

The new role of XML in cloud data integration

Using XML to integrate Salesforce data with enterprise applications

Skill Level: Intermediate

[Ryan Knight \(ryan@anvilflex.com\)](mailto:ryan@anvilflex.com)

Senior Technical Evangelist

Freelance

30 Jun 2009

Learn how to leverage XML Web services to integrate cloud data with enterprise applications, and build a sample application using the Salesforce Web Services API with the Java API for XML Web Services (JAX-WS).

Cloud computing: redefining IT

Over the past year, cloud computing has exploded to include a wide variety of applications—such as Salesforce CRM and Google Apps—and services—such as IBM® DB2® hosted on Amazon Elastic Compute Cloud (Amazon EC2), Google App Engine, and the Force.com platform by Salesforce. These services are often called a *Platform-as-a-Service* (PaaS), because the services provide a complete platform on which businesses can build and host their IT applications.

Frequently used acronyms

- API: Application programming interface
- HTTPS: Hypertext Transfer Protocol over Secure Sockets Layer
- IT: Information technology
- SOAP: Simple Object Access Protocol
- UI: User Interface

- URL: Universal Resource Locator
- XML: Extensible Markup Language

PaaS offerings are hosted in a multi-tenant environment, where the hardware and software infrastructure is shared. These environments are set up to ensure that each organization's data is isolated in a secure manner from other organizations—similar to leasing rather than buying office space so you don't have to worry about maintenance and upgrades. With PaaS, organizations realize not only significant cost savings but also unique advantages such as automatic upgrades and zero maintenance. The model also offers a unique advantage for reducing the risks common with new IT projects that often run over budget and miss deadlines.

Salesforce is one example of a PaaS provider. The company started out by providing a hosted customer relationship management (CRM) solution, and this offering provided a turnkey solution for common business applications such as sales, partner relationship management, and marketing. Salesforce built on this platform to launch Force.com, a complete PaaS for building custom business applications either in a browser or by using the Force.com integrated development environment (IDE) based on Eclipse. Applications can also be customized using a proprietary Java™-like programming language called *Apex*.

Salesforce allows you to interact with the server using SOAP, which has the advantage of being language and platform independent. You use Web Services Description Language (WSDL) to describe which operations are available on the server. A WSDL document describes a series of network endpoints called ports, then defines the XML format for the messages or data being exchanged with the server.

The samples in this article use the Java API for XML Web Services (JAX-WS). JAX-WS provides a number of tools to simplify working with XML and SOAP, including automatically generating the necessary domain objects from a WSDL document and automatically binding the XML to Java objects.

Integrating Salesforce with enterprise applications

You can build applications that integrate with Salesforce in a number of ways. The first method is configuring a workflow in Salesforce to send a message using Web services any time a record is created or updated. This process uses an outbound message WSDL—a process similar to configuring a trigger on a database. A workflow configures Salesforce to send a SOAP message to a preconfigured URL destination whenever data is created, updated, or deleted. For example, in the Mileage example in this article, you can send the mileage report to an internal system any time a new report is added.

The second method provides the ability to directly interact with Salesforce using SOAP Web services. This method uses Salesforce-generated WSDL files that are consumed by custom Web services. For this type of integration, Salesforce provides two primary types of WSDL:

- **Partner WSDL** is loosely typed and can be used across multiple organizations; the Partner WSDL is more difficult to work with, because the XML has to be marshaled into the correct object representation for the organization.
- **Enterprise WSDL** is strongly typed and bound to a single organization; this makes the WSDL easier to work with, but it only works with the schema of one organization. Also, any changes to objects require the WSDL to be regenerated.

This article demonstrates integration with Salesforce by expanding on the example used in the Force.com Workbook: a mileage-tracking application (see [Resources](#) for links to the tutorials). For the samples, you will use the enterprise Web Services WSDL.

