

Exception Handling

December 29, 2016

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | Exception Handling |
| **Program Name** |  |
| **Author** | P M Arjun |
| **Version** |  |
| **Status** |  |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 1.0 | 11/08/2016 | Initial | P M Arjun |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

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

Contents

[1. Overview 2](#_Toc466379865)

[2. Logging Framework 2](#_Toc466379866)

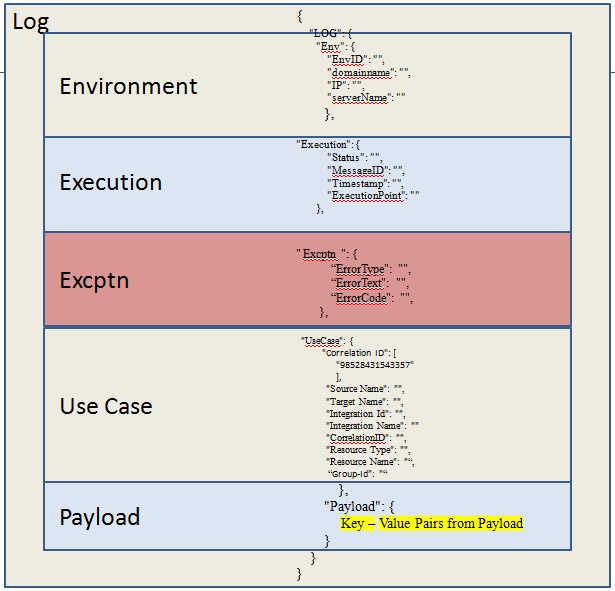
[3. Exception Handling Process 3](#_Toc466379867)

1. Overview

This document provides a generic process on how to handle the exceptions across all the Coca Cola Integration projects. It contains the basic exception strategy to be used and how exceptions are logged so that they are handled.

1. Logging Framework

In Coca Cola Integration projects logging is performed at each level starting from the main flows, to the sub flows, iterative flows and even the exceptions are logged so that they can be handled. Mulesoft CloudHub generates the logs and they are audited and analyzed using Splunk. Each log that is generated must follow a format as shown below.

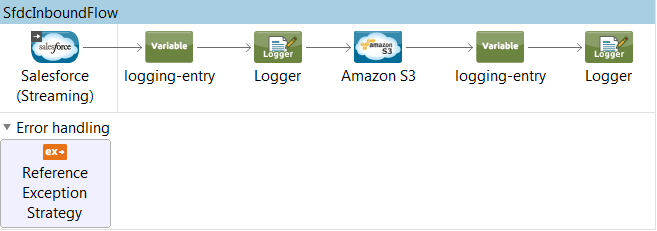


1. Exception Handling Process

So we log the flows at every level, similarly when an exception occurs we log the flow so that exception can be handled appropriately.

Let us consider a flow in understanding how exceptions are handled using logging.

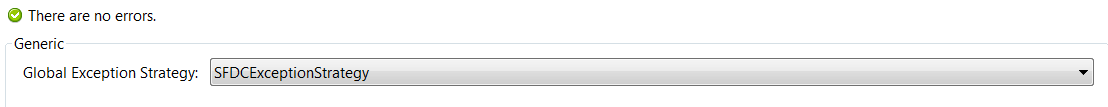
**Using Reference Exception Strategy**:



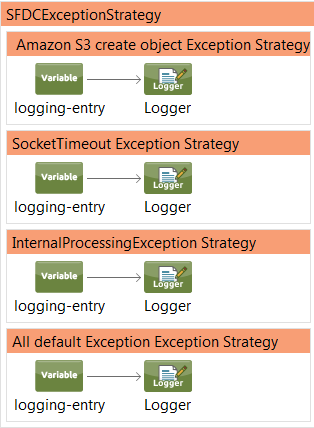
In the above flow we are using Reference exception strategy in order to refer and adhere to the error handling parameters defined in a global choice exception strategy.

So in the above flow the inbound connector is a Salesforce connector, we receive the inbound message, log the inbound message and then we would send the message in the form of an object and store it in the S3 bucket and we log at the end of the flow again as per the logging guidelines. Now during the entire flow there are various exceptions or error’s that could occur.

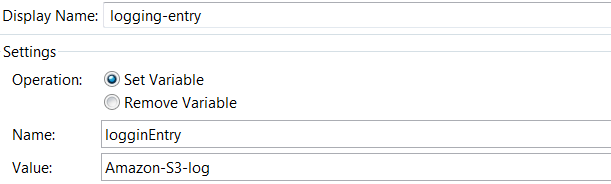
In case any kind of exception occurs Reference exception strategy would refer a global Choice Exception strategy, while in this case we are using a SFDCException Strategy.



