

AWS (S3/SNS/SQS) Connector Template

January 2, 2017

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | AWS (S3/SNS/SQS) Connector Template |
| **Program Name** |  |
| **Author** | P M Arjun |
| **Version** |  |
| **Status** |  |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 1.0 | 16/12/2016 | Initial | P M Arjun |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

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

Contents

1. **Introduction……………………………..4**
2. **Prerequisites…………………………….4**
3. **SQS Connector………………………….4**
4. **Use Cases……………………..4**
5. **S3 Connector…………………………….9**
6. **Use Cases……………………..9**
7. **SNS Connector………………………….12**
8. **Use Cases……………………..12**
9. **Reconnection Strategy ……………..15**

**Introduction:**

There are three connectors that can be used to utilize the Web Services provided by Amazon. These are S3, SQS and SNS connectors that provides connectivity to the Amazon S3, SQS and SNS API’s respectively. This document illustrates on how to setup theses connectors so that we can exploit the Web Services provided by Amazon.

**Prerequisites:**

Now the prerequisites for utilizing all the three services are same, which are:

* Access to Amazon Web Services – SQS, SNS and S3.
* To access AWS with the connector, you need the credentials in the form of IAM.
* Anypoint Studio Enterprise edition.

**SQS Connector:**

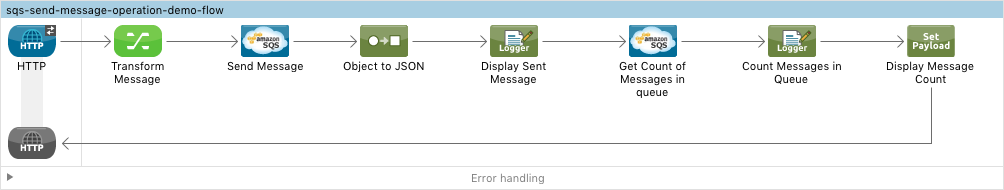
Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable hosted queue for storing messages as they travel between computers. By using Amazon SQS, developers can simply move data between distributed application components performing different tasks, without losing messages or requiring each component to be always available. Amazon SQS makes it easy to build an automated workflow, working in close conjunction with the Amazon Elastic Compute Cloud (Amazon EC2) and the other AWS infrastructure web services.

**Use Cases:**

Amazon SQS can perform a wide variety of operations, so we can add the connector to the flow and configure the connector for a specific operation, some of these operations are:

* Add Permission
* Change message visibility
* Change message visibility batch
* Create queue
* Delete message
* Delete message batch
* Delete queue
* Get approximate number of messages
* Get queue attributes
* Get queue URL
* List dead letter source queues
* List queues
* Purge Queue
* Receive Messages
* Remove permission
* Send message batch
* Send message
* Set Queue Attributes

In order to understand let’s consider a sample use case example:

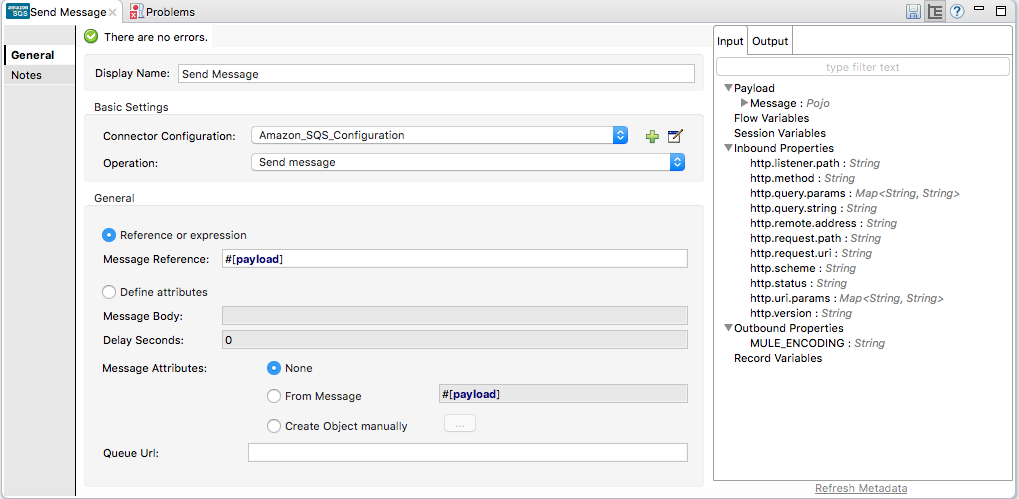


Let’s consider the above flow:

1. We are going to send the message to the SQS, then get the count of messages in SQS.

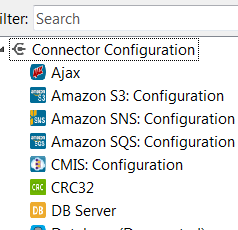
To start with HTTP connector is sending an inbound message, then the transform metadata component would add a metadata.

Then the flow enters the SQS connector, in order to send the message we have to configure the SQS connector in the following way:

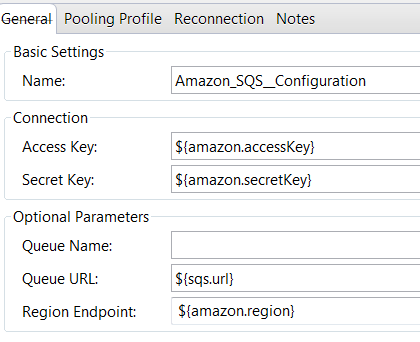


We need to configure the connector so that we can access AWS, send the message to correct queue by providing valid authentication, this can be done as shown below:

1. Go to **Global Elements** tab-> and click on create.
2. A window would pop up->expand connector configuration:



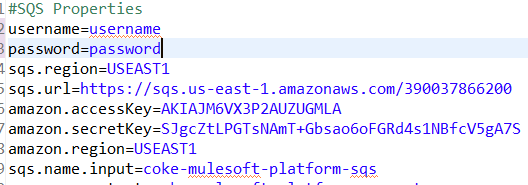
Select **Amazon SQS: Configuration** and a window pop’s up:



Provide the configurations as shown above and refer these fields from a soft coded property file.

Now Access Key and Secret key are user specific.

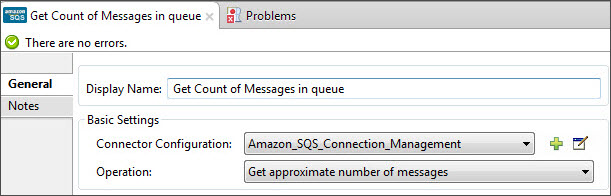
Now the credentials to access AWS, the region where is queue is located, the url pointing the queue,access key, secret key, the queue name are soft coded in a properties file.



As we are sending a message operation **sending message** is selected and message reference **#[payload]** so that it sends the payload.

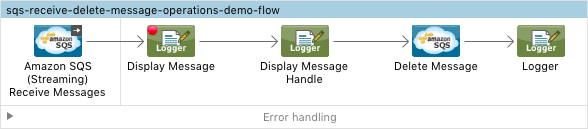
Then we convert the code from object to json and then we log the payload.

Next we try to fetch number of messages in the queue we provide the following configuration for SQS connector:

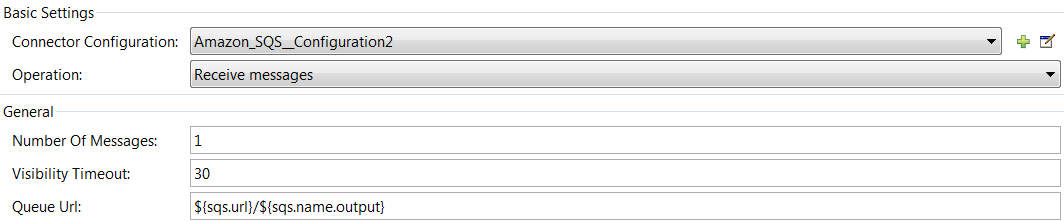


We will log to get the see the number of messages that were queued.