When working with Salesforce Web services, you must have a security token. To receive or reset the security token, use the Salesforce UI. Click **Setup > My Personal Information > Reset your security token**. There, you'll see an option to reset your security token and have it sent to your e-mail address. After you have your security token, use it in combination with your password to log in. For example, if your password is `aaaaaa` and the security token is `xxxxxxxxxxx`, you must type `aaaaaaXXXXXXXXXX` in place of your password.

Integrating with Salesforce through Web services

To provide an example of how to use XML to integrate with Salesforce, I developed a sample Java application that logs in to Salesforce, queries for data, and creates new records. All communication with Salesforce goes over the secure HTTPS channel. Follow these steps to set up Java Web services to integrate with Salesforce:

1. Generate the Enterprise WSDL document in Salesforce.
Log in to your organization's Salesforce account and click **Setup > Develop > API**. Then, right-click the WSDL document you want to generate.
2. Generate the Java classes and domain objects from the WSDL.
For the sample, I used the *wsimport* tool included in the JAX-WS toolkit.

To simplify the process of running wsimport, the sample uses the build template that is part of the Salesforce samples. The Java domain objects that this tool generates allows the XML documents to be automatically bound to Java objects to simplify communication with the server.

3. Log in to Salesforce through Web services.
You must log in first to acquire a server URL and session ID. The code in [Listing 1](#) shows how to log in.

An important part of the login is getting the server URL. Salesforce runs on multiple instances, such as na1.salesforce.com or na2.salesforce.com, to improve reliability and performance. Once you log in to a server, you need to use the same URL for all subsequent Web service calls to maintain the session information.

Listing 1. Log in to Salesforce

```
public void doLogin(String userName, String password) {
    if (userName.length() == 0 || password.length() == 0) {
        throw new RuntimeException("user name length and/or password length
            cannot be 0 length. \n",
            new IllegalArgumentException("Invalid password or user name\n"));
    } else {
        try {
            URL wsdlLocation =
                this.getClass().getClassLoader().
                    getResource("etc/enterprise.wsdl");
            if (wsdlLocation == null) {
                WebServiceException e =
                    new WebServiceException("enterprise.wsdl not found!");
                //exceptionLogger(e.getMessage(), e);
                throw e;
            }
            port = new SforceService(wsdlLocation,
                new QName("urn:enterprise.soap.sforce.com",
                    "SforceService")).getSoap();
        } catch (WebServiceException wse) {
            //exceptionLogger("Error creating salesface port ", wse);
            throw wse;
        }

        try {
            loginResponse = port.login(userName, password);
        } catch (Exception e) {
            System.out.println("Error logging in to Salesforce.com " + e);
            return;
        }

        System.out.println("Login was successful.");
        System.out.print("The returned session id is: ");
        System.out.println(loginResponse.getSessionId());
        System.out.print("Your logged in user id is: ");
        System.out.println(loginResponse.getUserId() + " \n\n");
        System.out.print("The server url is: ");
        System.out.println(loginResponse.getServerUrl() + " \n\n");
        // on a successful login, you should always set up your session id
        // and the url for subsequent calls
    }
}
```

