**Monitoring Profiling**

1. **Introduction:**

This document provides in depth design details of the profile monitoring for the project. It captures enabling of profile monitoring and log the high level and low-level details events occurs in the flow.

1. **Purpose & Audience**
   1. **Purpose:**

The purpose of this document is to provide high and low level functional details of the monitoring profiling for the project. This document serves as application design for the actual implementation of service using ACE.

* 1. **Audience:**

This document should be read and understood by all stakeholders, architects, technical leads, developers and test team with an interest in the detailed workings of the system. Most importantly, this applies to those who will use it to carry out their business responsibilities and those who are building the service. Developers will use this document as input for implementation. Application implementers will use this document to coordinate configuration and deployment of the coded solution. Test team will use this in conjunction with other documents to prepare test cases.

1. **Profile Monitoring Design :**

**Logging and Error handling:**

Logging and Error handling implemented using ACE Monitoring feature. Message flows configured to emit monitoring events. These events consumed by ACE or external applications for transaction logging, monitoring and tracking. Emission of the events controlled dynamically by monitoring profiles.

**Event Monitoring:**

This feature lets you configure a message flow to emit monitoring events without any coding. When an event occurs in a message flow, an event message generated. These event messages are in XML format and published on a pre-defined topic. These events consumed by applications to track and monitor the transactions flowing through IIB. Event messages generated at various points within a message flow on need basis.

**This Event message has four sections:**

* eventPointData
  + This section contains event source details
  + It also contains 3 different transaction id’s – local, parent and global
* Message Flow Data
  + This section contains IIB node, Integration Server, application, message flow and message flow node level details
* Application Data
  + In this section, we can pass any user-defined piece of information. For this project, application specific metadata information is passed in this section
* Bit stream Data
  + This section contains application payload and headers as base-64 encoded/hexadecimal/CDATA



**Event Correlation:**

Event monitoring allows up to 3 Transaction id’s that can be used to correlate transactions. They are:

* LocalTransactionId
  + Unique ID that identifies transaction. This ID can be generated dynamically or taken from payload
* ParentTransactionId
  + This is used to establish relation between 2 components (within or across IIB) using some form of ID. Custom value can also be set
* GlobalTransactionId

This is a unique ID used to track transactions when there are multiple hops

**Logging:**

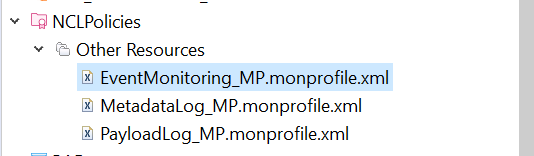
As mentioned earlier, logging is achieved using event monitoring. Events are generated as consists of application payload, application metadata and transaction id’s.

When a message flows through ACE flow, enabled monitoring events result in the generation of log messages which are published on standard ACE topic.

### Logging component

Logging component is a flow which receives message from main message flow, construct metadata and payload log events depending on which events are enabled. The generation of these events can be dynamically controlled by monitoring profiles which is described in previous section.

“CreatePayloadLog” - This compute node has been configured to emit logs, which contain application payload, Application Data, log event of audit and error handling Data along with metadata and transaction id’s.

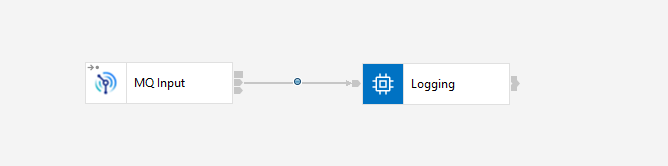
NCLPolicies is a policy created to include the MetadataLog\_MP.monoprofile.xml. and deployed this policy bar file into the respected server bar file location.

**MQ End Point:**

Created one local queue “EVENT\_LOG” and subscriber for consuming event messages from integration server topic.

Queue name: EVENT\_LOG

Subscriber name: '$SYS/Broker/integration\_server/Monitoring/#/#/#'



EVENT\_LOG API developed to read the event messages from the respected the EVENT\_LOG, frame the event messages, and log into the respective database table.

**Logging Functions:**

Below functions are used in the services, given functions place the require details into the environment.eventlog field.