As SFDC Exception strategy is a global exception Choice strategy it would have multiple exception strategies for the exceptions that could occur across different flows.



Now suppose it is an S3 Client exception, that is some error has occurred while storing an object into the S3 bucket or creating an object or the S3 bucket does not exist, that is any exception that occurs when we try interact with Amazon Simple Storage Service, in such a case Amazon S3 create object Exception Strategy would be chosen. Now all the strategies in SFDCException strategies will have the same components, they only vary in the variable value.



As we are handling exceptions by logging them, we need to have the logging framework configured within our project.

So the three main configurations that we need to setup are:

1. Add external jar “loggingframework.jar”.



1. In the src/main/ resources folder, create metadata folder and add the json file named “logging-metadata.json”. Please find the file below:



1. In the src/main/ resources folder, create properties folder and add the properties file named “loggingsetup.properties”.



There are few lines that needs to be added to the configuration.xml as well:

xmlns:context=<http://www.springframework.org/schema/context>

<context:property-placeholder location=*"properties/loggingsetup.properties"*/>

<spring:beans>

<spring:import resource=*"classpath:logging-config.xml"*/>

</spring:beans>

So each time we log we need to set a variable with the name ‘loggingEntry’ and the variable value that would be an ID that would match a unique value in the ‘logging-metadata.json’ file.

Next we need to place a logger with the message #[LoggingAppendString]. This message is basically used to print the logging details that is collected by the external jar.

We use a logging meta-data contained within a JSON file in order to store different logging attributes. The attributes might change by environment and therefore can be configured globally as such with this meta-data file. The file is called ‘logging-metadata.json’. In addition, please also add a ‘LoggingEntry’ POJO to the project which gets populated with the metadata at startup time.

So information regarding each exception that could occur must be specified in the metadata json file.

So in the above flow, S3 Client Exception, SocketTimeOut Exception, Internal message processing exception and other default exception are the possible exceptions that could occur. Information regarding these exceptions has to be provided in the json metadata file. Now these information regarding exceptions are flow specific. For the above flow json file would have following information regarding exceptions:



Coming back to the flow, the variable value that we have provided is ‘Amazon-S3-log’, so depending upon this particular value the external jar would collect data regarding this particular exception from “logging-metadata.json” file.

A Singleton class which is generic to all integrations needs to be added to the project and registered as a singleton for the logging to work at runtime. A ‘StartupUtilityBean’ is provided as a part of most projects which can be used for this purpose. Register this as a singleton class as below to ensure the metadata gets picked up correctly.

|  |
| --- |
| <spring:beans>  <spring:bean id=*"StartupUtilityBean"* name=*" StartupUtilityBean"* class=*"com.cocacola.services.startup. StartupUtilityBean"* scope=*"singleton"*/>  </spring:beans> |

The Singleton class would match the data in the Logging-metadata.json file using the id i.e the variable value. It would then extract all the information from the Json file.

So the log would have three attributes with respect to exception:

1. **Error Code**
2. **Error Type**
3. **Error Text.**

We fetch the ‘ErrorCode’ and ‘ErrorType’ from the metadata json file, whereas ‘ErrorText’ is extracted from the payload during runtime.

The log when an exception occurs would be of the following format:

{

"LOG": {

"Excptn":{

"ErrorType ":"org.mule.module.s3.exception.S3ClientException",

"ErrorText":"Failed to invoke ScriptComponent{testloggingframeworkFlow.component.295466076}. Component that caused exception is: ScriptComponent{testloggingframeworkFlow.component.295466076}. Message payload is of type: String",

"ErrorCode":"E104"

},

"Execution": {

"Status": "success",

"ExecutionPoint": "within a flow",

"Timestamp": "2016-12-27 04:33:25:36",

"MessageID": "1663d870-cc24-11e6-bc65-2ecc20524153"

},

"UseCase": {

"Resource Type": "flow",

"Integration Name": "INT SalesforceInbound[DEV]",

"Source Name": "Salesforce",

"Resource Name": "mainflow",

"Target Name": "CAIDM",

"Integration ID": "UID000000453431",

"Correlation ID": "98519300169665"

},

"Env": {

"IP": "10.0.0.6",

"domainName": "LIN36000529.corp.capgemini.com",

"EnvID": "LIN36000529..Exception Hnadling ",

"serverName": "LIN36000529"

}

"Payload": {},

}

So once we know the exception that is occurring, we can appropriately handle them.

Similarly the process followed for all the exceptions across all the integrations.