

Salesforce

January 3, 2017

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | Salesforce |
| **Program Name** |  |
| **Author** | P M Arjun |
| **Version** |  |
| **Status** |  |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 1.0 | 15/12/2016 | Initial | P M Arjun |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

|  |  |  |  |
| --- | --- | --- | --- |
| **Team** | Name | Role | Contact (Email / Phone) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

1. Introduction…………………………………………………………………………4
2. Prerequisites ……………………………………………………………………….4
3. Installation…………………………………………………………………………...4
4. Use Cases……………………………………………………………………………..5
5. Salesforce Outbound Connector…………………………………………..5
6. Upsert Operation……………………………………………………………….…8
7. Query Operation………………………………………………………………..…9
8. Salesforce Batch Processing Outbound………………………………..10
9. Salesforce Inbound Connector……………………………………………..12
10. Salesforce Batch Processing Inbound Using Poll……………….….13
11. Exception Handling……………………………………………………………...14

**Introduction:**

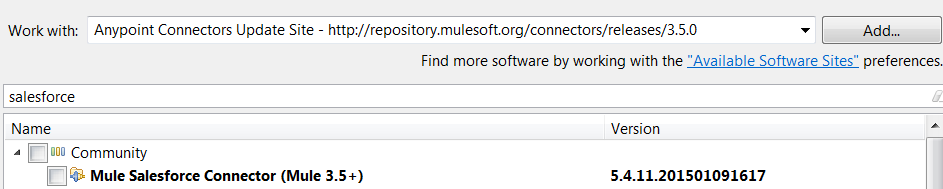
This document is intended to illustrate the basic setup and implementation of Salesforce Inbound, Streaming and Outbound connectors across all the integration projects. Basically Salesforce connector provides you a connection with the Salesforce platform. It also provides various methods or operations that allow us to exploit the capabilities of the Salesforce platform.

**Prerequisites:**

1. An instance of Anypoint studio.
2. A Salesforce developers account- You can sign up using the link : <https://developer.salesforce.com/>
3. You need a security token, as you would be accessing into Salesforce using the Mule ESB. Regular sign in wouldn’t require a token. In order to get a security token- login to developer’s account-> click on your name on the upper right hand corner-> click My Profile -> on the left hand panel click on Reset My security Token-> click on reset my Security Token  you will get your new token on your registered email.
4. Consumer key and secret- We need it when using the Salesforce connector to access an **OAuth API.**

**Installation:**

To install Salesforce connectors please follow the below steps:

1. Go into Anypoint studio-> click on help on the task bar-> click on Install new Software’s.
2. A windows pop’s up-> click the drop down and select-> Anypoint Connectors Update Site - <http://repository.mulesoft.org/connectors/releases/3.5.0>
3. Filter it by typing Salesforce. 

Check the checkbox Mule Salesforce Connector-> click next and finish the setup. Restart the mule studio.

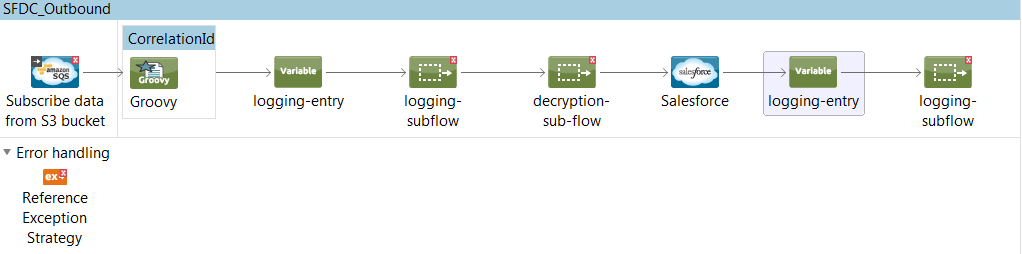
**Use Cases:**

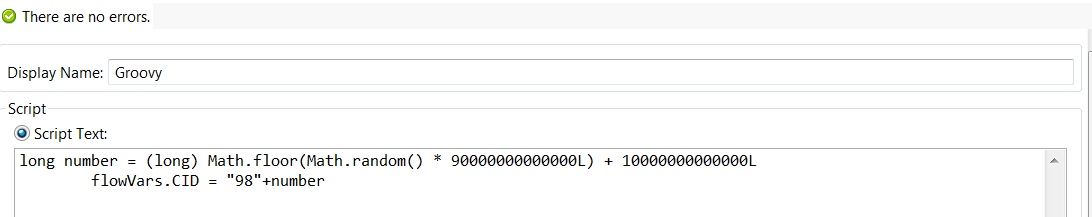
1. Salesforce connector provides us various operations one being the upsert operation, where it creates new records and updates the existing records from the payload that we are pushing into the Salesforce by using an External Id Field.
2. Receiving events from the Salesforce for any kind of changes in Salesforce by creating Topics and subscribing them.
3. Using the Salesforce Query Language, we can write queries and receive events for any changes that are done to the Salesforce data.
4. It also provides message durability where you can retrieve the stored events during a retention period of 24 hours.
5. The Salesforce Bulk API is based is optimized for loading or deleting large sets of data. It allows you to query, insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce.