**CreateAuditLogEvent():** This function creates basic log event structure in Environment tree and sets transaction ids.

**AddLogEvent():** This function adds audit trail entry to log event.

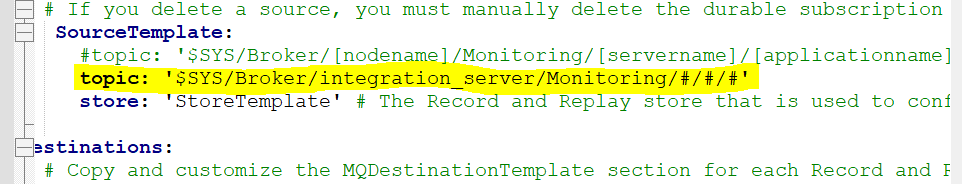
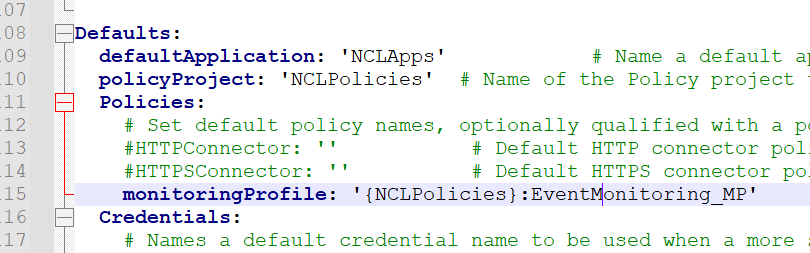
**AddPayloadLogEvent():** This function adds application payload to log event.

**CreateMetadata():** This functions adds application metadata to log event.

**UpdateAuditLogEventWithError():** This functions updates log event with error details.

**Procedure to enable the logging:**

1. Create a policy with given monoprofilexml file.
2. Create the bar file for policy and deploy in the respected container.
3. Add the policy and topic in the server yaml files:



1. Create the subscriber and queue in the queue manager.
2. Restart the server and run the one service.
3. Event data will be placed in the EVENT\_LOG queue.
4. Create one table(NCL\_EVNT\_LOG) in NCLSEA and grant permissions for NCLWEBC.

CREATE TABLE "NCL\_EVNT\_LOG"

("EVENT\_LOG\_TYPE" VARCHAR2(30 BYTE),

"SERVICE\_NAME" VARCHAR2(50 BYTE),

"FLOW\_NAME" VARCHAR2(50 BYTE),

"TRANSACTION\_ID" VARCHAR2(50 BYTE),

"LOGGING\_TIME" VARCHAR2(50 BYTE),

"EVENT\_NAME" VARCHAR2(50 BYTE),

"SERVER\_NAME" VARCHAR2(30 BYTE),

"STATUS\_CODE" VARCHAR2(10 BYTE),

"EVENT\_LOG\_FLAG" VARCHAR2(10 BYTE),

"EVENT\_LOGS" CLOB,

"LOG\_MESSAGE" CLOB,

"METADATA\_FLAG" VARCHAR2(10 BYTE),

"METADATA\_LOGS" CLOB,

"FINAL\_RESP\_MSG" VARCHAR2(100 BYTE));

GRANT DELETE ON NCLSEA.ncl\_evnt\_log TO NCLWEBC;

GRANT INSERT ON NCLSEA.ncl\_evnt\_log TO NCLWEBC;

GRANT SELECT ON NCLSEA.ncl\_evnt\_log TO NCLWEBC;

GRANT UPDATE ON NCLSEA.ncl\_evnt\_log TO NCLWEBC;

GRANT ALL PRIVILEGES ON NCLSEA.ncl\_evnt\_log TO NCLWEBC;

1. Develop one API to read the event data from queue and place into Database table.
2. Check the table for logs.

Note: default all the servers enabled with monitoring profile.

We can use below option to start and stop the flow monitoring of a server.

<http://hdqqibmace02lv.nclmiami.ncl.com:7609/apiv2/monitoring/flow-monitoring/start-monitoring>

<http://hdqqibmace02lv.nclmiami.ncl.com:7609/apiv2/monitoring/flow-monitoring/stop-monitoring>

