**Development -**

**Creating a tooling project**

* When you start WSO2 Integration Studio, a default workspace is created to store your projects. Change the default workspace location to a preferred location that you can easily access.
* Create ESB project for specific use case.
* When you create the project structure, add it to source control such as Git.
* When you create projects, ensure that you follow the standard naming conventions.

**Creating artifacts –**

* When you create an artifact, use the proper naming convention from the start.
* If an ESB project contains many artifacts related to multiple use cases, name the artifacts by fixing the use case name.
* Create specific ESB projects for specific use cases. This makes it easy to manage the code/project.
* Design the ESB logic into loosely coupled modules.
* Avoid creating large, complex mediation code as much as possible. Always follow the separation of concerns design principle and split lengthy logic into separate mediation components.
* When you have a common set of code, implement it in a sequence or a template that can be reused.
* Externalize endpoint and policy references using the Registry or Property file.

**Naming artifacts –**

find us below example how naming conventions mention sub folders based on main folder name.Same way follows similar structure for new artifacts.

Project Name: Prepaid (main folder)

Subfolder (DSS) : Prepaid+DataService.

Subfolder (Connector) : Prepaid+Connector

Subfolder (Registry) : Prepaid+Registry

Subfolder (Mediator) : Prepaid+Mediator

Subfolder (Compositeexporter) : Prepaid+Compositeexporter

Subfolder (Connectorexporter) : Prepaid+Connectorexporter

Subfolder (Registryexporter) : Prepaid+Registryexporter

**Using mediators –**

**Using the Log mediator -**

* When a server encounters errors, you should have enough information about the errors in the logs.
* To log mediated messages, the best option is to use the Log mediator.
* In a production setup, it is not recommended to use the Log mediator within sequences and proxy services other than in the Fault sequences.
* If you are in a development environment, ensure that you set the global log level of org.apache.synapse.mediators.builtin.LogMediator to DEBUG.
* If you are in a production environment, ensure that you set the global log level of org.apache.synapse.mediators.builtin.LogMediator to INFO. Setting the log level to INFO allows you to troubleshoot the message flow in production whenever required without having to go through the synapse configuration.
* You should use Log mediators in the FaultSequence to capture information about errors that occur. The logs entries printed via the FaultSequences will be available in the wso2carbon.log that is in the main log file.

**Using the Class mediators -**

* You should not write a Class mediator if the intended functionality can be achieved using the enterprise service bus capabilities of WSO2 Micro Integrator. Following this practice avoids maintenance overhead.
* If you are writing a Class mediator, ensure that you have a good understanding of the performance impact and possible memory leaks so that you can take these into consideration.

**Using the Script mediators –**

* the advantage for this we can write inline easy to test and deploy but the disadvantage of this performance is much slower than the Class.

**Using the Call vs. Callout vs. Send mediators** -

* You can use either the Call mediator, Callout mediator or Send mediator to send a message to an endpoint.
* If you use the Send mediator to send a message, the response goes to the OutSequence (or to the specified receiving sequence).
* If you use the Call mediator to send a message, the response goes to the next mediator, which is placed right after the send mediator in the mediation flow.
* The use of Call mediator is recommended for service chaining scenarios.
* Both the Send mediator and Call mediator use the non-blocking transport. Therefore, there is no difference between the performance of the Send and Call mediators.
* Behaviour of the Callout mediator is similar to the Call mediator, but it uses the blocking transport to send the message out. Therefore, in terms of performance, Callout mediator is not as good as the Call mediator or Send mediator. If there are scenarios where blocking behaviour is required, you can use the Callout mediator. For example, see JMS Transactions.

**Using the For Each mediator -**

* Use the For Each mediator only for message transformations. If you need to make back-end calls from each iteration, then use the iterate mediator, For Each supports modifying the original payload.