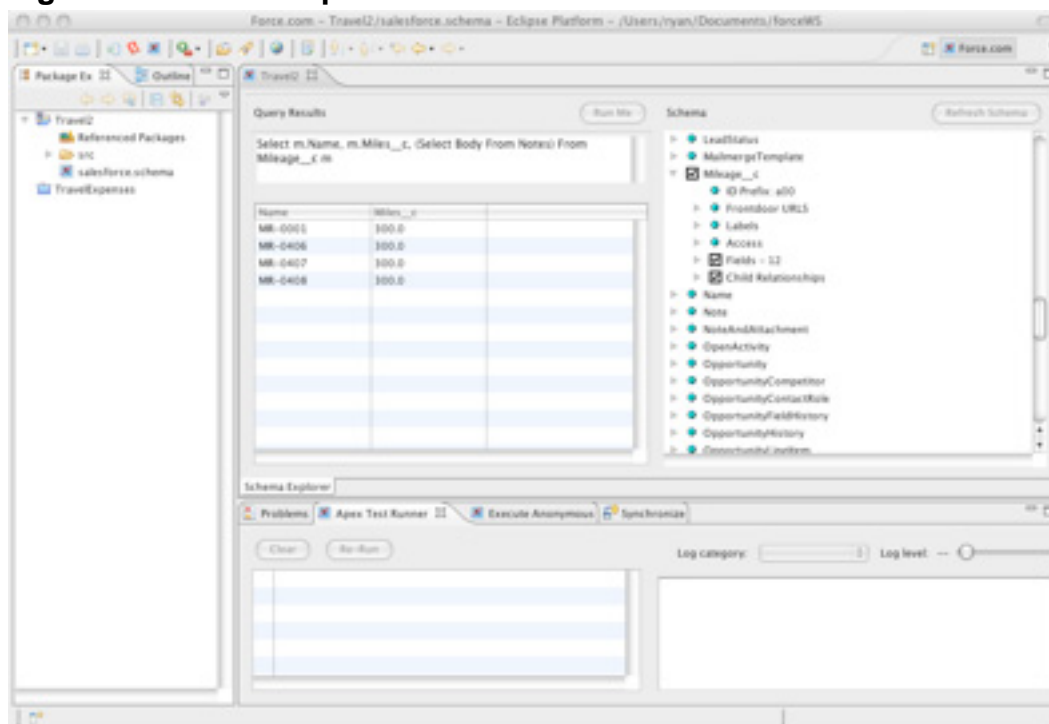
Now that you have set up the client session, you can call and interact with the server through Web services calls. This paper covers two operations: reading data from the server through queries and adding new records to the server.

Querying Salesforce through Web services

To query Salesforce data, you must use a special language called the *Salesforce Object Query Language* (SOQL). You can use SOQL queries to search specific objects or fields of an object, similar to how `SELECT` queries specific fields in a table. You can also use SOQL queries to count the number of records that meet a search criterion and sort the results in a specific order.

The best way to become familiar with SOQL is to use Schema Explorer in the Force.com IDE (see [Figure 1](#)). To open Schema Explorer in a project, double-click `salesforce.schema` in the root of your project. Within Schema Explorer, you can browse all the objects for your organization; by drilling into the various objects, you can select individual objects or fields to automatically generate a query. You can also include related objects in a query by drilling into the listing of child relationships for an object.

Figure 1. Schema Explorer



When you have perfected the SOQL query you want to execute, embed it in your

Web services call. The result from this query is an XML representation of your data. This XML document is automatically bound to the Java domain objects you previously generated with JAX-WS.

For the sample application, you're simply going to get all the mileage records. The results of the query are then bound to the domain objects generated in the setup, as in [Listing 2](#).

Listing 2. Query results for the mileage search

```
public void getMileageReports(ForceLogin login) throws UnexpectedErrorFault,
    InvalidSObjectFault, InvalidIdFault, InvalidQueryLocatorFault,
    MalformedQueryFault, InvalidFieldFault {

    QueryResult queryResult = login.port
        .query("Select Contact__c, Date__c, Miles__c from Mileage__c");

    if (queryResult.getSize() > 0) {
        List<SObject> records = queryResult.getRecords();
        for (SObject record : records) {
            MileageC mileageC = (MileageC) record;
            System.out.println(mileageC.getMilesC().getValue()
                + " " + mileageC.getContactC().getValue());
        }
    }
}
```

An important thing to remember when you write these queries is to distinguish custom objects and fields from standard objects and fields. Custom objects are custom Salesforce database tables that are specific to an organization. To ensure custom objects and fields are unique from standard Salesforce objects with the same name, append `__c` after the name of the object. This distinguishes them from standard objects and fields.

