accusoft®

Communicating with External Systems in a Salesforce Application

Stephen Bucholtz
Senior Software Engineer II

Salesforce applications provide a way to easily extend the capabilities of the popular customer relationship management platform Salesforce. A Salesforce user can choose from thousands of apps that are available on the Salesforce App Exchange.

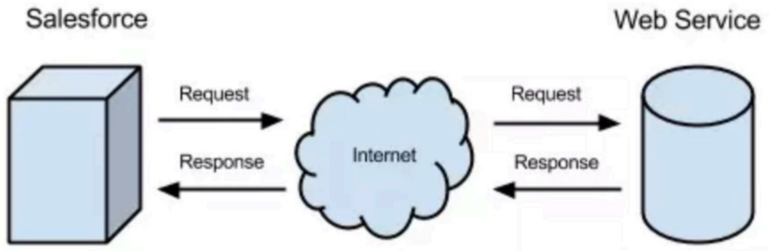
One common capability provided by Salesforce apps is the ability to communicate with external systems in order to extend the functionality of Salesforce. For example, the OnTask Salesforce app provides the ability to launch and monitor OnTask workflows directly in Salesforce.

As a Salesforce app developer, how does one incorporate communication with external systems into an app? Continue reading as we provide a brief introduction to this topic.

Calling REST API Endpoints via HTTP

Web-based services frequently expose their services via a REST API. HTTP is typically the protocol used to access these services.

Salesforce apps are developed using Apex, a strongly typed, object-oriented programming language. Apex allows integration with external REST Web services using callouts. Callouts allow communication with external systems via HTTP. Each callout request is associated with an HTTP method and an endpoint.



In order to develop a callout, several different Apex classes are used. These classes are described below.

HTTP Class

This class is used to initiate an HTTP message exchange and obtain the response. The method *send ( )* is used to send the message.

HTTPRequest Class

This class is used to create an HTTP request that can then be sent using the HTTP class. Various attributes of the HTTP request can be set, such as the method, endpoint, and header fields.

HTTPResponse Class

This class represents the response received from the HTTP class after a request has been sent. The class has methods for getting various parts of the response, such as *getStatusCode ( )* and *getBody ( )*.

Example

The following example involves calling an OnTask REST API endpoint. This endpoint retrieves the start form associated with a workflow model. The start form allows the user to provide input data when starting a workflow.

In this example, two functions are defined:

- *sendGetWorkflowStartForm ( )* - encapsulates the lowest level processing, which consists of constructing the HTTP request and sending the request

- *getWorkflowStartForm ( )* - calls the lower-level function and processes the response

```
public static HttpResponse sendGetWorkflowStartForm(String wfModelId) {
        // Create an instance of the Http class
        Http http = new Http();

        // Create an instance of the HttpRequest class
        HttpRequest req = new HttpRequest();

        // Set the HttpRequest method to GET
        req.setMethod(‘GET’);

        // Set the API Endpoint to be called
        req.setEndpoint(baseOnTaskApiUrl + ‘/workflowModels/’ +
            wfModelId + ‘/startEventForm’);

        // Set the HttpRequest header Content-Type to indicate
        // the returned data should be JSON content
        req.setHeader(‘Content-Type’,’application/json’);

        // Send the request and store the response in an instance
        // of the HttpResponse class
        HttpResponse res = http.send(req);
        return res;
    }

public static String getWorkflowStartForm(String wfModelId) {
        String baseErrorMsg = ‘There was an error getting the start
                form. wfModelId: ‘ + wfModelId;
         try {
                // Call the function that sends the HTTP Request
                HttpResponse res = sendGetWorkflowStartForm(wfModelId);

                // Check the status code contained in the HTTP Response
                if (res.getStatusCode() == 404) {
                   // A status code of 404 indicates there is no start form.
                   // In this case, return null.
                   return null;
                }
                else if(res.getStatusCode() == 200) {
                // Return the data contained in the HTTP Response Body.
                return res.getBody();
                } else {
                   String errorMsg = ‘OnTask returned statusCode: ‘ +
                        res.getStatusCode() + ‘, statusMessage: ‘ +
                        res.getStatus();
                   throw new OnTaskCallerException(errorMsg);
                }
        } catch (Exception e) {
                String errorMsg = e.getMessage();
                throw new OnTaskCallerException(errorMsg);
        }
 }
```

Handling JSON Data

REST API endpoints typically transmit data in request and response bodies as JSON data. Apex provides classes for generating and parsing JSON data:

- JSONGenerator - Generates standard JSON-encoded content and allows construction of JSON content, element by element.
- JSONParser - Parses JSON-encoded content and enables parsing a JSON-formatted response that's returned from a call to an external service.

JSONGenerator

The following is a simple example that shows how to generate JSON content.

```
// Create a JSONGenerator object.
// Pass true to the constructor for pretty print formatting.
JSONGenerator gen = JSON.createGenerator(true);

// Generate the tag that signifies the start of a JSON object
gen.writeStartObject();

// Write some JSON object fields
gen.writeStringField(‘firstName’, ‘Jack’);
gen.writeStringField(‘firstName’, ‘Smith’);
gen.writeStringField(title, CEO);

// Generate the tag that signifies the end of a JSON object
gen.writeEndObject();

// Retrieve a string representation of the JSON object.
// This string can then be assigned to the body of an HTTP message.
String jsonBody = gen.getAsString();
//Create a request and set the request body to the JSON string.
HttpRequest req = new HttpRequest();
req.setMethod(‘POST’);
req.setBody(jsonBody);
```

One Possible Gotcha

When making a callout to an external system, there are often related Salesforce database updates. It is common for the results obtained in a callout to be persisted with a database update. Conversely, a database item may be updated, then that updated data is transmitted to an external system. It is very important to note that there is a Salesforce restriction on the timing of callouts and database transactions:

- Callouts are not allowed when there is an uncommitted database transaction pending.

If the above rule is violated, you will encounter this error:

- "You have uncommitted work pending. Please commit or rollback before calling out."

You will encounter this error if you have database updates interspersed with callouts. In this scenario, all the database operations should be invoked only after you're done with the callouts. If you are making multiple callouts, then save all the database requests in a list or map, then after all the callouts have been completed, you can execute the saved database requests.

Conclusion and Further Reading

This article has provided a brief introduction to communicating with external systems from within a Salesforce app. We have seen that it is easy and straightforward to send REST API requests and process the responses. There is support for both constructing and parsing JSON data.

Hopefully this gives you a starting point for incorporating these concepts into your apps.

For additional general information on integrating with external systems, see the **Integration and Apex** Utilities section of the Salesforce Apex Developer Guide.

For additional information on handling JSON content, see the **JSON Support** section of the Salesforce Apex Developer Guide.

For an article on callout errors due to database transactions, see this **knowledge base article**.