**Salesforce Outbound Connector:**

Use Salesforce outbound connector in order to push the data into Salesforce. We can also use it in a batch process to push the data.

Consider the following flow:

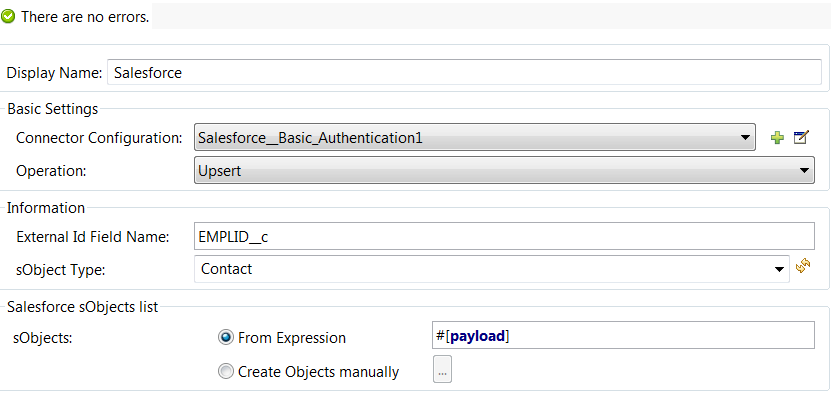


In the above flow the inbound data is received from a S3 bucket.Once we receive the inbound message from S3 bucket we first generate a Correlation ID, this should be a unique ID that is generated by Mulesoft CloudHub or Mule On-prem. It is of 14 digit random number with ‘98’ as first 2 characters. It is generated using a Groovy Component and added as flow variable to the inbound message using the message enricher component.

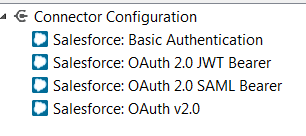
Then we are creating a logging entry variable for logging purposes. Then we log the sub-flow if there are any and decrypt the sub-flow if it is necessary. Then we push the data to the outbound Salesforce connector.

There are various configurations that are required for the connector to work as expected, those are:

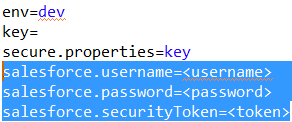
**Connector Configurations**:



For configuring Salesforce connector we must first provide the type of authentication that we are going to use. There are four types of authentication that can be specified:



**Basic Authentication** is the simplest way of authentication. You just need to soft code the credentials in mule-app.properties.

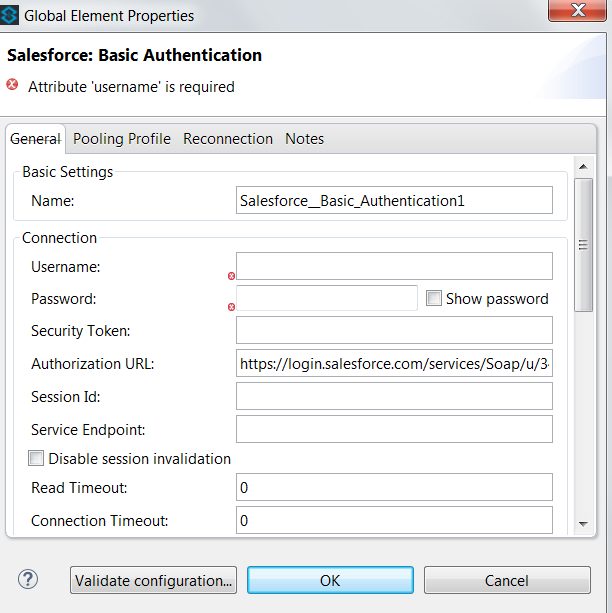


**OAuth v2.0**: The web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the web server flow is that the server must be able to protect the consumer secret.

**OAuth v2.0 JWT Bearer:** JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.

**OAuth v2.0 SAML Bearer:** A SAML assertion is an XML security token, generally issued by an identity provider and consumed by a service provider who relies on its content to identify the assertion’s subject for security-related purposes.

In our flow we are using the Basic Authentication, as soon as we select that option a new window pop’s up:

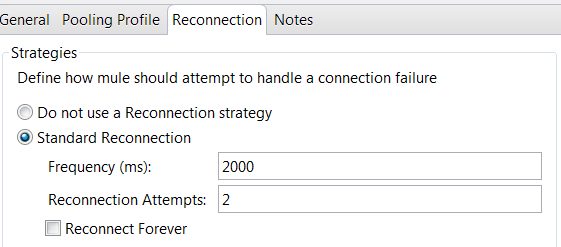


Here we need to provide username, password and the security token and Validate the configuration by clicking on validate configuration button. If the credentials are correct then you would get **Test Connection Successful message.**

Now in the same window you can also provide a Reconnection Strategy:

**Reconnection Strategy**:

Reconnection strategy is used to specify what a connector should do when a connection attempt fails.



