refresh token to be returned if the app is eligible to receive one. This scope lets the app interact with the user's data while the user is offline. The `refresh_token` scope is synonymous with `offline_access`.

Provide access to custom applications (visualforce)
> Allows access to Visualforce pages.

Provide access to your data via the Web (web)
> Allows use of the `access_token` on the web. It includes `visualforce`, which allows access to Visualforce pages.

4. Control how the OAuth request handles the ID token. If the OAuth request includes the `openid` scope, the returned token can include the ID token.

- To include the ID token in refresh token responses, select `Include ID Token`. It's always included in access token responses.

- With the primary ID token setting enabled, configure the secondary settings that control the ID token contents in both access and refresh token responses. Select at least one of these settings.

  Include Standard Claims
  > Include the standard claims that contain information about the user, such as the user's name, profile, phone_number, and address. The OpenID Connect specifications define a set of standard claims to be returned in the ID token.

  Include Custom Attributes
  > If your app has specified custom attributes, include them in the ID token.

  Include Custom Permissions
  > If your app has specified custom permissions, include them in the ID token.

When you've finished entering the information, click **Save** to save your new app. You can now publish your app, make further edits, or delete it. If you're using OAuth, saving your app gives you two new values that the app uses to communicate with Salesforce.

- `Consumer Key`: A value used by the consumer to identify itself to Salesforce. Referred to as `client_id` in OAuth 2.0.

- `Consumer Secret`: A secret used by the consumer to establish ownership of the consumer key. Referred to as `client_secret` in OAuth 2.0.

See Create a Connected App in the Salesforce online help for more for more information on connected apps.

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) {
```

9

```java
            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
```

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.