Now whatever message that has been sent to the queue, we will try to retrieve it back using SQS connector. Consider the below flow:

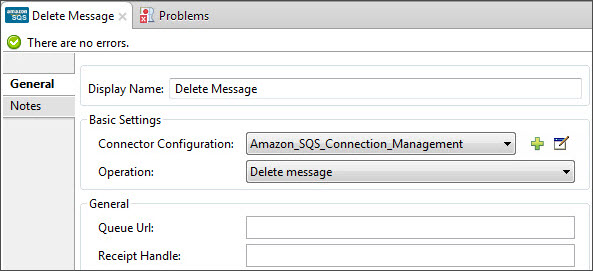


The inbound will be the Amazon SQS connector and it should have the following configuration:



As we are refering from the same queue, the credentials, url and other properties would be same.

We would display the message using logger and delete the messages from the queue using the SQS connector and it would have the following configuration:



**S3 Connector:**

Amazon Simple Storage Service (Amazon S3) is storage for the Internet. You can use Amazon S3 to store and retrieve any amount of data at any time, from anywhere on the web. You can accomplish these tasks using the simple and intuitive web interface of the AWS Management Console.

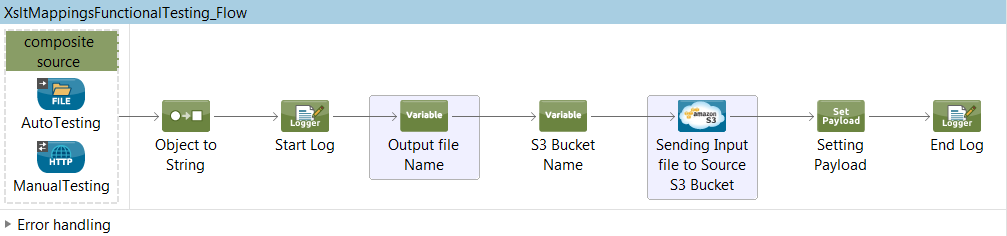
**Use Cases:**

Amazon S3 connector is an operation-based connector, which means that when you add the connector to your flow, you need to configure a specific operation for the connector to perform. S3 supports a whole variety of operations some of which are:

* Abort Multipart Upload
* Complete Multipart Upload
* Copy Object
* Create Bucket
* Create Object
* Create Object Presigned URI
* Delete Bucket
* Delete Bucket Cross Origin Configuration
* Delete Bucket Lifecycle Configuration
* Delete Bucket Policy
* Delete Bucket Tagging Configuration
* Delete Bucket Website Configuration
* Delete Object
* Delete Objects
* Get Bucket ACL
* Get Bucket Cross Origin Configuration

Let us consider a flow in understanding how S3 actually works. In this use case example we will receive an inbound message and send it to the S3 bucket.

Consider the below flow:



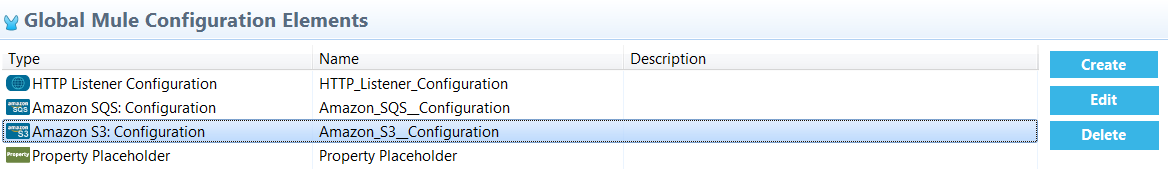
Now in the above flow, you might be receiving inbound message from dual sources from a FILE component and a HTTP inbound. It is not mandatory to use dual inbounds if you are using S3 connector. It has just been used in this particular flow.

The message that is received is converted from object to string and is logged.