So when we opt for a reconnection strategy we can specify the frequency with which connector should make an attempt to reconnect, In our flow it is every 2000ms that the connector would try to reconnect and it would do it for two times.

You can also set it as reconnect forever where it would try to reconnect until the connection is successful.

<sfdc:config name=*"Salesforce\_\_Basic\_Authentication"* username=*""* password=*""* doc:name=*"Salesforce: Basic Authentication"*>

<reconnect-forever frequency=*"1000"*/>

**Upsert Operation :**

Once the basic Authentication is done then we need to specify the operation that we are going to perform. There are many operations that can be performed starting from upsert, create, retrieve, query, merge, delete etc. In our flow we are doing an **Upsert.**

**Upsert:** Creates new records and updates existing records; uses a custom field to determine the presence of existing records. In most cases, we recommend that you use upsert () instead of create () to avoid creating unwanted duplicate records (idempotent).

So the data would be pushed into Salesforce as objects i.e sObjects.

So in our flow object would be pushed into Salesforce depending on an External Id EMPLID\_C where \_C tells that it is a Custom object.

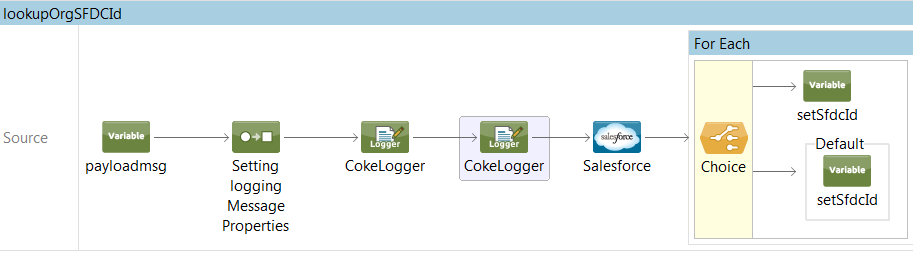
And the object type that the data would be pushed is of type ‘Contact’.

Once that is done it would log the payload that was upserted to the Salesforce and refer to a flow that would log the sub-flow.

**Query Operation:**

Query operation is basically used for retrieving data from the Salesforce. Salesforce query language (SOQL) is used to write a query and retrieve the data from Salesforce that matches the query.

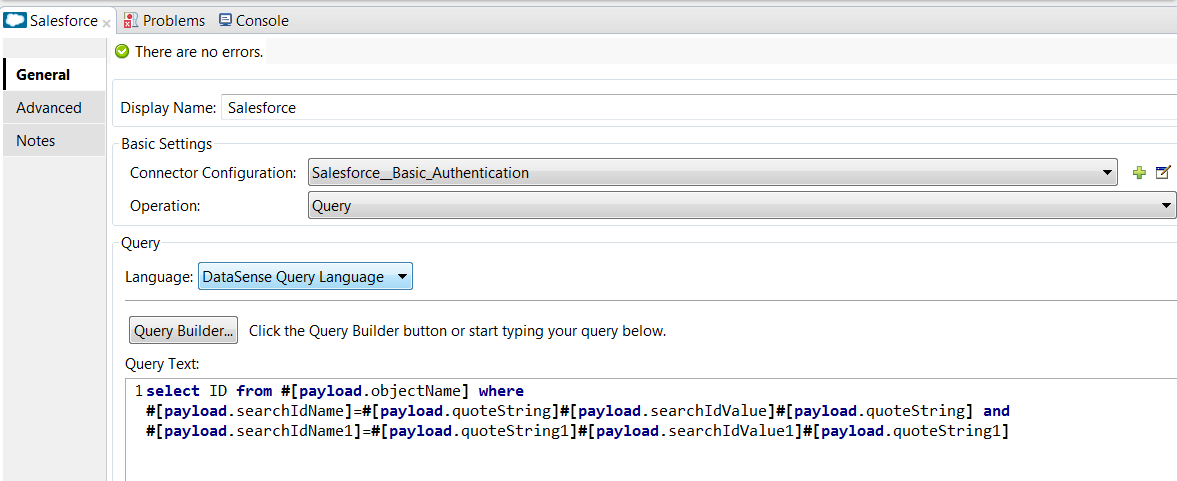
In order to understand the query operation let us consider another flow:



So in the above flow we are setting a variable payloadmsg with the value extracted from the payload, in this case we are extracting an ID.

Then we are setting the logging properties and logging the extracted ID using CokeLogger before the SFDC searches for that ID from its database.

In order to have queried a data, Salesforce must have the following configuration.



Basic Authentication is used and operation **QUERY** is selected.

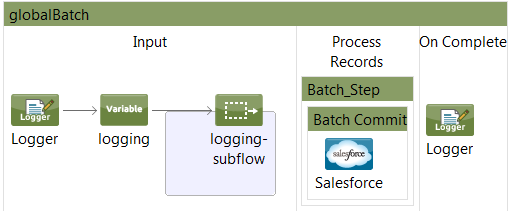
In the query text the query is written.

