AJAX Toolkit Developer Guide

Version 63.0, Spring ’25

Last updated: February 28, 2025

CONTENTS

CHAPTER 1 AJAX Toolkit Developer Guide

Embed API calls in Visualforce pages, buttons, and links with AJAX Toolkit — a JavaScript wrapper around SOAP API.

The AJAX Toolkit supports Microsoft® Internet Explorer® versions 9, 10, and 11 with the latest Microsoft hot fixes applied, and Mozilla® Firefox®, most recent stable version. The AJAX Toolkit is based on the partner WSDL. Because there's no type checking in JavaScript, the type information available in the enterprise WSDL isn't needed.

Note: Before you use the AJAX Toolkit, familiarize yourself with JavaScript and with the information about SOAP API in the *SOAP API Developer Guide*.

When to Use the AJAX Toolkit

For best performance, use the AJAX Toolkit when working with small amounts of data.

Because information is delivered via a browser, AJAX works best with relatively small amounts of data (up to 200 records, approximately six fields with 50 characters of data each). The larger the data set returned, the more time it will take to construct and deconstruct a SOAP message, and as the size of an individual record gets larger, the impact on performance becomes greater. Also, as more HTML nodes are created from the data, the potential for poor performance increases. Because browsers are not efficient, careful consideration needs to be given to browser memory management if you intend to display a large amount of data.

The following are examples of appropriate uses:

- Display or modify a single record.
- Display two or three fields from many records.
- Perform one or more simple calculations, then update a record.

The following are examples of scenarios that require case-by-case analysis:

- Update more than 200 records.
- Update records that are unusually large. For example, what happens if the user clicks the browser stop button?
- Recalculate a complex value for more than 200 records.

An example of inappropriate usage is providing a sortable grid of many records. This would require too much processing time, and browser rendering would be too slow.

AJAX Toolkit Support Policy

Other Resources

AJAX Typographical Conventions

Sample Visualforce Page Using the AJAX Toolkit
This example demonstrates using the AJAX Toolkit in a Visualforce page.

AJAX Toolkit Support Policy

The current release of the AJAX Toolkit is the only version that receives bug fixes and enhancements. When a new version is released, the previous version continues to be available, but is not supported.

Other Resources

In addition to the content of this document, there are other resources available for you as you learn to use the AJAX Toolkit:

- IDE: Salesforce extensions for Visual Studio Code
- Message boards: Salesforce Developers

AJAX Typographical Conventions

Topics about the AJAX Toolkit use the following typographical conventions:

Convention	Description
`<script src="/soap/ajax/63.0/connection.js" type="text/javascript"></script>`	In an example, Courier font indicates items that you should type the information as shown. This includes sample code, literals, methods, calls, functions, and events from a variety of languages.
`sforce.connection.`***header_option_name***`="`***value***`";`	In an example or syntax statement, italics represent variables. You supply the actual value.

Sample Visualforce Page Using the AJAX Toolkit

This example demonstrates using the AJAX Toolkit in a Visualforce page.

To add JavaScript to a Visualforce page, use this procedure:

1. Create the Visualforce page. For more information, see the *Visualforce Developer's Guide*.
2. Cut and paste the following sample code into your Visualforce page.

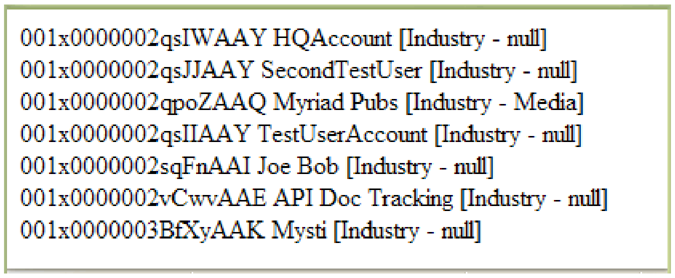
   The JavaScript code queries your organization and returns every account ID, account name, and industry type, if any:

```
<apex:page >
    <script type="text/javascript">
    var __sfdcSessionId = '{!GETSESSIONID()}';
```

```
  </script>
  <script src="../../soap/ajax/63.0/connection.js"
        type="text/javascript"></script>
  <script type="text/javascript">     window.onload = setupPage;
  function setupPage() {
    //function contains all code to execute after page is rendered

    var state = { //state that you need when the callback is called
        output : document.getElementById("output"),
        startTime : new Date().getTime()};

    var callback = {
        //call layoutResult if the request is successful
        onSuccess: layoutResults,

        //call queryFailed if the api request fails
        onFailure: queryFailed,
        source: state};

    sforce.connection.query(
        "Select Id, Name, Industry From Account order by Industry",
         callback);
}

function queryFailed(error, source) {
  source.output.innerHTML = "An error has occurred: " + error;
}

/**
* This method will be called when the toolkit receives a successful
* response from the server.
* @queryResult - result that server returned
* @source - state passed into the query method call.
*/
function layoutResults(queryResult, source) {
  if (queryResult.size > 0) {
    var output = "";

    //get the records array
    var records = queryResult.getArray('records');

    //loop through the records and construct html string
    for (var i = 0; i < records.length; i++) {
      var account = records[i];

      output += account.Id + " " + account.Name +
          " [Industry - " + account.Industry + "]<br>";
    }

  //render the generated html string
  source.output.innerHTML = output;
  }
}
</script>
```

```
    <div id="output"> </div>

</apex:page>
```

After you created and navigated to the Visualforce page, you see text similar to this image:



```
001x0000002qsIWAAY HQAccount [Industry - null]
001x0000002qsJJAAY SecondTestUser [Industry - null]
001x0000002qpoZAAQ Myriad Pubs [Industry - Media]
001x0000002qsIIAAY TestUserAccount [Industry - null]
001x0000002sqFnAAI Joe Bob [Industry - null]
001x0000002vCwvAAE API Doc Tracking [Industry - null]
001x0000003BfXyAAK Mysti [Industry - null]
```

Note: You can also use an Apex controller to create the Visualforce page. However, this sample is about basic functionality with the AJAX Toolkit that contains API calls and processes Salesforce data.

Working with the AJAX Toolkit

Most JavaScript that you add to Visualforce pages, buttons, or links has three sections: first, connecting to the AJAX Toolkit, next, embedding the API methods in JavaScript, and finally, processing the results. This section explains each of these steps.

Connecting to the API
The first portion of any JavaScript code that uses the AJAX Toolkit must make the toolkit available to the JavaScript code. The syntax for this is different depending on whether you are embedding JavaScript in a Visualforce page, or a button or link.

