




SETTING UP YOUR .NET DEVELOPER ENVIRONMENT

Summary

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

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.

To create a Developer Edition org, go to developer.salesforce.com/signup and follow the instructions for signing up for a Developer Edition organization.

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 Microsoft Visual Studio

You'll need to install Microsoft Visual Studio to use Salesforce APIs. Visual Studio is a development environment that enables you to create robust .NET applications.

To install Microsoft Visual Studio, you'll need a Windows system with Internet access. Depending on your system, you might also need administrator-level access to install Visual Studio.

Microsoft Visual Studio provides the necessary development tools and SDKs that are required to build Windows applications. There are several versions and editions of Visual Studio, each with different features and different Windows platform requirements. See <http://www.visualstudio.com/products/compare-visual-studio-products-vs> to compare current editions. For this tip sheet, we use Visual Studio Express 2013 for Windows on a Windows 7 system.

1. Navigate to <http://www.visualstudio.com/downloads/download-visual-studio-vs> in your browser.
2. Follow the instructions to download the version of Visual Studio that fits your development needs.
3. Double-click the installer executable and follow the steps to install Visual Studio.

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

- [Download Developer WSDL Files \(WSDL-Based APIs\)](#) on page 2
- [Verify the WSDL Environment \(WSDL-Based APIs\)](#) on page 2

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

- [Setting Up Connected App Access \(REST-Based APIs\)](#) on page 5
- [Verify the REST Environment \(REST-Based APIs\)](#) on page 5

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

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:

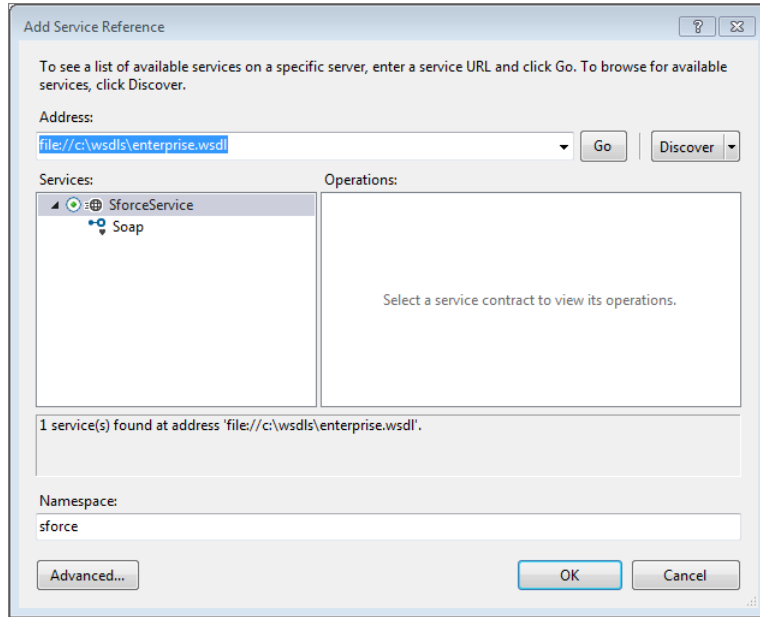
1. Log in to your Salesforce developer organization in your browser.
2. From Setup, enter `API` in the `Quick Find` box, then select **API**.
3. Download the appropriate WSDL files for the API you want to use.
 - a. If you want to use SOAP API you'll need either the Enterprise or Partner WSDL. See [Choosing 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.

Verify the WSDL Environment (WSDL-Based APIs)

You can verify your developer environment with a simple C# test application in Visual Studio.

You should have Visual Studio installed and have the WSDL files that you need to use available. You'll need the Enterprise or Partner WSDL to follow the verification steps.

1. Start Visual Studio.
2. Click **File > New Project**.
3. In the New Project window, navigate to **Installed > Templates > Visual C# > Windows**. Select **Console Application**.
4. In the New Project window, under **Name**, name the project `VerifyWSDLTest`. Under **Solution name**, name the solution `VerifyWSDLTest`. Under **Location**, pick a file location that you'll remember. Click **Ok**. A solution is created, and `Program.cs` is opened for editing.
5. Click **Project > Add Service Reference**.
6. In the Add Service Reference window, under **Address**, enter the file URL path to your Enterprise or Partner WSDL file. For example, if `enterprise.wsdl` is saved in `c:\wsdls`, enter `file:///c:\wsdls\enterprise.wsdl`. Click **Go**. The Services list is populated with an entry that is named `SforceService`.
7. Select `SforceService`. Under **Namespace** enter `sforce`. Your Add Service Reference window should look something like the following image.



Click **Ok**. The WSDL services are imported under the `sforce` namespace in your project.