This Query would fetch an ID from Salesforce.Using choice component I would then compare the ID and set an sfdc ID variable respectively.

**Salesforce Batch Processing Outbound:**

The Salesforce Bulk API is based is optimized for loading or deleting large sets of data. It allows you to query, insert, update, upsert, or delete a large number of records asynchronously by submitting a number of batches which are processed in the background by Salesforce.

Let’s consider a flow:



Now in the above flow we are using a global batch, a batch-step and a batch commit in order to push the payload as individual records into the Salesforce.

So logger would first log the inbound payload and then we will set the logging –entry variable and log the sub-flow if any.

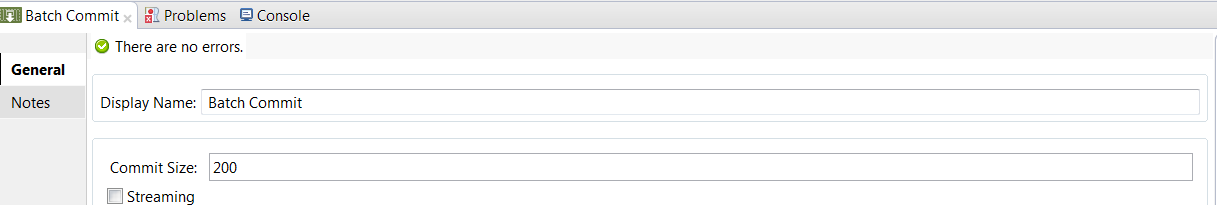
Next it goes into Batch\_step.



In Batch\_step we can provide a policy, only if that policy is satisfied it would go into Batch Commit.

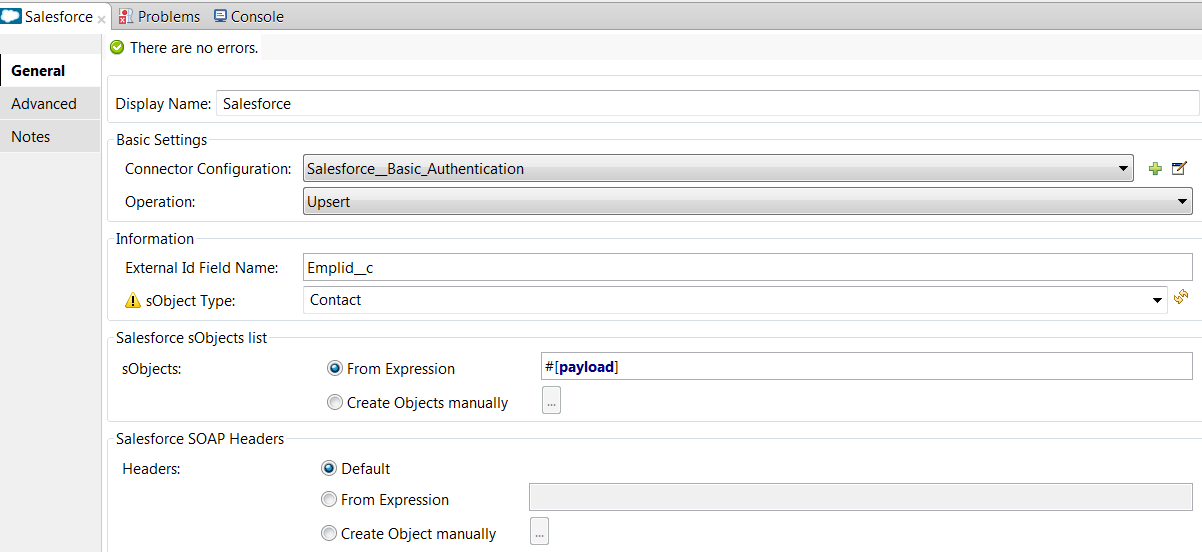
In our flow the policy is NO\_FAILURES, which means it would only process those records that has not failed in its preceding steps.

Once the policy is satisfied the records are pushed to Batch Commit:



In our flow each time it would push 200 records into the Salesforce each time, so that Salesforce can provide the specified operation on them. So each time it would push 200 records and other records that needs to be pushed will be in the persistent queue waiting to enter Batch Commit for processing.

Inside Batch Commit, Salesforce has the following configuration:



So it is following Basic Authentication and the operation is Upsert.

So it will perform upsert of 200 records at a time. It would perform an upsert on the basis of External ID and sObject type.

Finally the inserted SFDC data is logged.

**SALESFORCE INBOUND CONNECTOR:**

In the case of Salesforce inbound connector in order to receive any change in the events in Salesforce data we must create a PushTopic, and the inbound connector must subscribe that Topic.

Let us first create a PushTopic:

Sign into your Salesforce Account-> get into the Developer’s Console.