Embedding API Calls in JavaScript

Processing Results

Connecting to the API

The first portion of any JavaScript code that uses the AJAX Toolkit must make the toolkit available to the JavaScript code. The syntax for this is different depending on whether you are embedding JavaScript in a Visualforce page, or a button or link.

Note: API calls using the AJAX Toolkit always use the latest version of installed packages. If you're working with Visualforce components, keep in mind that you cannot access deleted components from previous package versions using the API.

- For Visualforce pages or any source other than a custom `onclick` JavaScript button, specify a `<script>` tag that points to the toolkit file:

```
<apex:page>
          <script type="text/javascript"
src="../../soap/ajax/63.0/connection.js"></script>
          <script type="text/javascript">
               sforce.connection.sessionId='{!GETSESSIONID()}';
          ...
          </script>
</apex:page>
```

- For Visualforce pages where `<apex:page showHeader="false">`, you must first set `var __sfdcSiteUrlPrefix = '{!$Site.Prefix}'` before you load the toolkit file:

  ```
  <apex:page showHeader="false">
            <script type="text/javascript">
                 var __sfdcSiteUrlPrefix = '{!$Site.Prefix}';
            </script>
            <script type="text/javascript"
  src="{!$Site.Prefix}/soap/ajax/63.0/connection.js"></script>
            <script type="text/javascript">
                 sforce.connection.sessionId='{!GETSESSIONID()}';
            ...
            </script>
  </apex:page>
  ```

  Alternatively, you can set the site path directly on the `UserContext` Javascript object with `UserContext.siteUrlPrefix = '{!$Site.Prefix}';`.

- For a custom `onclick` JavaScript button, use `!requireScript` to point to the toolkit file:

  ```
   <body>
       {!requireScript("/soap/ajax/63.0/connection.js")}
       ...
  ```

  The AJAX Toolkit picks up the endpoint and manages the session ID. You do not need to set them.

The version of the AJAX Toolkit is in the URL.

After this script executes, the toolkit is loaded and a global object, `sforce.connection`, is created. This object contains all of the API calls and AJAX Toolkit methods, and manages the session ID. No other session management is needed.

Salesforce checks the IP address from which the client app is logging in and blocks logins from unknown IP addresses. For a blocked login via the API, Salesforce returns a login fault. Then, the user must add their security token to the end of their password in order to log in. A security token is an automatically-generated key from Salesforce. For example, if a user's password is *`mypassword`*, and their security token is *`XXXXXXXXXX`*, then the user must enter *`mypasswordXXXXXXXXXX`* to log in. Users can obtain their security token by changing their password or resetting their security token via the Salesforce user interface. When a user changes their password or resets their security token, Salesforce sends a new security token to the email address on the user's Salesforce record. The security token is valid until a user resets their security token, changes their password, or has their password reset. When the security token is invalid, the user must repeat the login process to log in. To avoid this, the administrator can make sure the client's IP address is added to the organization's list of trusted IP addresses. For more information, see "Security Token" in the in the SOAP API Developer Guide.

Tip: We recommend that you obtain your security token via the Salesforce user interface from a trusted network prior to attempting to access Salesforce from a new location.

If single sign-on (SSO) is enabled, users who access the API or a desktop client can't log in unless their IP address is included on your org's list of trusted IP addresses or on their profile, if their profile has IP address restrictions set. The delegated authentication authority usually handles login lockout policies for users with the Uses Single Sign-On permission. However, if the security token is enabled, your login lockout settings determine how many times a user can try to log in with an invalid security token before getting locked out. For more information, see Setting Login Restrictions and Setting Password Policies in Salesforce Help.

Embedding API Calls in JavaScript

After you have made the toolkit available using the procedure in Connecting to the API, you can write the JavaScript code that contains your API calls and processing. Be sure to check the *SOAP API Developer Guide* for information about each call that you wish to use. The syntax for calls is different in the AJAX Toolkit; for details see API Calls and the AJAX Toolkit.

The following example shows a simple synchronized call that you can issue after connecting. This query returns the `Name` and `Id` for every `User` and writes them to the log.

```
  result = sforce.connection.query("Select Name, Id from User");
  records = result.getArray("records");

  for (var i=0; i< records.length; i++) {
    var record = records[i];
    log(record.Name + " -- " + record.Id);
  }
```

We recommend that you wrap your JavaScript code so that the entire HTML page is rendered by the browser before the code executes, to avoid errors. For example:

```
<body onload="setupPage();">
     <div id="output"></div>
</body>
```

When you specify `setupPage()` in the `body onload`, the browser initializes all HTML elements before it calls `setupPage()`.

For example, the following code could be added to a Visualforce page to retrieve data:

```
<script type="text/javascript">
    function setupPage() {
           sforce.connection.query("Select Id, Name, Industry From Account order by
Industry",
           {onSuccess : layoutResults,
            onFailure : queryFailed,
            source : {
                      output : document.getElementById("output"),
                      startTime : new Date().getTime()
                     }
           });
        }
</script>
```

The API interaction in the code above is accomplished in the first line of the `setupPage` function. A SOQL statement specifies what data to return. For more information about the `source` context variable, see source Context Variable.

After fetching the data in this example, you should handle error conditions, for example:

```
       function queryFailed(error, source) {
       source.output.innerHTML = "<font color "red">
                 An error has occurred: </font> <p>" + error;
       }
```

For more about error handling, see Error Handling with the AJAX Toolkit.

Use a callback function to handle the results of this asynchronous call. A callback function is a function that is passed by reference to the AJAX Toolkit. The AJAX Toolkit calls the callback function under defined conditions, for example, upon completion. For more information about callback function syntax, see API Calls and the AJAX Toolkit.

For example, the following code verifies that at least one result was returned, and iterates through the result set if it exists:

```
/**
* This method will be called when the toolkit receives a successful
* response from the server.
* @queryResult - result that server returned
* @source - state passed into the query method call.
*/

function layoutResults(queryResult, source) {

     if (queryResult.size > 0) {
     var output = "";

     //get the records array
     var records = queryResult.getArray('records');

     //loop through the records and construct html string
     for (var i = 0; i < records.length; i++) {
          var account = records[i];
          output += account.Id + " " + account.Name +
          " [Industry - " + account.Industry + "]<br>";
     }

     //render the generated html string
     source.output.innerHTML = output;
     }
}
```

A suggested best practice is to use JavaScript `onFailure` as the callback function for failure conditions and JavaScript `onSuccess` for processing results that are successfully returned.