The query in [Listing 2](#) gets all the records from the `Mileage` object. Notice the `__c` convention that clarifies names of the custom fields from the mileage object. You can then iterate through the query results, binding them to the `MileageC` object, a domain object that was generated from the enterprise WSDL. The `MileageC` objects will automatically be populated with the actual data for the mileage records.

The previous query returned all mileage records, which is often more than you need. Fortunately, you can specify a filter criterion—for example, a date or contact—so that you get only a certain subset of records. Suppose you wanted to find all mileage reports in which the miles were greater than 300. You can change the query to be:

```
Select Name, Miles__c From Mileage__c where Miles__c > 300
```

Another way to filter the data is to get only the data that changed over a certain time period. To do this you use the `getUpdated` method instead of running a query,

which takes the object to be searched for updates along with a start and end date. `getUpdated` returns all objects that changed in the time period specified. The code in [Listing 3](#) returns all mileage records that changed in the past month.

Listing 3. Getting updated mileage records

```

GregorianCalendar cal =
    port.getServerTimestamp().getTimestamp().
        toGregorianCalendar();
GregorianCalendar calEnd = (GregorianCalendar) cal.clone();
cal.add(GregorianCalendar.MONTH, -1);
DatatypeFactory datatypeFactory = DatatypeFactory.newInstance();
GetUpdatedResult updatedRecords = port.getUpdated("Mileage",
    datatypeFactory.newXMLGregorianCalendar(cal),
    datatypeFactory.newXMLGregorianCalendar(calEnd));

```

A similar operation to `getUpdated` is `getDeleted`. This method has the same method signature and returns all the records deleted over a certain time period.

The other option for querying Salesforce is by using *Salesforce Object Search Language* (SOSL). SOSL is similar to text-based searches, allowing a more free-form search. With SOSL, it is possible to search across multiple, unrelated objects even when you don't know which object or objects contain the data. Queries made with SOSL can also return multiple unrelated objects.

The code in [Listing 4](#)—an example of an SOSL query from the Salesforce JAX-WS sample—is an excellent example of this flexibility. This code queries contacts, leads, and accounts for a phone number, demonstrating how you can search across multiple unrelated objects.

Listing 4. Searching across multiple objects for a phone number

```

SearchResult searchResult = port
    .search("find {4159017000} in phone fields returning
        contact(id, phone, firstname, lastname),
        lead(id, phone, firstname, lastname),
        account(id, phone, name)");
List<SearchRecord> records = searchResult.getSearchRecords();
List<Contact> contacts = new ArrayList<Contact>();
List<Lead> leads = new ArrayList<Lead>();
List<Account> accounts = new ArrayList<Account>();

if (records != null && !records.isEmpty()) {
    for (SearchRecord recordType : records) {
        SObject record = recordType.getRecord();
        if (record instanceof Contact) {
            contacts.add((Contact) record);
        } else if (record instanceof Lead) {
            leads.add((Lead) record);
        } else if (record instanceof Account) {
            accounts.add((Account) record);
        }
    }
}

```

In this example, the `search` method is used instead of `query` to execute an SOSL query. The results are mapped to a `SearchResults` object, and the individual records of the search results are then inspected to determine their type.

Adding data to Salesforce through Web services

To add data to Salesforce, you use the same domain objects generated from the enterprise WSDL. These objects are populated like normal objects, then serialized as XML to the server for persistence. The one challenge is creating relationships between the different objects. For example, suppose you want to add a new mileage record for a specific contact. First, you need to look up the contact ID you want to use. In this example, you search contacts for a person with the first name *Edna*, as in [Listing 5](#).

Listing 5. Finding a specific contact in the database

```
QueryResult qr = login.port
    .query("Select Id, FirstName, LastName, AccountId " +
        "from Contact where FirstName = 'Edna'");

String contactID = null;
if (qr.getSize() > 0) {
    Contact contact = (Contact) qr.getRecords().get(0);
    contactID = contact.getId().getValue();
}
```

This query provides the ID of the first contact in a list. (In a real application, you want to do more extensive error checking to be sure you have the correct record.) When you have the ID of the user, you store that in your new mileage record, in [Listing 6](#).

