

Logging Template

December 28, 2016

**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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

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

Contents

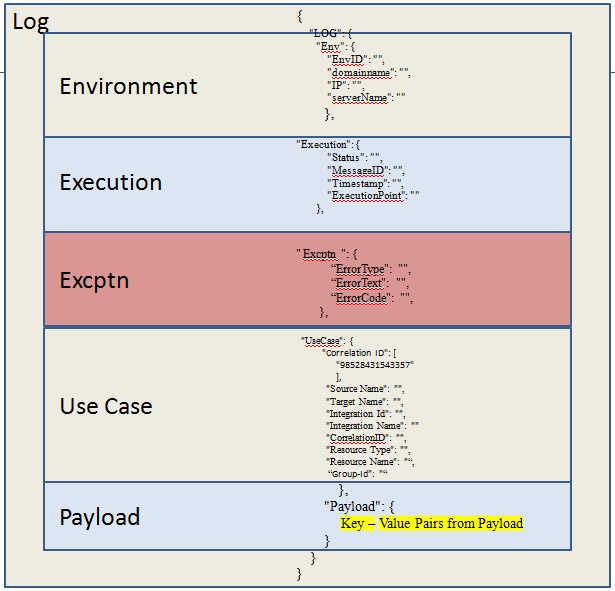
1. Introduction…………………….4
2. Basic Structure…………………4
3. Logging Guidelines…………..5
4. Prerequisites……………………5
5. Logging Framework………….6

**Introduction:**

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

**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.



**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 payload is not available when entering a flow, a log entry at the info level has to be created at the point when payload is available.
6. If the sub-flow is iterative, log entries have to be made for each iteration.
7. Create a log entry before and after a business object is modified (transformed)
8. Create a log entry when accessing external resources such as object store, lookup repository, S3, message systems etc.
9. Log exception with all data provided in the structure including exception section.

**Prerequisites:**

1. Create a mule project.
2. Add external jar “loggingframework.jar”. Please find the jar below:



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”.

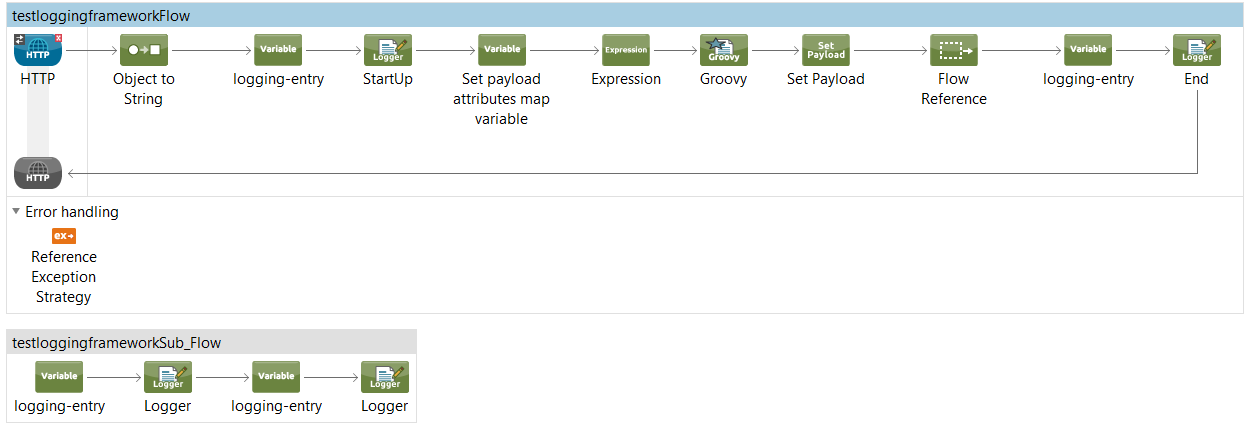


**Logging Framework:**

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:

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 we receive an inbound request or a message from the HTTP connector, the payload then goes through an object to string component.

So we need to log the flow at the beginning as well as at the end of the flow. So logging really starts with the next component. 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.