For more information about embedding API calls in JavaScript with the AJAX Toolkit, especially the differences in syntax and availability of asynchronous calls, see API Calls and the AJAX Toolkit.

Processing Results

You can process the results of a query that returns enough rows to require `queryMore` and `queryLocator`, much as you do now, iterating across the results:

```
    var result = sforce.connection.query("select name, id from account");
    var queryMore = true;
    while (queryMore) {
        var records = result.getArray("records");
        for (var i = 0; i < records.length; i++) {
            //process records[i]
        }
        if (result.getBoolean("done")) {
            queryMore = false;
        } else {
            result = sforce.connection.queryMore(result.queryLocator);
        }
    }
```

However, the AJAX Toolkit provides the `QueryResultIterator` object so that you can easily iterate through results without invoking `queryMore` and `queryLocator`. If you are experienced with the API and JavaScript, see QueryResultIterator.

For other calls, you must handle the batching of up to 200 records at a time yourself. For example, the following sample shows how to batch files for a `create()` call:

```
var accounts = [];

for (var i=0; i<10; i++) {
    var account = new sforce.SObject("Account");
    account.Name = "my new account " + i;
    accounts.push(account);
    }

var result = sforce.connection.create(accounts);

var sb = "";

for (var i=0; i<result.length; i++) {
    if (result[i].getBoolean("success")) {
    sb += "\n new account created with id " + result[i].id;
    } else {
    sb += "\n failed to create account " + result[i];
    }
}

alert("Result : " + sb);
```

For more examples, see Examples of Synchronous Calls.

API Calls and the AJAX Toolkit

This toolkit supports all SOAP API calls, as well as `runTests()` from Apex.

Synchronous and Asynchronous Calls with the AJAX Toolkit
The AJAX Toolkit supports both synchronous and asynchronous calls.

Object Functions
Property values can be accessed directly or by using a generic `set` or `get` method.

Data Types in AJAX Toolkit
The AJAX Toolkit returns all data as strings. If needed, you can convert the data into an appropriate datatype by using one of the functions supplied with the returned object.

source Context Variable
Pass in any context and get it back in the callback method by using the `source` context variable.

Debugging with the AJAX Toolkit
The AJAX Toolkit provides a debugging window that pops up when certain errors are encountered.

Example Calls Using the Ajax Toolkit

Synchronous and Asynchronous Calls with the AJAX Toolkit

The AJAX Toolkit supports both synchronous and asynchronous calls.

Asynchronous calls allow the client side process to continue while waiting for a call back from the server. To issue an asynchronous call, you must include an additional parameter with the API call, referred to as a callback function. Once the result is ready, the server invokes the callback method with the result.

API Call Syntax in the AJAX Toolkit

SOAP API calls use slightly different syntax in AJAX Toolkit, depending on whether the call is synchronous or asynchronous.

Synchronous Calls

Syntax:

```
sforce.connection.method("arg1","arg2", ...);
```

Example:

```
sforce.connection.login("MyName@MyOrg.com","myPassword1");
```

Asynchronous Calls

Syntax:

```
method("arg1","arg2", ..., callback_method);
```

Example:

```
var callback = {onSuccess: handleSuccess, onFailure: handleFailure};
function handleSuccess(result) {}
function handleFailure(error) {}
sforce.connection.query("Select name from Account", callback);
```

In this example, `onSuccess` is the callback function, which returns the results when they are ready.

See Core Calls in the *SOAP API Developer Guide* for call usage, arguments, and best practices, but use the AJAX Toolkit syntax for methods you embed in JavaScript.

Note: Because `delete` is a JavaScript keyword, use `deleteIds` instead of the API call `delete`.

Object Functions

Property values can be accessed directly or by using a generic `set` or `get` method.

- A `get` function for each field in the object. For example, an Account object has a `get("Name")` function. This can be used instead of ***`object.Field`*** (for example, `account.Name`).
- A `set` function for each field in the object. For example, an Account object has a `set("Name)` function. This can be used instead of ***`object.Field`*** `=` ***`value`***.

Examples

For example, you can get the value of the `Name` field from an Account using either of these methods:

- `account.get("Name")`
- `account.Name`
- `account["Name"]`

You can set the value of the `Name` field from an Account using either of these methods:

- `account.set("Name", "MyAccount");`
- `account.Name = "MyAccount";`
- `account["Name"]="MyAccount";`

SEE ALSO:

Processing Results

Data Types in AJAX Toolkit

The AJAX Toolkit returns all data as strings. If needed, you can convert the data into an appropriate datatype by using one of the functions supplied with the returned object.

- `getDate` maps dates to JavaScript Date.
- `getDateTime` maps dateTime values to JavaScript Date.
- `getInt` maps integer values to JavaScript Int.
- `getFloat` maps float values to JavaScript Float.
- `getBoolean` maps boolean values to JavaScript Boolean.
- `getArray` retrieves arrays of values.
- `getBase64Binary` returns the decoded value of a Base64 binary encoded string. This is typically used for working with documents and attachments. See Working with Base64 Binary Encoded Strings for more information.

If you request a field whose value is null in a query, the returned value will be null. If you do not request a field, whether the value is null or not, the value is not returned, and is therefore undefined.

`source` Context Variable

Pass in any context and get it back in the callback method by using the `source` context variable.

For an example of how to use `source` in an error handling context, see Error Handling with the AJAX Toolkit.

Debugging with the AJAX Toolkit

The AJAX Toolkit provides a debugging window that pops up when certain errors are encountered.

You can invoke logging explicitly using the `log` method. For example, if you wanted to display the debugging window with the value of a variable at a certain point in your client application, you could add this line at the appropriate place:

```
sforce.debug.log(myVar);
```

You can open the debugging window at any point by using this command:

```
sforce.debug.open();
```

Example Calls Using the Ajax Toolkit

The next two sections contain examples of synchrononous and asynchronous calls.

The AJAX Toolkit provides a debugging window that pops up when certain errors are encountered.

You can invoke logging explicitly using the `log()` method. For example, if you wanted to display the debugging window with the value of a variable at a certain point in your client application, you could add this line at the appropriate place:

```
sforce.debug.log(myVar);
```

The AJAX Toolkit samples in the following sections use `log()`. To use the samples in the following sections, add this simple version of the `log` code before the first use of `log`:

```
function log(message) {
  alert(message);
}
```

You can make `log()` as sophisticated as you wish.

