

Logging Template

January 11, 2017

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | Logging Template |
| **Program Name** |  |
| **Author** | P M Arjun |
| **Version** |  |
| **Status** |  |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 1.0 | 15/12/2016 | Initial | P M Arjun |
| 1.1 | 10/01/2017 | Updated use of logging framework guidelines | Ankit Prakash |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

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

Contents

[1. Introduction 4](#_Toc471918563)

[2. Basic Structure 4](#_Toc471918564)

[3. Logging Guidelines 5](#_Toc471918565)

[4. Prerequisites 5](#_Toc471918566)

[5. Working with Logging Framework 6](#_Toc471918567)

1. Introduction

This document provides us the generic steps to be followed to set up logging framework across all

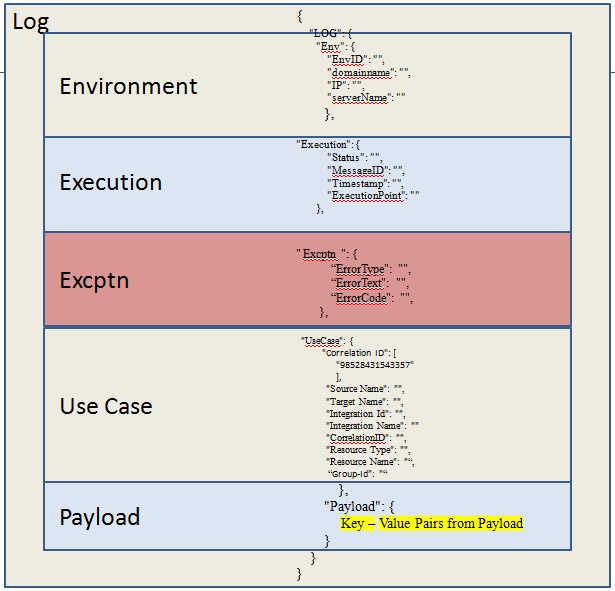
the integration projects.

1. Basic Structure

Logging is performed at each level starting from the main flows, to the sub flows, iterative flows

and even the exceptions are logged. 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. Logging Guidelines

Logging has to be performed on the basis of the below guidelines:

1. Entry to a main flow has to be logged.
2. Exit from a main flow has to be logged.
3. Entry to a sub-flow has to be logged.
4. Exit from a sub-flow has to be logged.
5. If the sub-flow is iterative, log entries have to be made for each iteration.
6. Create a log entry before and after a business object is modified (transformed)
7. Create a log entry when accessing external resources such as object store, lookup repository, S3, message systems etc.
8. Log exception with all data provided in the structure including exception section.
9. Prerequisites
10. Create a mule project.
11. Add external jar “loggingframework.jar”. Please find the jar below:



If the code is mavenized then install this jar in your local repository. In order to do this place the jar in lib folder of your application and configure the following dependency in the pom.xml

<dependency>

<groupId>com.icoe</groupId>

<artifactId>loggingframework</artifactId>

<version>2.0.0</version>

</dependency>

Then open the command line from the location where the pom is located and type following command to install the jar in local repo.

mvn install:install-file -DgroupId=com.icoe -DartifactId=loggingframwork -Dversion=2.0.0 -Dpackaging=jar -Dfile=lib\loggingframework-2.0.0.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”.