Click**Debug** |**Open Execute Anonymous Window.**

**In the Enter Apex Code Window, you could type in the below code, and click Execute.**

PushTopic pushTopic = new PushTopic();

pushTopic.Name = 'InvoiceStatementUpdates';

pushTopic.Query = 'SELECT Id, Name, Status\_\_c, Description\_\_c FROM Invoice\_Statement\_\_c';

pushTopic.ApiVersion = 38.0;

pushTopic.NotifyForOperationCreate = true;

pushTopic.NotifyForOperationUpdate = true;

pushTopic.NotifyForOperationUndelete = true;

pushTopic.NotifyForOperationDelete = true;

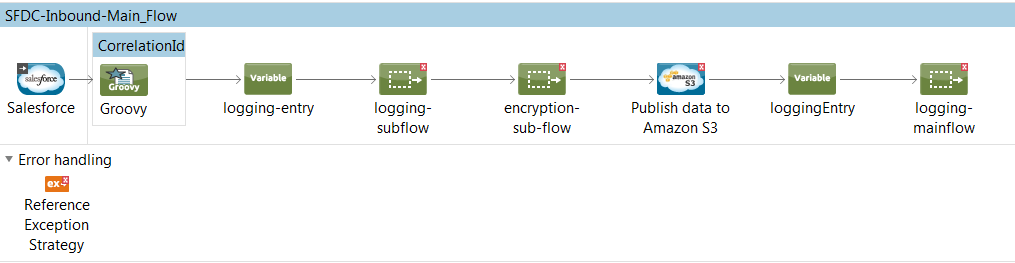
pushTopic.NotifyForFields = 'Referenced';

insert pushTopic;

Using the above code we are creating a new topic named InvoiceStatementUpdates.

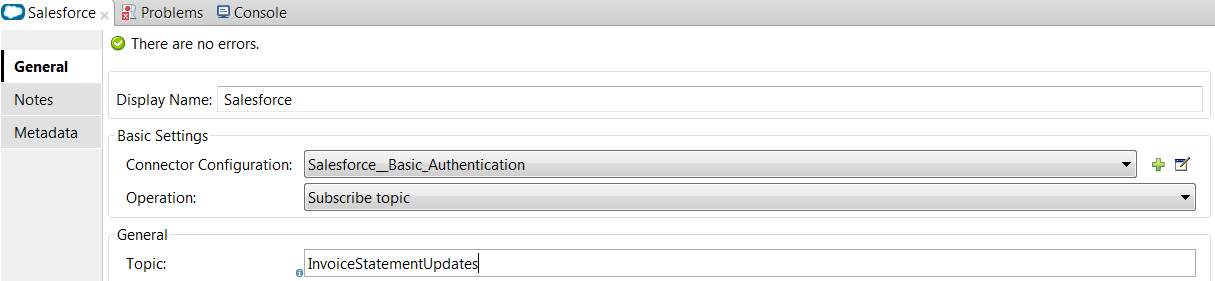
So whenever there is a change in events in Id, Name, Status\_c objects from the namespace Invoive\_statement\_c for the operations Create, Update,, Delete and Undelete a notification will be generated on this channel and it would notify the subscribed Salesforce inbound connector.

In order to understand better let’s consider a flow:



So in the above flow the inbound connector is the Salesforce connector and outbound connector is the Amazon S3 bucket.

Let us first look at the inbound Salesforce connector configuration:



It is using Basic Authentication by soft coding the credentials.

The operation is subscribing a topic.

The name of the topic that it is subscribing is InvoiceStatementUpdates.

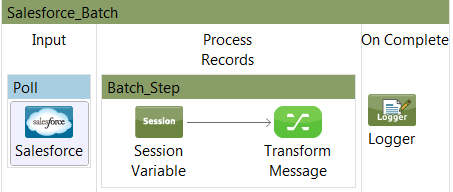
So whenever there is a Create, Update, Delete and Undelete on Id, Name, Status\_c objects from the namespace Invoive\_statement\_c a notification would be created in the channel and it would notify the SFDC connector.

The payload received from the Salesforce would then go into the message enricher where the groovy component is creating a correlation Id and including it as flow variable in the inbound payload.

A logging-entry variable is created and if there are sub flows they are logged and the payload is encrypted if it is needed.

**Salesforce Batch Processing Inbound Using Poll:**

Let us consider a flow:



In the above flow poll scope would poll the Salesforce inbound on regular intervals for any data, In case the operation is Query then depending on the Query it would match the records send it over as inbound payload.

So the Salesforce receives the payload and it prepares for message processing.

Mule then creates a batch job instance and turns a serialized message into collection of records.