Examples of Synchronous Calls

Examples of Asynchronous Calls

Examples of Synchronous Calls

[other]: Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Data Call Examples

`login` Example:

```
  try{
    var result = sforce.connection.login("myname@myemail.com", "password");
    log("logged in with session id " + result.sessionId);
  }catch(error) {
    if (error.faultcode.indexOf("INVALID_LOGIN") != -1) {
      log("check your username and passwd, invalid login");
    } else {
      log(error);
    }
  }
```

`query` Example:

```
  result = sforce.connection.query("Select Name, Id from User");
  records = result.getArray("records");

  for (var i=0; i< records.length; i++) {
    var record = records[i];
    log(record.Name + " -- " + record.Id);
  }
```

`queryMore` Example:

```
var result = sforce.connection.query("select name, id from account");

  var queryMore = true;
  while (queryMore) {
      var records = result.getArray("records");
      var sb = new sforce.StringBuffer();

      for (var i = 0; i < records.length; i++) {
        sb.append(records[i].Name).append(",");
      }

      log(records.length);
      log(sb.toString());

      if (result.getBoolean("done")) {
        queryMore = false;
      } else {

        result = sforce.connection.queryMore(result.queryLocator);
    }
  }
```

`queryAll` Example:

```
  var result = sforce.connection.queryAll("Select Name, Id from Account");
  var records = result.getArray("records");

  for (var i=0; i<records.length; i++) {
    var record = records[i];
    log(record.Name + " -- " + record.Id);
  }
```

Relationship Query Example—Child to Parent:

```
  var result = sforce.connection.query("SELECT c.Id, c.firstname, " +
    "c.lastname, c.leadsource, a.Id, a.name, a.industry, c.accountId " +
    "FROM Contact c, c.account a ORDER BY leadsource LIMIT 10");

  var it = new sforce.QueryResultIterator(result);

  while(it.hasNext()) {
    var record = it.next();
    var accountName = record.Account ? record.Account.Name : null;

    log( record.FirstName + " " + record.LastName +
         " in account " + accountName);
  }
```

Note: Relationship name formats differ depending on the direction of the relationship (parent-to-child or child-to-parent), and also depending on whether the objects are standard or custom objects. For more information, see Relationship Queries in the *Salesforce SOQL and SOSL Reference Guide* at `www.salesforce.com/us/developer/docs/soql_sosl/index.htm`.

Relationship Query Example—Parent to Child:

```
var result = sforce.connection.query("select a.Name, a.Industry, " +
  "(select c.LastName, c.LeadSource from a.Contacts c) " +
  "from account a order by industry limit 100");

var ait = new sforce.QueryResultIterator(result);

while(ait.hasNext()) {
  var account = ait.next();

  var contacts = [];
  if (account.Contacts) {
    var cit = new sforce.QueryResultIterator(account.Contacts);
    while(cit.hasNext()) {
      var contact = cit.next();
      contacts.push(contact.LastName);
    }
  }

  log(account.Name + ": " + contacts.join(","));
}
```

`create` Example:

```
var account = new sforce.SObject("Account");
account.Name = "my new account";
var result = sforce.connection.create([account]);

if (result[0].getBoolean("success")) {
  log("new account created with id " + result[0].id);
} else {
  log("failed to create account " + result[0]);
}
```

Batch `create` Example:

```
var accounts = [];

for (var i=0; i<10; i++) {
  var account = new sforce.SObject("Account");
  account.Name = "my new account " + i;
  accounts.push(account);
}

var result = sforce.connection.create(accounts);

for (var i=0; i<result.length; i++) {
  if (result[i].getBoolean("success")) {
    log("new account created with id " + result[i].id);
  } else {
    log("failed to create account " + result[i]);
    }
  }
```

`delete` Example:

```
  //create an example account
  var account = new sforce.SObject("Account");
  account.Name = "my new account";
  var result = sforce.connection.create([account]);

  if (result[0].getBoolean("success")) {
    log("new account created with id " + result[0].id);
    account.Id = result[0].id;
  } else {
    throw ("failed to create account " + result[0]);
  }

  //now delete the example account
  var delResult = sforce.connection.deleteIds([account.Id]);
  if (delResult[0].getBoolean("success")) {
    log("account with id " + result[0].id + " deleted");
  } else {
    log("failed to delete account " + result[0]);
  }
```

`merge` Example:

```
  //create two accounts
  var account1 = new sforce.SObject("Account");
  account1.Name = "myName";
  account1.Phone = "2837484894";

  var account2 = new sforce.SObject("Account");
  account2.Name = "anotherName";
  account2.Phone = "938475950";

  var result = sforce.connection.create([account1, account2]);
  if (result.length != 2) throw "create failed";

  account1.id = result[0].id;
  account2.id = result[1].id;

  //create merge request
  var request = new sforce.MergeRequest();
  request.masterRecord = account1;
  request.recordToMergeIds = account2.id;

  //call merge
  result = sforce.connection.merge([request]);

  if (result[0].getBoolean("success")) {
    log("merge success " + result[0]);
  } else {
    log("merge failed " + result[0]);
  }
```

`process` Example:

```
var request = new sforce.ProcessSubmitRequest();
request.objectId = "id of object that has a workflow rule on it"; // valid id
request.comments = "automated approval";

var request2 = new sforce.ProcessSubmitRequest();
request2.objectId = 'id of object that does NOT have a workflow rule on it' ; // valid id,
 not useful for workflow
request2.comments = "approval that will fail";

var processRes = sforce.connection.process([request, request2]);

if(!processRes[0].getBoolean('success')){
            log("The first process request failed and it should not have");
}

if(processRes[1].getBoolean('success')){
            log("The second process request succeeded and it should not have");
}

log(processRes[0].errors);
log(processRes[1].errors);
```

`update` Example:

```
  //create an account
  var account = new sforce.SObject("Account");
  account.Name = "myName";
  account.Phone = "2837484894";
  result = sforce.connection.create([account]);

  //update that account
  account.id = result[0].id;
  account.Phone = "12398238";
  result = sforce.connection.update([account]);

  if (result[0].getBoolean("success")) {
    log("account with id " + result[0].id + " updated");
  } else {
    log("failed to update account " + result[0]);
  }
```