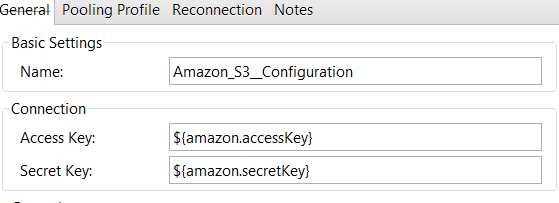
Now from the next component the configuration for sending the message to a S3 bucket starts.

Similar to what we did in SQS, we need to perform the same steps for configuration.

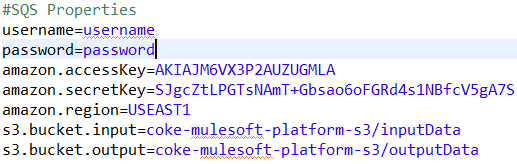
Create a global configuration for S3 bucket:



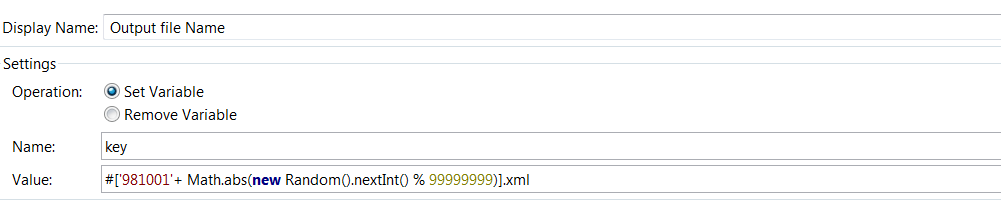
With following configuration:



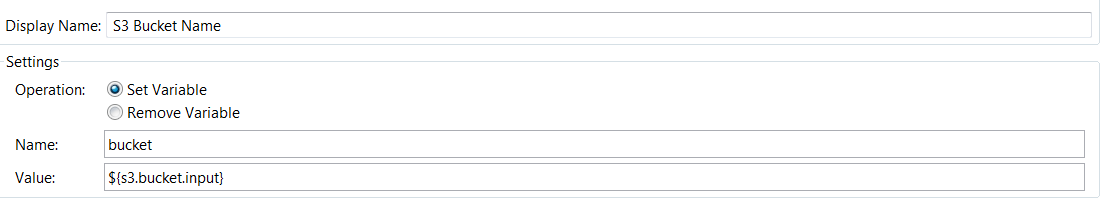
We need to soft code the AWS access credentials, access key, secret name, bucket name in a properties file:



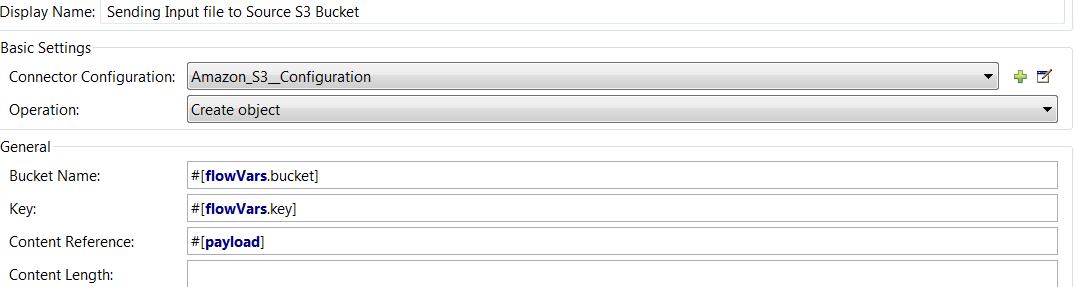
Next we define a variable which would become the name of the file that we are sending. Now this a random name that is chosen, where the value is set using Math.Random() method.



Next we are setting another variable for extracting the bucket name from the properties file:



Then the flow enters the S3 connector and it would have the following configuration:



So here the operation is **create object**, so it would send the the file in form of an object and store it in the bucket.

Bucket name and key are already flow variables and we are just referring the values in the S3 connector.

Content Reference is **#[payload]**, the payload that we are sending.