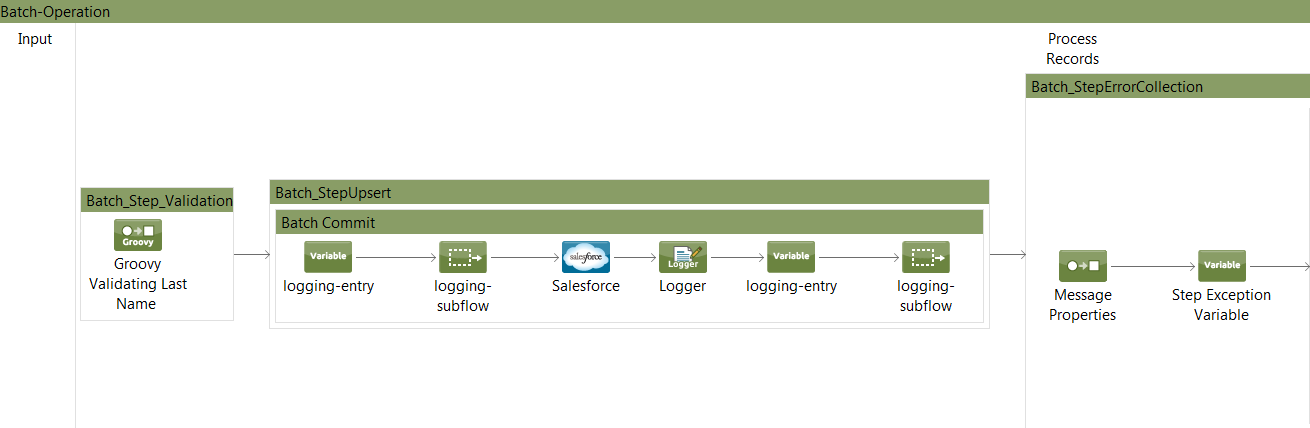
Records go into Batch-step, asynchronous processing of the records take place in the batch.

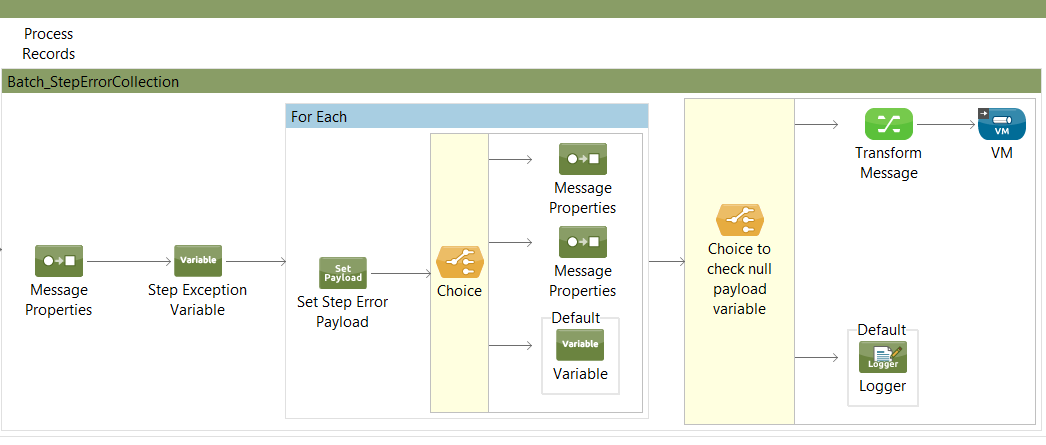
If there are one or more batch-steps then the processed records would go into a Queue waiting for the processing of the next Batch-step.

Once all the records are processed the logger would display the payload as individual records.

**Handling Soft Error’s:**

Consider the below flow, both the screenshots combine a single batch process.

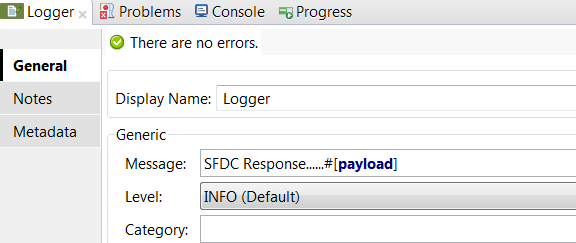




Now the batch process starts from the batch step ‘**Batch\_Step\_Validation**’, where a groovy component is validating a field called ‘**Last\_Name**’ in the payload.

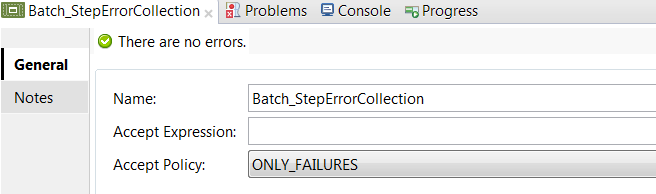
Next the flow goes into the next batch step ‘**Batch\_StepUpsert**’, where we are logging the flow and using the SFDC connector we are performing an upsert operation for the records.

Once the batch upsert is done, SFDC would send the result of the operation. The result contains how many records failed the upsert operation, an error message with the cause for the failure, records that were processed, the records that are loaded etc. many such information would sent back.