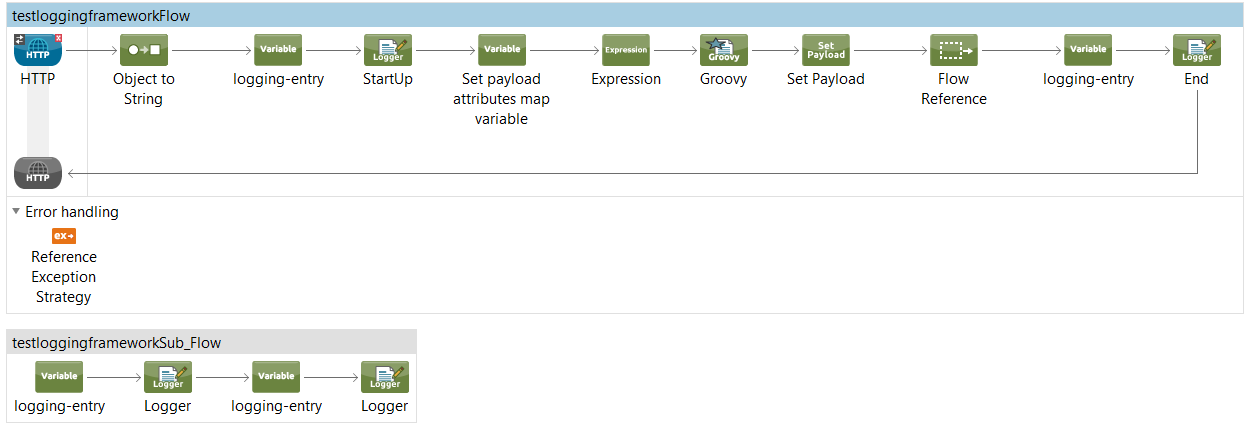


1. Working with Logging Framework

As mentioned earlier there are three mandatory logging flow:

1. We need to log when main flow starts and ends.
2. We need to log when sub-flow starts and ends.
3. We need to log when an error occurs.

So let us consider a flow and generate the logging for the flow:



Before we look into the flow, there are some changes that needs to be made in configuration.xml, we need to add the below lines:

**<context:property-placeholder**

**location=*"properties/loggingsetup.properties"*/>**

**<spring:beans>**

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

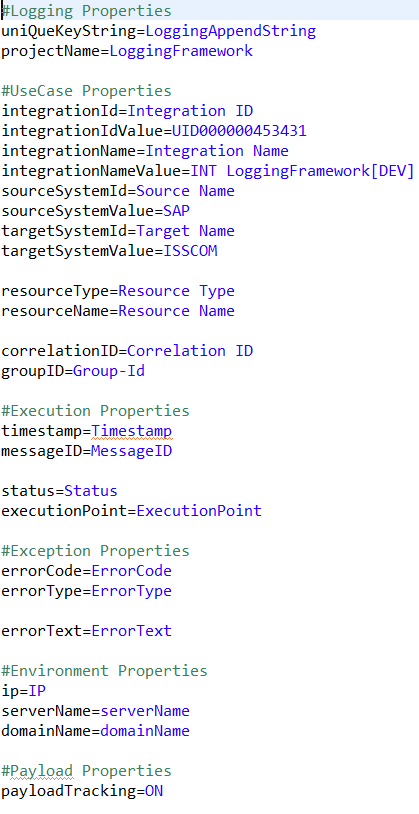
**</spring:beans>**

In the above flow, we receive an inbound request or a mule message along with payload from the HTTP connector, here we need to log the flow at the beginning as well as at the end of the flow.

For this we need to set a variable with the name ‘loggingEntry’ and the variable value would be an ID which matches a unique entry id value in the ‘logging-metadata.json’ file. Next we need to place a logger with the message #[LoggingAppendString]. These two components are the key components for logging.

Here we need to import an external jar ‘loggingframework-2.0.0.jar’, which is responsible for preparing different key -value pair values with the help of java classes for logging.

Some of the key value pairs are hard coded while some are extracted from the payload or set during the runtime. Now the ‘loggingsetup.properties’ inside the properties folder would contain all the key and value pairs for logging. Only some key value pairs are constant for the entire application project, where as other values would change with each flow or sub flow. So the loggingsetup.properties would have following key value pairs.



Now the integaration ID, Value, Name, Namevalue, SytemId, sourceSytem ID, value ,targetSystem Id values would be constant and will not change throughout this project, for any flow inside this integration these values are same, so we can directly hard code these values. Now key value pairs such as resourceType these vary from flow to flow and from each log to the next log and like timestamp, messageID is different every time so they can be set only during the runtime. So the keys are defined in the loggingsetup.properties but the values might either be collected from the metadata that is defined in ‘logging-metadata.json’ file or from the payload.