`undelete` Example:

```
  var account = new sforce.SObject("Account");
  account.Name = "account to delete";
  account.Phone = "2837484894";
  result = sforce.connection.create([account]);
  account.id = result[0].id;
  log("account created " + account);

  result = sforce.connection.deleteIds([account.id]);
  if (!result[0].getBoolean("success")) throw "delete failed";
  log("account deleted " + result);

  result = sforce.connection.undelete([account.id]);
  if (!result[0].getBoolean("success")) throw "undelete failed";
  log("account undeleted " + result[0]);
```

upsert Example:

```
var account = new sforce.SObject("Account");
account.Name = "TestingAjaxUpsert";
account.Phone = "2837484894";
// this will insert an account
var result = sforce.connection.upsert("Id", [account]);

account.Id = result[0].id;
account.Name = "TestingAjaxUpsert2";
// this will update the account
result = sforce.connection.upsert("Id", [account]);

if(result[0].getBoolean("success") && result[0].id == account.Id) {
log("upsert updated the account as expected");
}
else {
log("upsert failed!");
}
```

retrieve Example:

```
  var account = new sforce.SObject("Account");
  account.Name = "retrieve update test";
  account.Phone = "2837484894";
  var result = sforce.connection.create([account]);
  if (result[0].getBoolean("success") == false) throw "create failed";
  log("account created " + result[0]);

  result = sforce.connection.retrieve("Name,Phone", "Account", [result[0].id]);
  if (result[0] == null) throw "retrive failed";
  log("account retrieved: " + result[0]);

  result[0].Phone = "111111111111";
  result = sforce.connection.update(result);
  if (result[0].getBoolean("success") == false) throw "update failed";
  log("account updated: " + result[0]);
```

search Example:

```
  var result = sforce.connection.search(
    "find {manoj} in Name fields RETURNING Account(name, id)");

  if (result) {
     var records = result.getArray("searchRecords");

     for (var i=0; i<records.length; i++) {
       var record = records[i].record;
       log(record.Id + " -- " + record.Name);
    }
}
```

`getDeleted` Example:

```
var start = new Date();
var end = new Date();
start.setDate(end.getDate() - 1);

var result = sforce.connection.getDeleted("Account", start, end);

var records = result.getArray("deletedRecords");

log("following records are deleted:");

for (var i=0; i<records.length; i++) {
  log(records[i].id);
}
```

`getUpdated` Example:

```
var start = new Date();
var end = new Date();
start.setDate(end.getDate() - 1);

var result = sforce.connection.getUpdated("Account", start, end);

var records = result.getArray("ids");

log("following records are updated:");
for (var i=0; i<records.length; i++) {
  log(records[i]);
}
```

`convertLead` Example:

```
var account = new sforce.SObject("Account");
account.Name = "convert lead sample";
account.Phone = "2837484894";
result = sforce.connection.create([account]);
account.Id = result[0].id;

var lead = new sforce.SObject("Lead");
lead.Country = "US";
lead.Description = "This is a description";
lead.Email = "someone@somewhere.com";
lead.FirstName = "first";
lead.LastName = "last";
lead.Company = account.Name;
result = sforce.connection.create([lead]);
lead.Id = result[0].id;

var convert = new sforce.LeadConvert();
convert.accountId = account.Id;
convert.leadId = lead.Id;
convert.convertedStatus = "Qualified";
```

```
result = sforce.connection.convertLead([convert]);
if (result[0].getBoolean("success")) {
  log("lead converted " + result[0]);
} else {
  log("lead convert failed " + result[0]);
}
```

Describe Examples

`describeSObject` Account Example:

```
var result = sforce.connection.describeSObject("Account");

log(result.label + " : " + result.name + " : ");

log("---------- fields ---------");
for (var i=0; i<result.fields.length; i++) {
  var field = result.fields[i];
  log(field.name + " : " + field.label + " : " + field.length + " : ");
}

log("---------- child relationships ---------");

for (var i=0; i<result.childRelationships.length; i++) {
  var cr = result.childRelationships[i];
  log(cr.field + " : " + cr.childSObject);
}

log("---------- record type info ----------");
for (var i=0; i<result.recordTypeInfos.length; i++) {
  var rt = result.recordTypeInfos[i];
  log(rt.name);
}
```

`describeSObjects` Example:

```
var result = sforce.connection.describeSObjects(["Account", "Contact"]);

for (var i=0; i<result.length; i++) {
  log(result[i].label + " : " + result[i].name + " : ");
}
```

`describeGlobal` Example:

```
var result = sforce.connection.describeGlobal();

var sobjects = result.getArray("sobjects");
for (var i=0; i<sobjects.length; i++) {
  log(sobjects[i].name);
}
```

`describeLayout` Example:

```
var result = sforce.connection.describeLayout("Account");

var layouts = result.getArray("layouts");

for (var i=0; i<layouts.length; i++) {
  var layout = layouts[0];
  detailLayoutSections(layout.detailLayoutSections);
}

 function detailLayoutSections(sections) {
  for (var i=0; i<sections.length; i++) {
    var section = sections[i];
    log(section.columns + ":" + section.heading + ":");
    layoutRows(section.getArray("layoutRows"));
  }
}

function layoutRows(rows) {
  for (var i=0; i<rows.length; i++) {
    var row = rows[i];
    layoutItems(row.getArray("layoutItems"));
  }
}

function layoutItems(items) {
  for (var i=0; i<items.length; i++) {
    var item = items[i];
    log("   " + item.label);
  }
}
```

`describeTabs` Example:

```
var result = sforce.connection.describeTabs();

for (var i=0; i<result.length; i++) {
  var tabSet = result[i];
  log( tabSet.label);
  displayTabs(tabSet.get("tabs"));
}

function displayTabs(tabs) {
  for( var i=0; i<tabs.length; i++) {
    var tab = tabs[i];
    log( "   " + tab.label + " " + tab.url);
  }
}
```

Utility Examples

`getServerTimestamp` Example:

```
  var result = sforce.connection.getServerTimestamp();
  log(result.timestamp);
```

`getUserInfo` Example:

```
  var user = sforce.connection.getUserInfo();
  log("Hello " + user.userName);
  log("Your email id is " + user.userEmail);
  log("and you work for " + user.organizationName);
```

`resetPassword` and `setPassword` Example:

```
 var username = "myname@myemail.com";

  var result = sforce.connection.query(
    "SELECT ID from User WHERE User.username='" + username + "'");

  var records = result.getArray("records");
  if (records.length != 1) throw "unable to find user";
  var id = records[0].Id;

  sforce.connection.resetPassword(id);
  sforce.connection.setPassword(id, "123456");
```