**SNS Connector:**

The Anypoint Amazon SNS Connector provides connectivity to the Amazon Simple Notification Service (Amazon SNS) API, enabling you to build distributed web-enabled applications. Mule applications can use the Amazon SNS Connector to easily push real-time notification messages to interested subscribers over multiple delivery protocols.

**USE CASES:**

Amazon SNS connector is an operation-based connector, which means that when you add the connector to your flow, you need to configure a specific operation for the connector to perform. Some of the operations supported by the connector are:

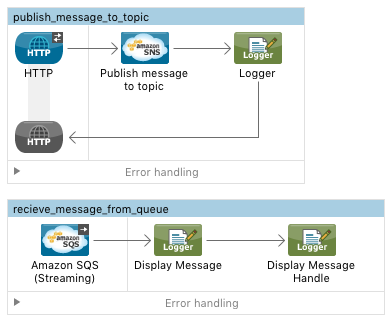
* Add Permission
* Confirm Subscription
* Create Platform Application
* Create Platform Endpoint
* Create Topic
* Delete Endpoint
* Delete Platform Application
* Delete Topic
* Get Endpoint Attributes
* Get Platform Application Attributes
* Get Subscription Attributes
* Get Topic Attributes
* List Endpoint By Platform Application
* List Platform Application
* List Subscriptions By Topic
* List Subscriptions
* List Topics
* Publish

So let us consider a use case example in order to understand the working of SNS Connector.

So basically when you **subscribe an Amazon SQS queue to an Amazon SNS topic**, you can publish a message to the topic and Amazon SNS sends an Amazon SQS message to the subscribed queue.

You can send messages to a topic where it would publish it to the subscribed queue, but SNS connector in turn won’t retrieve messages back to the flow from the queue.

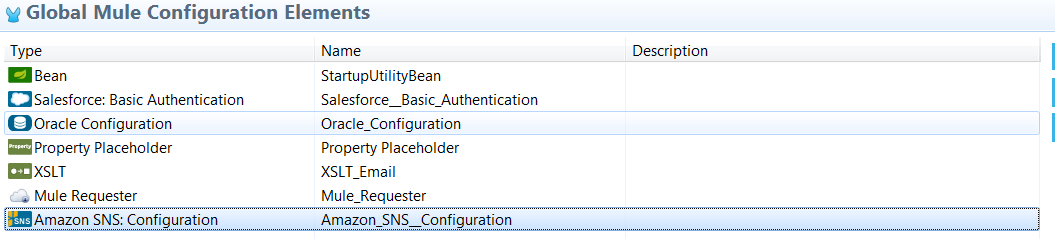
So let’s publish a message to a topic:



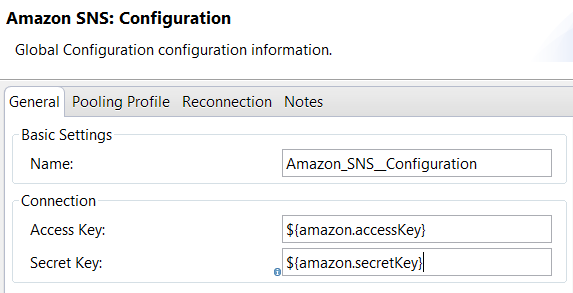
So in the above flow we are publishing a message from SNS connector to a topic and receiving that message back using the SQS connector.

So we receive an inbound message from HTTP connector and the flow goes into SNS connector.

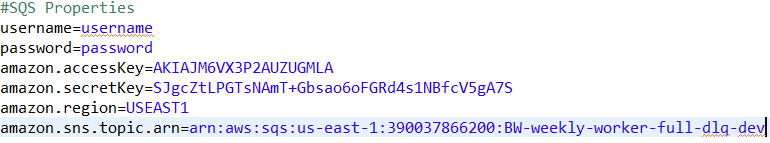
For SNS connector configuration we need to again follow the same steps, we need to create a global configuration element with following configuration:



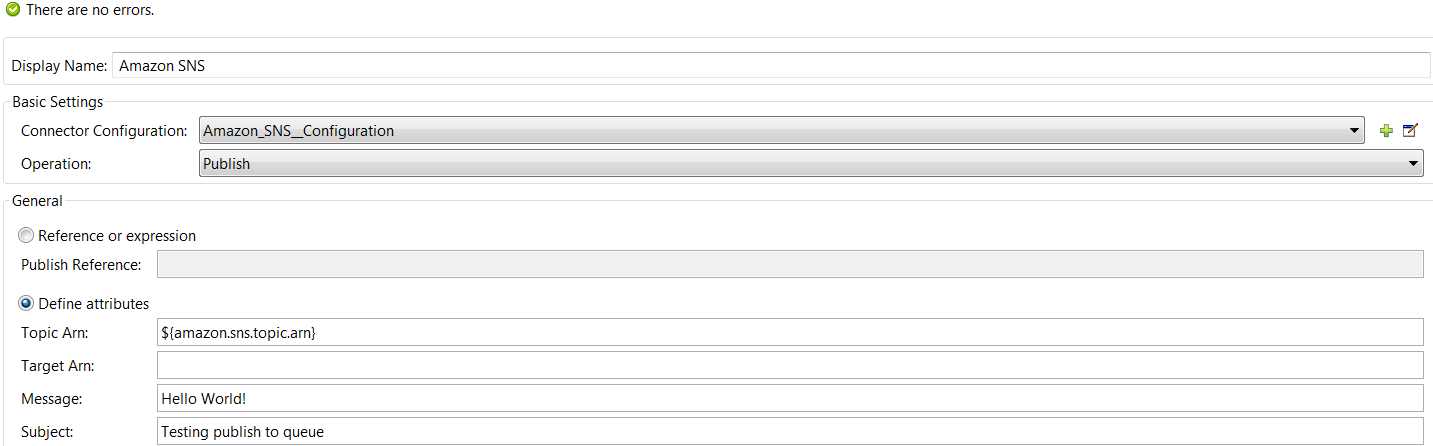
And the configuration would be following:



The AWS access credentials, secret key, access key, the topic arn are all soft coded in the properties file.



The S3 connector would have the following configuration:



So the operation would be Publish.

It would take the Topic Arn from the properties file.

And it publish a message **Hello World** with subject Testing publish to queue.

A specific SQS would have subscribed this particular topic. So topic would send the message to that queue.

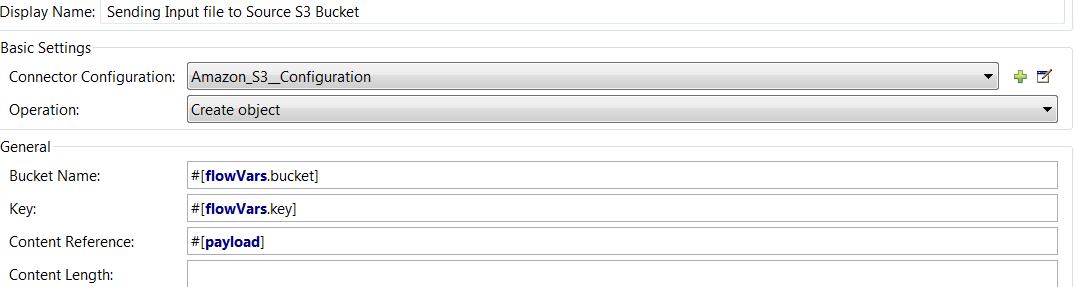
And by using the SQS connector in the second flow we can retrieve back the message.

**Reconnection Strategy:**

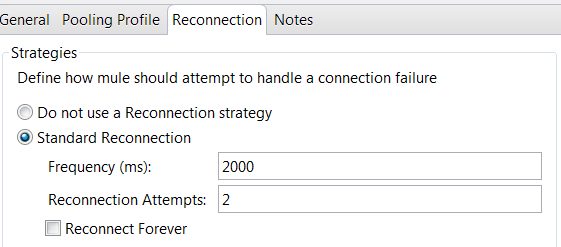
All the three connector’s S3, SQS and SNS provides us a reconnection strategy in case the attempt to build a connection to AWS web services fails.