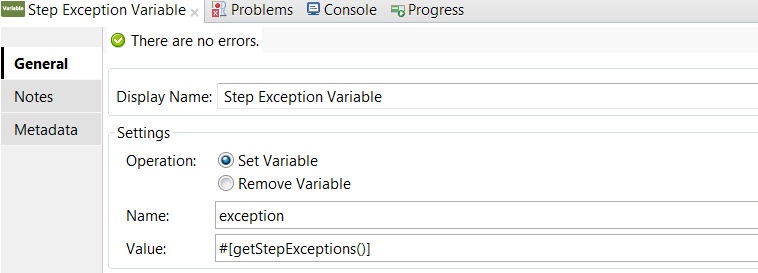


So the logger just after the SFDC component would log the result.

Now the next batch step ‘**Batch\_StepErrorCollection’** is used to gather all the records that has failed in its previous batch steps. Since the goal is to gather failures it makes sense to skip the ones who haven’t got any, thus we set the accept policy failures only.



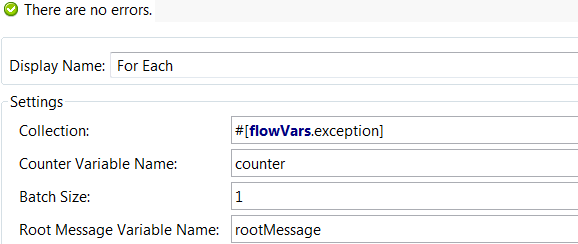
So we first create a variable named ‘**exception**’:



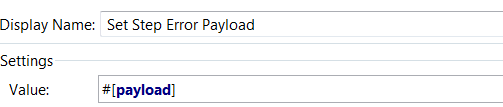
**getStepException**: Returns a java Map<String, Exception> in which the keys are the name of a batch step in which the current record has failed and the value is the exception itself. If the record hasn’t failed in any step, this Map will be empty but will never be null. Also, the Map contains no entries for steps in which the record hasn’t failed.

So using the above MEL function the variable is set to an exceptions map. This exception map gives you access to all the exceptions in all its previous steps.

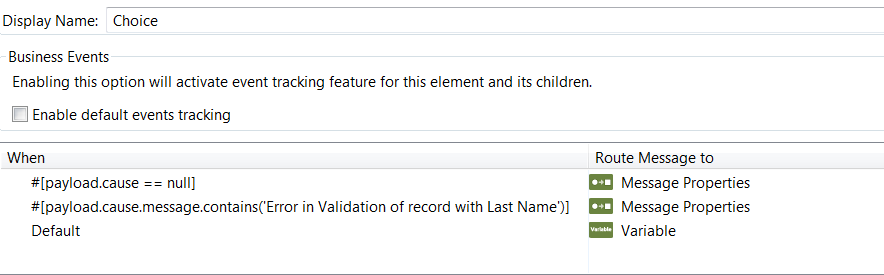
Next we use a for each to handle each exception individually.



So we would iterate over all the map values. Inside for each we would set the payload, which would be the individual map value.



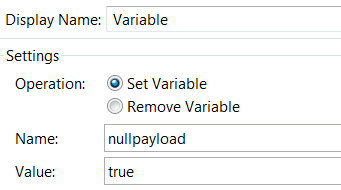
So Choice component would check the cause for the exception, if it is ‘**null**’ which normally it never happens because each exception map will have valid exception or would be empty, never null. Whether the failure has been caused in the ‘**Batch\_Step\_Validation**’ step or in the ‘**Batch\_StepUpsert**’ step.



So the 1st condition it rarely happens. The second condition happens when the record underwent an error during the **Validation on last name** step, i.e. is the first step in the batch process.

The third happens whenever there is an error during the upsert operation.

If the third condition is satisfied it goes to the variable ‘**nullpayload**’ where the value is set to true so that the condition is true for the next choice component and the record that has been failed because of the upsert operation are handled by placing them in a VM queue.



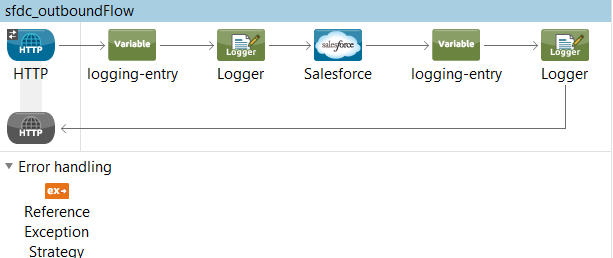
If the record has gone through an exception in upsert operation, it would go to the next choice component. If the value is true, it would do a transformation to message and store it in the VM queue.

These records can be later retrieved using the Mule requestor.

**Exception Handling:**

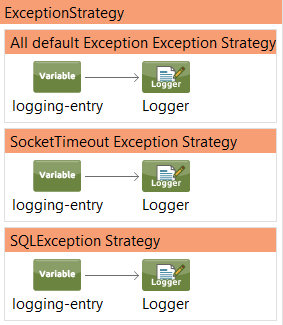
During the entire flow various error’s or exceptions could arise, even exceptions could arise due to Salesforce connector as well. For example if the login credentials are not correct **NoAccessException** could happen or if there is a connectivity issue, that could raise an exception.