`sendEmail` Example:

```
// single mail request
var singleRequest = new sforce.SingleEmailMessage();
singleRequest.replyTo = "jsmith@acme.com";
singleRequest.subject = "sent through ajax test driver";

singleRequest.plainTextBody = "this test went through ajax";
singleRequest.toAddresses = ["noone@nowhere.com"];

// mass mail request - need to get email template ID

var queryResponse = sforce.connection.query("select id from emailtemplate");
var templatedId = queryResponse.getArray("records")[0].Id;
var massRequest = new sforce.MassEmailMessage();
massRequest.targetObjectIds = [globalContact.id];
massRequest.replyTo = "jsmith@acme.com";
massRequest.subject = "sent through ajax test driver";
massRequest.templateId = templateId;

var sendMailRes = sforce.connection.sendEmail([singleRequest, massRequest]);
```

The following sample shows best practice techniques by putting all processing in a function that does not execute until the HTML page is loaded.

```
<html>
<head>
  <script src="/soap/ajax/63.0/connection.js"></script>
  <script>
```

```
    var contactId = "{!Contact_ID}";
    function initPage() {
 try{
        var contact = sforce.connection.retrieve("AccountId", "Contact", [contactId])[0];

        var accountsRetrieved = sforce.connection.retrieve("Id, Name, Industry,
                                LastModifiedDate", "Account", [contact.AccountId]);
        if (accountsRetrieved.length > 0) {
             var account = accountsRetrieved.records[0];
             document.body.innerHTML += "Account name: <a href='/" + account.Id;
             document.body.innerHTML += "' target='_blank'>" + account.Name + "</a><br>;
             document.body.innerHTML += "Industry: " + account.Industry + "<br>";
        }
 } catch (e) {
    document.body.innerHTML += "Error retrieving contact information";
    document.body.innerHTML += "<br>Fault code: " + e.faultcode;
    document.body.innerHTML += "<br>Fault string: " + e.faultstring;
    }
 }
 </script>
</head>
<body onload="initPage();">

</body>
</html>
```

Examples of Asynchronous Calls

`query` Example:

```
  var result = sforce.connection.query("Select Name,Id from User", {
      onSuccess : success,
      onFailure : failure
    });

  function success(result) {
    var records = result.getArray("records");

    for (var i=0; i<records.length; i++) {
      var record = records[i];
      log(record.Name + " -- " + record.Id);
    }
  }

  function failure(error) {
    log("An error has occurred " + error);
  }
```

`query` Inline Function Example:

```
  var result = sforce.connection.query("Select Name,Id from User", {
      onSuccess : function(result) {
         var records = result.getArray("records");
```

```
        for (var i=0; i<records.length; i++) {
          var record = records[i];
          log(record.Name + " -- " + record.Id);
        }
     },
     onFailure : function(error) {
       log("An error has occurred " + error);
     }
});
```

`query` With `LIMIT` Example:

```
var result = sforce.connection.query("Select Name, Id from Account
        order by Name limit 10", {
  onSuccess : success, onFailure : failure});

function success(result) {
  var it = new sforce.QueryResultIterator(result);
  while(it.hasNext()){
    var record = it.next();
    log(record.Name + " -- " + record.Id);
  }
}

function failure(error) {
  log("An error has occurred " + error);
}
```

`queryResultIterator` Example:

```
var result = sforce.connection.query("Select Name,Id from User", {
  onSuccess : success, onFailure : failure});

function success(result) {
  var it = new sforce.QueryResultIterator(result);
  while(it.hasNext()){
    var record = it.next();
    log(record.Name + " -- " + record.Id);
  }
}

function failure(error) {
  log("An error has occurred " + error);
}
```

`queryMore` Example:

```
sforce.connection.query("Select Name,Id from Account", {
    onSuccess : success, onFailure : log });

function success(result) {
  var records = result.getArray("records");

  var sb = new sforce.StringBuffer();
  for (var i=0; i<records.length; i++) {
    var record = records[i];
    sb.append(record.Name).append(",");
  }

  log(records.length);
  log(sb.toString());

  if (result.queryLocator) {
    sforce.connection.queryMore(result.queryLocator, {
        onSuccess : success, onFailure : log});
  }
}
```

`create` Example:

```
var account = new sforce.SObject("Account");
account.Name = "my new account";

sforce.connection.create([account],
  {onSuccess : success, onFailure : failed});

function success(result) {
  if (result[0].getBoolean("success")) {
    log("new account created with id " + result[0].id);
  } else {
    log("failed to create account " + result[0]);
  }
}

function failed(error) {
  log("An error has occurred " + error);
}
```

`create` Other Objects Example:

```
var campaign = new sforce.SObject("Campaign");
campaign.Name = "new campaign";
campaign.ActualCost = 12938.23;
campaign.EndDate = new Date();
campaign.IsActive = true;

sforce.connection.create([campaign ],
  {onSuccess : success, onFailure : log});

function success(result) {
  if (result[0].getBoolean("success")) {
    log("new campaign created with id " + result[0].id);
  } else {
    log("failed to create campaign " + result[0]);
  }
}
```

`retrieve` Account Example:

```

var account = new sforce.SObject("Account");
```

```
account.Name = "retrieve update test";
account.Phone = "2837484894";
var result = sforce.connection.create([account]);
if (result[0].getBoolean("success") == false) throw "create failed";
log("account created " + result[0]);

var callback = {
  onSuccess: function(result) {
    if (result[0] == null) throw "retrive failed";
    log("account retrieved: " + result[0]);
  },
  onFailure: function(error) {
    log("failed due to " + error);
  }
};

result = sforce.connection.retrieve("Name,Phone", "Account",
   [result[0].id], callback);
```

API End-of-Life Policy

See which SOAP API versions are supported, unsupported, or unavailable.

Salesforce is committed to supporting each API version for a minimum of 3 years from the date of first release. To improve the quality and performance of the API, versions that are over 3 years old sometimes are no longer supported.

Salesforce notifies customers who use an API version scheduled for deprecation at least 1 year before support for the version ends.

Salesforce API Versions	Version Support Status	Version Retirement Info
Versions 31.0 through 63.0	Supported.	
Versions 21.0 through 30.0	As of Summer '22, these versions have been deprecated and no longer supported by Salesforce. Starting Summer '25, these versions will be retired and unavailable.	Salesforce Platform API Versions 21.0 through 30.0 Retirement
Versions 7.0 through 20.0	As of Summer '22, these versions are retired and unavailable.	Salesforce Platform API Versions 7.0 through 20.0 Retirement

If you request any resource or use an operation from a retired API version, SOAP API returns `500:UNSUPPORTED_API_VERSION` error code.