8. Replace the code in `Program.cs` as described in the following section.

Use the following simple login example code for your `Program.cs` contents. 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/7d533cf2b4822fb29317>.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using VerifyWSDLTest.sforce;

namespace VerifyWSDLTest
{
    class Program
    {

        private static SoapClient client;
        private static LoginResult loginResult;

        private static bool login()
        {

            client = new SoapClient();

            string acctName = "YOUR DEVORG USERNAME";
            string acctPw = "YOUR DEVORG PASSWORD AND SECURITY TOKEN";
```

```

        try
        {
            loginResult = client.login(null, acctName, acctPw);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unexpected login error: " +
e.Message);
            Console.WriteLine(e.StackTrace);
            return false;
        }
        return true; // success
    }

    static void Main(string[] args)
    {
        if (login())
        {
            // display some current login settings
            Console.Write("Service endpoint: " +
loginResult.serverUrl + "\n");
            Console.Write("Username: " +
loginResult.userInfo.userName + "\n");
            Console.Write("SessionId: " + loginResult.sessionId +
"\n");

            Console.Write("Press any key to continue:\n");
            Console.ReadKey();
        }
    }
}

```

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

```

Service endpoint:
https://yourInstance.salesforce.com/services/Soap/c/29.0/00DU0000000T5f0/0DFU00000008XpB
Username: myUser@my_devedition.com
SessionId:
00DU0000000L5f0!ARoAQOYUBIRHn9tDOCF...zESyoJ2JwHN2ovyY2vIP1TGjc8vn_C
Press any key to continue:

```

If the verification project runs and displays output that matches your organization, your developer environment is set up and you can start developing .NET applications that integrate with Salesforce. If you have additional WSDL files you can add them to your project via the Add Service Reference dialog as described in steps 5 and 6 above.

If you are using Microsoft Visual Studio 2010 or earlier, in step 5, Add Service Reference might be unavailable. Use Add Web Reference instead. Under **URL**, enter the file path to your WSDL, such as `file://c:\wsdls\enterprise.wsdl`. Name the web reference `sforce`, and click **Add Reference**. If you have to use a web reference, use an instance of `SforceService` to call `SforceService.login()` instead of `SoapClient.login()`.

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 [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 C# test application in Visual Studio.

You must have Visual Studio installed to create a C# test application.

1. Start Visual Studio.
2. Click **File > New Project**.
3. In the New Project window, navigate to **Installed > Templates > Visual C# > Windows**. Select **Console Application**.
4. In the New Project window, under **Name**, name the project `VerifyRESTTest`. Under **Solution name**, name the solution `VerifyRESTTest`. Under **Location**, pick a file location that you'll remember. Click **Ok**. A solution is created, and `Program.cs` is opened for editing.
5. Click **Project > Add Reference**.
6. In the Reference Manager window, navigate to **Assemblies > Framework**. Select **System.Runtime.Serialization**. Ensure that the checkbox is selected, and click **Ok**.
7. Replace the code in `Program.cs` as described in the following section.

Use the following simple login example code for your `Program.cs` file. 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/db367194e6e24faa081b>.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;

namespace VerifyRESTTest
```

```

{
    class Program
    {
        // Class used for serializing the OAuth JSON response
        [DataContract]
        public class OAuthUsernamePasswordResponse
        {
            [DataMember]
            public string access_token { get; set; }
            [DataMember]
            public string id { get; set; }
            [DataMember]
            public string instance_url { get; set; }
            [DataMember]
            public string issued_at { get; set; }
            [DataMember]
            public string signature { get; set; }
        }

        private static string accessToken = "";
        private static string instanceUrl = "";

        private static void login()
        {
            string acctName = "YOUR DEVORG USERNAME";
            string acctPw = "YOUR DEVORG PASSWORD AND SECURITY TOKEN";

            string consumerKey = "YOUR OAUTH CONSUMER KEY";
            string consumerSecret = "YOUR OAUTH CONSUMER SECRET";

            // Just for testing the developer environment, we use the
            simple username-password OAuth flow.
            // In production environments, make sure to use a stronger
            OAuth flow, such as User-Agent
            string strContent = "grant_type=password" +
                "&client_id=" + consumerKey +
                "&client_secret=" + consumerSecret +
                "&username=" + acctName +
                "&password=" + acctPw;

            string urlStr =
                "https://login.salesforce.com/services/oauth2/token?" + strContent;

            HttpRequest request = WebRequest.Create(urlStr) as
            HttpRequest;
            request.Method = "POST";

            try
            {
                using (HttpWebResponse response = request.GetResponse()
                as HttpWebResponse)
                {
                    if (response.StatusCode != HttpStatusCode.OK)
                        throw new Exception(String.Format(

```

```

        "Server error (HTTP {0}: {1}).",
        response.StatusCode,
        response.StatusDescription));

        // Parse the JSON response and extract the access
        token and instance URL
        DataContractJsonSerializer jsonSerializer = new
DataContractJsonSerializer(typeof(OAuthUsernamePasswordResponse));
        OAuthUsernamePasswordResponse objResponse =
jsonSerializer.ReadObject(response.GetResponseStream()) as
OAuthUsernamePasswordResponse;
        accessToken = objResponse.access_token;
        instanceUrl = objResponse.instance_url;
    }
}
catch (Exception e)
{
    Console.WriteLine("\nException Caught!");
    Console.WriteLine("Message :{0} ", e.Message);
}
}

static void Main(string[] args)
{
    login();
    if (accessToken != "")
    {
        // display some current login settings
        Console.Write("Instance URL: " + instanceUrl + "\n");

        Console.Write("Access Token: " + accessToken + "\n");

        Console.Write("Press any key to continue:\n");
        Console.ReadKey();
    }
}
}
}

```

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

```

Instance URL: https://yourInstance.salesforce.com
Access Token:
00DU0000000L3f0!ARoAQK14rimMZ7kh11QpvHv...VreCvYJafmsCd2MgBM1UltPflQ
Press any key to continue:

```

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