So in our integration projects exceptions are handled by logging them. Each time an exception takes place they are logged.



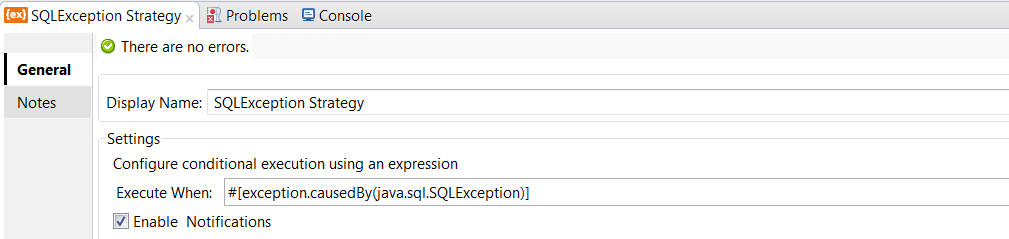
In the above flow we are using Reference exception strategy in order to refer and adhere to the error handling parameters defined in a global choice exception strategy.

In case any kind of exception occur Reference exception strategy would refer a global Choice Exception strategy.



So the global Choice exception strategy would have multiple exception strategies for different exceptions that could occur across different flows.

Now suppose we are using an outbound Salesforce connector with a Query operation, then in case the query is not correct, and returns no records then SQL Exception could take place, where the flow would call the Reference Exception strategy.



As it is a Query exception the SQLException strategy would be chosen and the exception would be logged with ‘ErrorCode’, ‘ErrorType’ and ‘ErrorText’.

Similarly there are various exceptions that could take place because of the Salesforce connector some of which are:

| **Exception** | **Description** |
| --- | --- |
| AsyncException | Any problem with an asynchronous operation, such as failing to enqueue an asynchronous call. |
| CalloutException | Any problem with a Web service operation, such as failing to make a callout to an external system. |
| DmlException | Any problem with a DML statement, such as an insert statement missing a required field on a record. |
| EmailException | Any problem with email, such as failure to deliver. |
| ExternalObjectException | Any problem with external object records, such as connection timeouts during attempts to access the data that’s stored on external systems. |
| InvalidParameterValueException | An invalid parameter was supplied for a method or any problem with a URL used with Visualforce pages. |
| LimitException | A governor limit has been exceeded. This exception can’t be caught. |
| JSONException | Any problem with JSON serialization and deserialization operations. For more information, see the methods of [System.JSON](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_class_System_Json.htm" \l "apex_class_System_Json" \o "Contains methods for serializing Apex objects into JSON format and deserializing JSON content that was serialized using the serialize method in this class.), [System.JSONParser](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_class_System_JsonParser.htm" \l "apex_class_System_JsonParser" \o "Represents a parser for JSON-encoded content.), and[System.JSONGenerator](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_class_System_JsonGenerator.htm#apex_class_System_JsonGenerator). |
| ListException | Any problem with a list, such as attempting to access an index that is out of bounds. |
| MathException | Any problem with a mathematical operation, such as dividing by zero. |
| NoAccessException | Any problem with unauthorized access, such as trying to access an sObject that the current user does not have access to. This is generally used with Visualforce pages. |
| NoDataFoundException | Any problem with data that does not exist, such as trying to access an sObject that has been deleted. This is generally used with Visualforce pages. |
| NoSuchElementException | This exception is thrown if you try to access items that are outside the bounds of a list. This exception is used by the [Iterator](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_iterable.htm#apex_classes_iterable) next method. For example, ifiterator.hasNext() == false and you call iterator.next(), this exception is thrown. This exception is also used by the Apex Flex Queue methods and is thrown if you attempt to access a job at an invalid position in the flex queue. |
| NullPointerException | Any problem with dereferencing null, such as in the following code:   |  |  |  | | --- | --- | --- | | 1 | | String s; | | 2 | s.toLowerCase(); // Since s is null, this call causes | | |  |  |  | | --- | --- | | 3 | // a NullPointerException | |
| QueryException | Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable. |
| RequiredFeatureMissing | A Chatter feature is required for code that has been deployed to an organization that does not have Chatter enabled. |
| SearchException | Any problem with SOSL queries executed with SOAP API search() call, for example, when the searchString parameter contains less than two characters. |
| SecurityException | Any problem with static methods in the Crypto utility class. |
| SerializationException | Any problem with the serialization of data. This is generally used with Visualforce pages. |
| SObjectException | Any problem with sObject records, such as attempting to change a field in an updatestatement that can only be changed during insert. |
| StringException | Any problem with Strings, such as a String that is exceeding your heap size. |
| TypeException | Any problem with type conversions, such as attempting to convert the String 'a' to an Integer using the valueOf method. |
| VisualforceException | Any problem with a Visualforce page. |
| XmlException | Any problem with the XmlStream classes, such as failing to read or write XML. |