To identify requests made from old or unsupported API versions, use the API Total Usage event type.

SOAP Header Options with the AJAX Toolkit

All header options in the SOAP API are supported in the toolkit, but they are specified differently than in the API.

Syntax for Specifying Header Options

- For headers that have only one option such as queryOptions:

```
sforce.connection.header_option_name="value";
```

- For headers that have more than one option such as assignmentRuleHeader:

```
sforce.connection.header_name = {}
sforce.connection.header_name.header_option_name="value";
```

Valid Options

Here's each valid option, organized by its corresponding SOAP header name in the API for your reference.

- From the assignmentRuleHeader:

  `assignmentRuleId`
  ID of a specific assignment rule to run for the case or lead. Can be an inactive assignment rule. The ID can be retrieved by querying the AssignmentRule object. If specified, do not specify `useDefaultRule`. This element is ignored for accounts, because all territory assignment rules are applied. If the value is not in correct ID format (15-character or 18-character Salesforce ID), the call fails and a `MALFORMED_ID` exception is returned.

  `useDefaultRule`
  If true for a Case or Lead, uses the default (active) assignment rule for a Case or Lead. If specified, do not specify an `assignmentRuleId`. If true for an Account, all territory assignment rules are applied, and if false, no territory assignment rules are applied.

- From callOptions:

  `client`
  A string that identifies a particular client.

  `defaultNamespace`
  A string that identifies a developer namespace prefix.

- From emailHeader:

  `triggerAutoResponseEmail`
  Indicates whether to trigger auto-response rules (`true`) or not (`false`), for leads and cases. In the Salesforce user interface, this email can be automatically triggered by a number of events, for example resetting a user password.

  `triggerOtherEmail`
  Indicates whether to trigger email outside the organization (`true`) or not (`false`). In the Salesforce user interface, this email can be automatically triggered by creating, editing, or deleting a contact for a case.

  `triggerUserEmail`
  Indicates whether to trigger email that is sent to users in the organization (`true`) or not (`false`). In the Salesforce user interface, this email can be automatically triggered by a number of events; resetting a password, creating a new user, adding comments to a case, or creating or modifying a task.

- From loginScopeHeader:

  `organizationId`
  The ID of the organization against which you will authenticate Self-Service users.

`portalId`

Specify only if user is a Customer Portal user. The ID of the portal for this organization. The ID is available in the Salesforce user interface:

- From Setup, enter *`Customer Portal Settings`* in the `Quick Find` box, then select **Customer Portal Settings**
- Select a Customer Portal name, and on the Customer Portal detail page, the URL of the Customer Portal displays. The Portal ID is in the URL.

- From mruHeader:

  `updateMru`
  Indicates whether to update the list of most recently used items (`true`) or not (`false`). For retrieve, if the result has only one row, MRU is updated to the ID of the retrieve result. For query, if the result has only one row and the ID field is selected, the MRU is updated to the ID of the query result.

- From queryOptions:

  `batchSize`
  Batch size for the number of records returned in a query or queryMore call. Child objects count toward the number of records for the batch size. For example, in relationship queries, multiple child objects may be returned per parent row returned. The default is 500; the minimum is 200, and the maximum is 2,000.

- From sessionHeader:

  `sessionId`
  Session ID returned by the login call to be used for subsequent call authentication. Since session management is done for you by the AJAX Toolkit, most scripts won't need to use this header option.

- From userTerritoryDeleteHeader:

  `transferToUserId`
  The ID of the user to whom open opportunities in that user's territory will be assigned when an opportunity's owner (user) is removed from a territory.

Error Handling with the AJAX Toolkit

The AJAX Toolkit provides the ability to handle errors for synchronous and asynchronous calls.

Error Handling for Synchronous Calls

If the API call fails, then the AJAX Toolkit throws an exception. The exception contains all the available error information. For example:

```
<html>
<head>
    <script src="/soap/ajax/63.0/connection.js" type="text/javascript"></script>

    <script>

        function setupPage() {
            var output = document.getElementById("output");
            var startTime = new Date().getTime()

             try {
                var queryResult = sforce.connection.query("Select Id, Name, Industry From

                     Account order by Industry limit 30");
                layoutResults(queryResult, output, startTime);
            } catch(error) {
                queryFailed(error, output);
            }
        }

        function queryFailed(error, out) {
            out.innerHTML = "<font color=red>An error has occurred:</font> <p>" + error;
        }

        function layoutResults(queryResult, out, startTime) {
            var timeTaken = new Date().getTime() - startTime;

            if (queryResult.size > 0) {
                var output = "";
                var records = queryResult.getArray('records');

                for (var i = 0; i < records.length; i++) {
                    var account = records[i];
                    output += account.Id + " " + account.Name + " [Industry - "
                         + account.Industry + "]<BR>";
                }

               out.innerHTML = output + "<BR> query complexted in: " + timeTaken + " ms.";

            } else {
                out.innerHTML = "No records matched.";
            }
        }

    </script>
</head>

 <body onload="setupPage()">
<div id="output"></div>
</body>
</html>
```

Error Handling for Asynchronous Calls

For asynchronous calls, the `onFailure` property of the asynchronous object is called. For example:

```
connection.query("Select Name From Account",
   {onSuccess: displayResult,
       onFailure: queryFailed});

function displayResult(result){}
function queryFailed(error){}
```

If the `onFailure` property was not defined, the AJAX Toolkit pops up a new read-only browser window showing the error.

Advanced Topics

This chapter contains information about advanced activities in the AJAX Toolkit.

QueryResultIterator

Iterate over query results returned by the AJAX Toolkit without invoking `queryMore` and `queryLocator`.

```
var result = sforce.connection.query("select id, name from account");
var it = new sforce.QueryResultIterator(result);

while (it.hasNext()) {
    var account = it.next();
    sforce.debug.log(account.Name);
}
```

1. The `sforce.connection.query` method returns a QueryResult object.
2. A QueryResultIterator object is created and passed the QueryResult object.
3. The code iterates through the records.

Differences in Escaping Reserved Characters

If you have a single quote or backslash in a string literal, use two backslashes instead of one to escape it.

For example, the following statement in a Java client program is valid for finding account names like Bob's B-B-Q.

```
SELECT ID from ACCOUNT WHERE Name LIKE 'Bob\'s B-B-Q%'
```

For the AJAX Toolkit, escape the single quote literal character twice.

```
SELECT ID from ACCOUNT WHERE Name LIKE 'Bob\\'s B-B-Q%'
```

Working with Base64 Binary Encoded Strings

When working with Base64 encoded binary documents, access the document directly using the Id, rather than decoding Base64 in JavaScript.