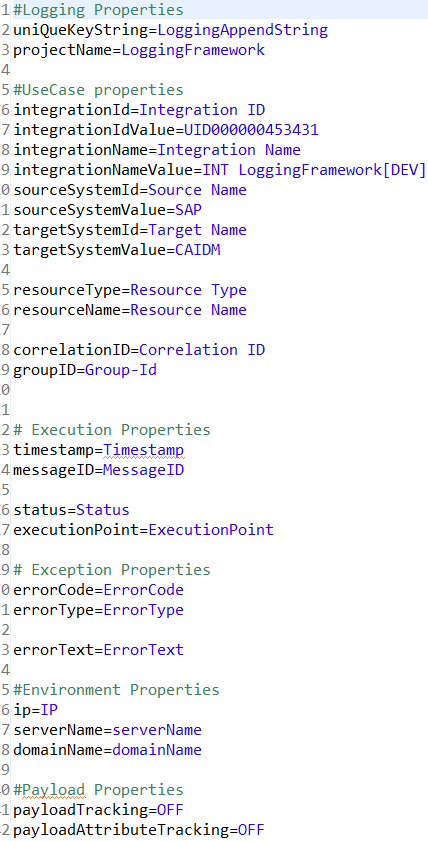
Next we need to place a logger with the message #[LoggingAppendString].

These two components are the key components for logging.

So we would be importing an external jar ‘loggingframework.jar’ that would call different java classes and set different key value pairs 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 certain key value pairs are constant for the entire 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, value would all 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 timestamp, message id, resourceName, resourceType these vary from flow to flow and from each log to the next log and they can be set only during the runtime.

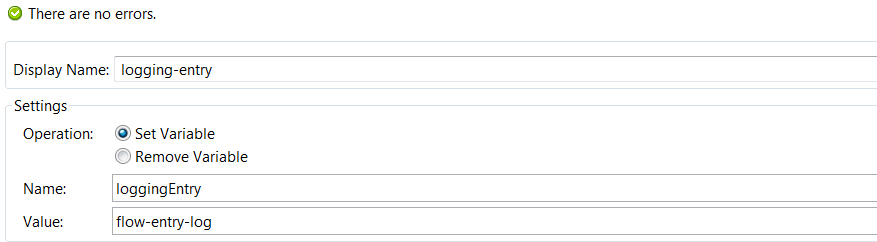
So the keys are defined in the loggingsetup.properties but the values might either be collected from the metadata that is defined in ‘logging-metadata.json’ file or from the payload. There is Singleton class that the jar calls that is generic to all the integration projects and registered as a singleton 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> |

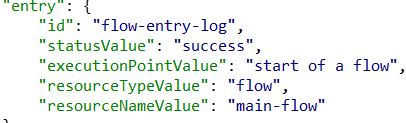
We use a logging meta-data contained within a JSON file in order to store different logging attributes. The attributes might change by environment and therefore can be configured globally as such with this meta-data file. The file is called ‘logging-metadata.json’. The metadata in logging-metadata.json file would contain the logging information in the following form:



So in the flow we saw that we first 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.



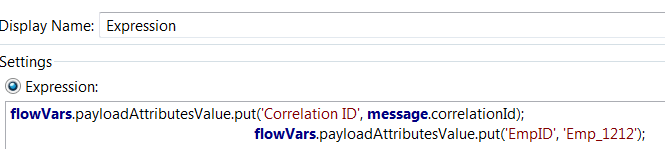
So in the above variable the value is ‘flow-entry-log’, this would be a unique id in fetching the values from the metadata json file. In this case from the above shown metadata it would fetch the following information for logging:



So once we place the logger with message #[LoggingAppendString], it would print all the information that is collected by the jar for logging.

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 would 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 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.

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": "within a subflow",

"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": "subflow",

"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": {},

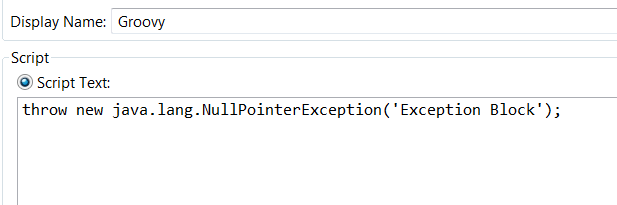
}

}

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

In the flow using the groovy component we are creating an exception:



So this would throw a NullPointer Exception, the reference exception strategy would call a global exception strategy where the exception would be caught and logged.

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.

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.