There is singleton class with the name “StartupUtilityBean” is placed inside the jar, used to read “logging-metadagta.json” file if the file is placed at location resources/metadata/logging-metadata.json. Otherwise it will throw an error “class path resource [metadata/logging-metadata.json] not found because it does not exit”

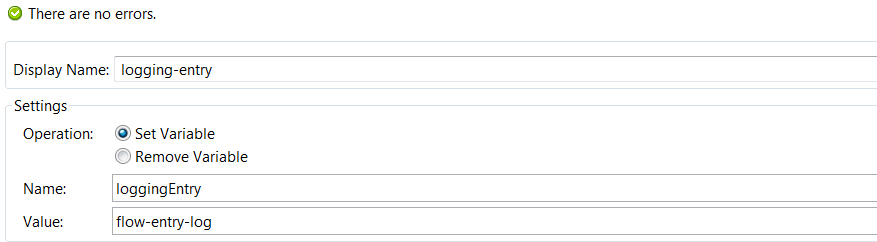
|  |
| --- |
| <spring:beans>  <spring:bean id=*"StartupUtilityBean"* name=*" StartupUtilityBean"*  class=*"com.cocacola.services.startup.StartupUtilityBean"* scope=*"singleton"*/>  </spring:beans> |

Logging meta-data contained a JSON file in order to store different logging attributes. The attributes might change by flow to flow and therefore can be configured as entry such with this meta-data file. The metadata in logging-metadata.json file would contain the logging information in the following form:

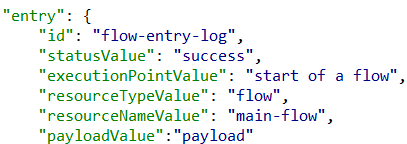


If there is any syntax error or if the json is invalid then it will throw an error “<errorMessage> , syntax error because of class path resource [metadata/logging-metadata.json] has invalid format”. To avoid this error developer needs to install Json Editor Plugin in Anypoint Studio from location “JSON Editor - <http://boothen.github.io/Json-Eclipse-Plugin/> ”. The Json editor immediately will notify us the line in which there is a syntax error or we have provided an invalid format.

Now in the above flow we saw that first we set a variable with name ‘loggingEntry’ and the variable value that would be a unique ID that would match a unique value in the ‘logging-metadata.json’ file. If the loggingEntry value which is used in code is not available in json file, then it will throw an error “<loggingEntryValue> loggingEntry not found in class path resource [metadata/logging-metadata.json]".

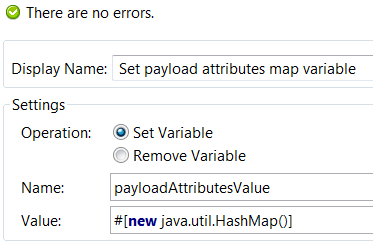


In the above variable the value is ‘flow-entry-log’, this would be a unique id in fetching the values from the metadata json file. In this case from the above shown metadata it would fetch the following information for logging:



The resourceTypeValue could be set in the metadata json file or if it is empty or has a blank space, the jar will search for a value from the flow variable resourceTypeValue, as it can change from one environment to another. Similarly for the payload value depending on the value that we specified in the metadata json file it can change based on requirement and the payloadTracking should kept **ON** in loggingsetup.properties file while tracking the payload, if the payloadValue has “payload” as specified value then it will print the payload that is locked and set earlier.

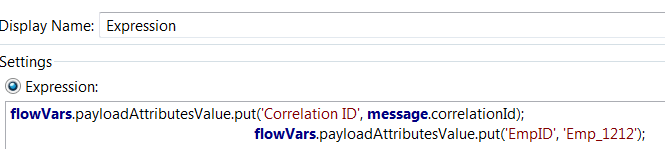
If the payloadValue is “payloadAttribute” then it would search for the attributes that are set earlier and print that, else it could also be empty. The payload attributes can be set by creating a HashMap like



After this we need to put the values in the map using expression like flowVars.payloadAttributesValue.put(Correlation ID,message.coorelationId);

flowVars.payloadAttributesValue.put(EmpId,’Emp\_1212’) ;

is used to set the payload attributes value and the jar will then pick up that value.



So once we place the logger with message #[LoggingAppendString], it would print all the information that is collected by the jar for logging and in json file details are given as

{

"entry": {

"id": "mainflow-entry-Log",

"statusValue": "success",

"executionPointValue": "mainflow",

"resourceTypeValue": "flow",

"resourceNameValue": "mainflow",

"payloadValue":"payloadAttribute"

}

},

Similarly, we would set up a variable loggingEntry and a value subflow-entry-log that would fetch the information from metadata json file and the logger with message #[LoggingAppendString] that would print the log. So, the log generated would be as shown below:

{

"LOG": {

"Execution": {

"Status": "success",

"ExecutionPoint": "mainflow",

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

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

},

"UseCase": {

"Resource Type": "flow",

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

"Source Name": "SAP",

"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..loggingframework-template",

"serverName": "LIN36000529"

}

"Payload": {

"Correlation ID": "98519300169665",

“EmpID”: ”EmpID\_1212”

},

}

}

Some values are set during the runtime, such as the correlation Id. Correlation ID is 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. In most cases the inbound message would contain the correlation Id, if it does not have the correlation Id then the jar will create an ID within the above given specifications. If it is already present in the payload then we need to set it as a payload attribute value. We can use the expression component to set it as an attribute value.