Base64 encoding and decoding is very slow in JavaScript. Also, encoding and decoding does not work correctly for binary or multibyte strings. We do not recommend that you manipulate Base64 binary encoded strings with the AJAX Toolkit. However, if you want to read a document with Base64 binary encoding, you can use the API to query for the Id of the document and then download it directly from the server.

The following example demonstrates how to query for the document Id and then download it from the server.

```
<html>
<head>
<script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/dojo/1.10.4/dojo/dojo.js"></script>
<script src="/soap/ajax/63.0/connection.js"></script>

<script>
function setup() {
  var document_ta = document.getElementById("document-ta");

  sforce.connection.query("select name, id from document limit 1",
    {onSuccess : querySuccess,
     onFailure : function(error, doc_ta) {
        doc_ta.value = "Oops something went wrong: " + error;
     },
     source: document_ta});
}

function querySuccess(result, doc_ta) {
  var records = result.getArray("records");

  if (records.length == 1) {
    dojo.io.bind({
      url: "/servlet/servlet.FileDownload?file=" + records[0].Id,
      load: loadDocument});
  } else {
    doc_ta.value = "no records found";
  }
}

function loadDocument(type, data, event) {
  var document_ta = document.getElementById("document-ta");
  document_ta.value = data;
}

</script>
</head>

<body onload="setup()">
<textarea id="document-ta" cols="80" rows="20">
</textarea>
</body>
</html>
```

Note: This example uses the JavaScript toolkit Dojo, which you'll need to upload as a static resource, reference from a CDN, or otherwise provide. For more information, see `http://dojotoolkit.org/`.

Timeout Parameter for Asynchronous Calls

If an asynchronous call does not complete in an appropriate amount of time, you can end the call. To do this, specify the `timeout` parameter in the callback section of any asynchronous call.

```
var account = new sforce.SObject("Account");
account.Name = "my new account";

sforce.connection.create([account], {onSuccess: print, onFailure: printerr, timeout: 100});
```

Values for this parameter are in milliseconds, and valid values are integers beginning with `1`.

If the call is successful within the time specified by the callout, no additional actions are taken. If the call is not successful, the `onFailure` action is performed.

Warning: Use this parameter with caution. Because the `timeout` is performed on the client side, it is possible that the call may complete on the server but the `timeout` is still triggered. For example, you might issue a create call to create 100 new accounts, and any number of them, 1 or 100, might be created just before the `timeout` is triggered; your `onFailure` action would still occur, but the accounts would have been created.

AJAX Proxy

Some browsers don't allow JavaScript code to connect to external servers directly. Therefore, you may need to send requests through the AJAX proxy.

Note: To use the AJAX proxy, you must register all external services in the Salesforce user interface. From Setup, enter *`Remote Site Settings`* in the `Quick Find` box, then select **Remote Site Settings**.

For security reasons, Salesforce restricts the outbound ports you may specify to one of the following:

- 80: This port only accepts HTTP connections.
- 443: This port only accepts HTTPS connections.
- 1024–66535 (inclusive): These ports accept HTTP or HTTPS connections.

The AJAX proxy is part of the AJAX Toolkit. Access it using `remoteFunction` defined in `connection.js`. You can specify any HTTP method in `remoteFunction`, for example HTTP GET or POST, and it will be forwarded to the external service.

The following examples illustrate typical approaches for GET and POST:

GET Example:

```
          sforce.connection.remoteFunction({
                url : "http://www.myExternalServer.com",
                onSuccess : function(response) {
                      alert("result" + response);
                   }
              });
```

POST Example:

```
           var envelope = ""; //request envelope, empty for this example
           sforce.connection.remoteFunction({
```

```
                    url : "http://services.xmethods.net:80/soap",
                    requestHeaders: {"Content-Type":"text/xml",
                           "SOAPAction": "\"\""
                       },
                    requestData: envelope,
                    method: "POST",
                    onSuccess : function(response) {
                           sforce.debug.log(response);
                       },
                    onFailure : function(response) {
                           alert("Failed" + response)
                       }
                });
```

remoteFunction Syntax and Parameters

AJAX proxy uses `remoteFunction` to proxy calls.

Download the Salesforce Client Certificate

Your application (endpoint) server's SSL/TLS can be configured to require client certificates (two-way SSL/TLS) to validate the identity of the Salesforce server when it takes the role of client to your server. If so, you can download the Salesforce client certificate from the Salesforce API page.

`remoteFunction` Syntax and Parameters

AJAX proxy uses `remoteFunction` to proxy calls.

The `remoteFunction` syntax and parameters:

```
sforce.connection.remoteFunction({
url : endpoint_url,
onSuccess : callback_method
onFailure : error_callback
method : http_method
mimeType : "text/plain" | "text/xml"
async : true | false
requestHeaders : http_headers
requestData : http_post_data
cache : true | false
timeout : client_side_timeout_in_ms
});
```

Note: `cache` and `timeout` are available in version 10.0 and later.

Download the Salesforce Client Certificate

Your application (endpoint) server's SSL/TLS can be configured to require client certificates (two-way SSL/TLS) to validate the identity of the Salesforce server when it takes the role of client to your server. If so, you can download the Salesforce client certificate from the Salesforce API page.

1. From Setup, enter *`API`* in the `Quick Find` box, then select **API**.
2. On the API WSDL page, click **Manage API Client Certificate**.
3. On the Certificate and Key Management page, in the API Client Certificate section, click the **API Client Certificate**.

4. On the Certificates page, click **Download Certificate**. The .crt file is saved in the download location specified in your browser.

   Import the downloaded certificate into your application server, and configure your application server to request the client certificate. The application server then checks that the certificate used in the SSL/TLS handshake matches the one you downloaded.

Note: Your application (endpoint) server must send intermediate certificates in the certificate chain, and the certificate chain must be in the correct order. The correct order is:

1. Server certificate
2. Intermediate certificate that signed the server certificate if the server certificate wasn't signed directly by a root certificate
3. Intermediate certificate that signed the certificate in step 2
4. Any remaining intermediate certificates

   Don't include the root certificate authority certificate. The root certificate isn't sent by your server. Salesforce already has its own list of trusted certificates on file, and a certificate in the chain must be signed by one of those root certificate authority certificates.

INDEX