If the connection fails using this reconnection strategy we try to rebuild the connection.

Now reconnection strategy can be configured as shown below:



When we click on the ‘+’ a window pops up, switch to Reconnection strategy tab:



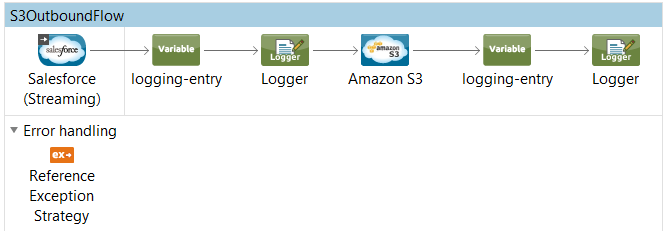
So when we opt for a reconnection strategy we can specify the frequency with which connector should make an attempt to reconnect. In our flow it is every 2000ms that the connector would try to reconnect and it would do it for two times.

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

**Exception Handling:**

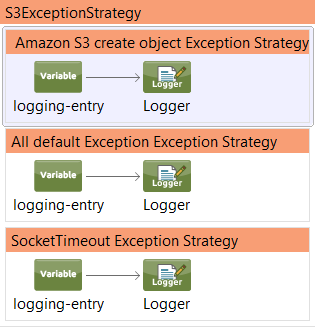
During the entire flow various error’s or exceptions could arise, even exceptions could arise due to AWS connector’s as well. For example if the login credentials are not correct **AccessDeniedException** could happen or if there is a connectivity issue, that could raise an exception.

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



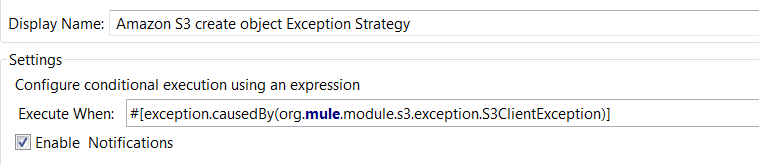
In the above flow we are using Reference exception strategy in order to refer and adhere to the error handling parameters defined in a global choice exception strategy.

In case any kind of exception occurs 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 S3 connector with a Create Object operation, then in case the specified S3 bucket does not exist or an error has occurred while storing the object or any exception that occurs when we try interact with Amazon Simple Storage Service, then the flow would call the Reference Exception strategy.



As it is an S3Client exception the Amazon S3 create object Exception 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 S3, SNS and SQS connector’s some of which are:

**S3 Exceptions:**

Below file contains all the S3 exceptions that could occur.

****

**SQS Exceptions:**

Some of the exceptions are:

|  |  |
| --- | --- |
| [SqsException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sqs.Exception.SqsException.html) | Default service exception class |

**SNS Exceptions:**

|  |  |
| --- | --- |
| [AuthorizationErrorException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.AuthorizationErrorException.html) | Indicates that the user has been denied access to the requested resource. |
| [EndpointDisabledException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.EndpointDisabledException.html) | Exception error indicating endpoint disabled. |
| [InternalErrorException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.InternalErrorException.html) | Indicates an internal service error. |
| [InvalidParameterException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.InvalidParameterException.html) | Indicates that a request parameter does not comply with the associated constraints. |
| [NotFoundException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.NotFoundException.html) | Indicates that the requested resource does not exist. |
| [PlatformApplicationDisabledException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.PlatformApplicationDisabledException.html) | Exception error indicating platform application disabled. |
| [SnsException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.SnsException.html) | Default service exception class |
| [SubscriptionLimitExceededException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.SubscriptionLimitExceededException.html) | Indicates that the customer already owns the maximum allowed number of subscriptions. |
| [TopicLimitExceededException](http://docs.aws.amazon.com/aws-sdk-php/v2/api/class-Aws.Sns.Exception.TopicLimitExceededException.html) | Indicates that the customer already owns the maximum allowed number of topics. |