Listing 6. Creating a new mileage record

```
MileageC mileageC = new MileageC();
GregorianCalendar cal = new GregorianCalendar();
cal.setTime(new Date());
XMLGregorianCalendar activityDate =
    DatatypeFactory.newInstance().newXMLGregorianCalendar(cal);
mileageC.setDateC(soFactory.createMileageCDateC(activityDate));
mileageC.setMilesC(soFactory.createMileageCMilesC(new Double(300)));
mileageC.setContactC(soFactory.createMileageCContactC(contactID));
```

Now that you've set up the mileage record, call the `create` method on the Web service to persist the record to Salesforce. Note that `create` takes a list of objects, allowing you to add a batch of mileage records. [Listing 7](#) shows the code to persist the data to Salesforce.

Listing 7. Saving a new mileage record to the database

```
// call the create method passing the array of tasks as Subjects
ArrayList list = new ArrayList();
list.add(mileageC);
List<SaveResult> saveResults = login.port.create(list);
for (SaveResult saveResult : saveResults) {
    if (saveResult.isSuccess()) {
        System.out.println("saveResult success, id= " + saveResult.getId());
    } else {
        System.out.println("saveResult error");
        // there were errors during the create call, go through the
        // errors and write them to the screen
        List<Error> errorList = saveResult.getErrors();
        for (Error error : errorList) {
            System.out.println("Error code is: "
                + error.getStatusCode().toString());
            System.out
                .println("Error message: " + error.getMessage() + "\n\n");
        }
    }
}
```

This code persists the record to Salesforce and returns a list of the save results, which shows whether the save was a success or a failure. The error messages that come back from Salesforce tend to be fairly descriptive, making it easy to figure out what went wrong. This simplistic approach does not ensure the data is properly saved to Salesforce, however. For critical data, consider automatically retrying to save the data until it is successfully saved.

In a similar manner, you can update data in Salesforce. The easiest way is to first query for the record you want to change, then load the record into an object. You can update the object, then call the `update` method.

The other option for updating data is to merge duplicate records. With a merge operation, you specify a master record along with the record to merge. In this way, you update the master record with all the data out of the merge record. You can merge up to three records at a time.

Conclusion

By leveraging the power of XML, you can easily integrate cloud data into existing enterprise applications. XML provides a common data format that can be exchanged between a variety of services and languages. The sample application in this article demonstrated how you can interact with Salesforce through XML Web services using SOAP specifically.

This technique of using Web services to integrate on-site applications with cloud data can also be used with a wide variety of other applications, ranging from Google Apps to Basecamp. By integrating with cloud services, organizations can quickly build a variety of new applications in the cloud and still leverage their existing investment in software.

Resources

Learn

- [Force.com Fundamentals](#): Get more information on how Force.com works with the cloud.
- [Force.com workbook](#): Learn more about using the Force.com platform to build your own on-demand application with the Force.com workbook.
- [JAX-WS](#): Learn more about JAX-WS which defines APIs and conventions that support XML-based remote procedure call (RPC) protocols in the Java platform.
- [JAX-WS User Guide](#): Learn what you need to use the JAX-WS.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [Technology bookstore](#): Browse for books on these and other technical topics.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

Get products and technologies

- [Mileage source code](#): Download the Force.com workbook Mileage tutorial source code.
- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

About the author

Ryan Knight

Ryan Knight is a Senior Technical Consultant with more than 12 years of experience helping companies leverage the latest technologies. He has worked with companies both large and small, such as Oracle, IBM, and Williams Pipeline, and leveraged technologies ranging from cloud computing to enterprise architecture. You can read more about his extensive experience with Adobe Flex and Java technology on his blog at anvilflex.com.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.