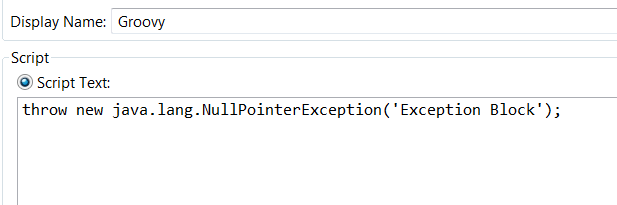
Above we are setting correlation Id and EmpID as attributes and theses are logged at the end of the main flow.

Next the flow references the subflow. Now the subflow has to be logged at the beginning as well at the end. Same process will be followed for this sub-flow as well.

So, for each subflow entry and exit there should be entries in the metadata json file that provides the global configurations about that environment.

Similarly we also log the exceptions, all the possible exceptions that could occur during the flow are logged, so information regarding each exception that could occur must be specified in the metadata json file.

Suppose we are creating an exception using the groovy component:



So this will throw a NullPointerException, the reference exception strategy would call a global exception strategy where the exception would be caught and logged. Similarly we need to set up a variable loggingEntry and a value NullPointerException that will fetch the information from metadata json file and the logger with message #[LoggingAppendString] that would print the log.

<choice-exception-strategy doc:name=*"Choice Exception Strategy"*>

<catch-exception-strategy

when=*"#[exception.causedBy(java.lang.Exception)]"*

doc:name=*"Catch Exception Strategy"*>

<set-variable variableName=*"loggingEntry"*

value=*"NullPointerException"* doc:name=*"logging-entry"*/>

<logger message=*"#[LoggingAppendString]"* level=*"ERROR"*

doc:name=*"Logger"*/>

</catch-exception-strategy>

</choice-exception-strategy>

The value which loggingEntry variable has , jar will search the same value in json for logging.

{

"entry": {

"id": "NullPointerException",

"statusValue": "exception occurs",

"executionPointValue": "exception handling",

"resourceTypeValue": "flow",

"resourceNameValue": "flow name",

"errorCodeValue": "E104",

"errorTypeValue": "java.lang.NullPointerException",

"payloadValue": ""

}

}

The log would contain some extra information regarding the exception, the error code and the error type are already specified in the json file while error text would be generated only during the runtime.

So the log that would be generated when an exception occurs is shown below:

{

"LOG": {

"Env": {

"EnvID": "LIN36000266..LoggingFramework\_POC",

"IP": "10.95.11.13",

"domainName": "LIN36000266.corp.capgemini.com",

"serverName": "LIN36000266"

},

"Execution": {

"Status": "exception occurs",

"MessageID": "4af9dad0-c8f5-11e6-b9b8-de7d20524153",

"Timestamp": "2016-12-23 03:20:53:572",

"ExecutionPoint": "exception handling"

},

"Excptn": {

"ErrorText": "Failed to invoke

ScriptComponent{testloggingframeworkFlow.component.1095090810}.

Component that caused exception is:

ScriptComponent{testloggingframeworkFlow.component.1095090810}.

Message payload is of type: String",

"ErrorType ": "java.lang.NullPointerException",

"ErrorCode": "E104"

},

"UseCase": {

"Correlation ID": "98111740881289",

"Source Name": "SAP",

"Group-Id": "file\_2016-12-23 03:20:53:307.txt",

"Target Name": "CAIDM",

"Resource Type": "flow",

"Integration ID": "UID000000453431",

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

"Resource Name": "flow name "

},

"Payload": {

}

}

}

In the flow next we would log the end of the main flow similarly as we did for the beginning of the flow. So all the logging must be done based on the guidelines that is mentioned at the beginning of the document.

In this logging frameowork the developer need not explicitly specify all the attributes that need to be logged. There is a lot of reduction in the code. All the values are implicty derived by the loggingframework jar using metadata json file and setting the loggingsetup.properties.