**Using the Iterate and Aggregate mediator -**

* You can use Iterate for situations where you send the split messages to a target and collect them by an Aggregate in a different flow.
* You need to always accompany an Iterate with an Aggregate mediator. For Each loop over the sub-messages and merges them back to the same parent element of the message.
* In Iterate you need to send the split messages to an endpoint to continue the message flow. However, for Each does not allow using Call, Send and Callout mediators in the sequence.
* for Each does not split the message flow, unlike Iterate Mediator. It guarantees to execute in the same thread until all iterations are complete.

**Using the Clone mediator –**

* Clone is mostly used for scatter gather Patterns.
* When using a Clone mediator, use a Call mediator in the target sequence to bring the responses back into the In-Sequence. This continues the mediation since the Continuation Stack gets pushed into the Synapse Message Context via the handle Message method in the SynapseCallbackReceiver class.
* Otherwise, the Continuation Stack becomes empty in the Synapse Message Context if you do not use a Call mediator in the target sequence.

**Using the Loopback mediator -**

* Do not include the Loopback mediator in the OutSequence .
* When a message passes from the InSequence (request path) to the OutSequence (response path), you cannot use the Loopback mediator to move the message back to the OutSequence again.

**Using the Send mediator -**

* You should not specify any mediator after the Send mediator or the Respond.
* A message flow must end from these two mediators.

**Mediators for message transformation -**

* Follow the guidelines below to use mediators in scenarios that require message transformation:
* The most common message transformation use case is reconstructing the entire message payload according to the required format of the back-end service. You can use the PayloadFactory Mediator to do the transformation in this use case.
* If the original message has repetitive segments and if you want to transform each of those segments into a new format with repetitive segments, you can use the For-Each Mediator together with the PayloadFactory mediator. The For-Each mediator iterates through the repetitive segments and the PayloadFactory mediator constructs the segments of the message to a new format.
* If only a small part of the message needs to be modified (e.g., you need to add/remove an element) you should use the Enrich Mediator.
* If the transformation logic is complex (i.e., the logic includes repetitive segments, conditional transformations, etc), you can use the XSLT Mediator to do the transformation.
* **Note -**
* When you use the XSLT mediator, a third-party engine does the message transformation. Therefore, this can impact the performance.
* If you want to change the structure of the data in a message or convert and transform one data format to another, use the Data Mapper Mediator.
* If you want to work with large file transformations, use the Smooks Mediator.

**General mediator best practices -**

* Use the Iterate mediator in association with the Aggregate mediator.
* Do not do any configuration after the Send mediator.
* Do proper error handling to handle mediation errors as well as endpoint errors.
* Use appropriate intervals for tasks.
* End of the sequence Use drop mediator for schedule tasks.
* Use the ForEach mediator only for message transformations. If you need to make back-end calls from each iteration, then use the iterate mediator.
* Do not use the DB mediators (DBReport and DBLookup ) with complex SQL queries or in scenarios where you need to simultaneously retrieve multiple rows. Instead, use the data services functionality of WSO2 Micro Integrator.
* Use dollar context (i.e., $ctx) instead of get-property() . This is because the get - property methods search even in Registry if the value is not available in the message context. Thus, it affects performance as Registry search is an expensive operation. However, $ ctx only checks in the message context.

**Reusing a defined sequence -**

* If you want to repeatedly use the same mediation sequence, you can define it in a sequence. Then you can call the mediation sequence from the main sequence.
* When you are adding the last mediator in a sequence, make sure to use one of the following mediators depending on the scenario. Any mediator added after one of the following mediators will not be applied.
* Use the Respond mediator as the last mediator if you want the message to be sent back to the client.
* Use the Send Mediator as the last mediator if you want the message to be sent to a specific endpoint. If you want to mediate the message after it is sent, you can use the Clone mediator to make two copies of the message and process them separately in order to avoid conflicts.
* Use the Drop mediator as the last mediator if you want to stop the mediation flow at a particular point (e.g., when a filter condition is not met).
* Use the Loopback mediator as the last mediator if you want the message to be moved from the InSequence to the OutSequence.
* The Loopback mediator prevents the execution of subsequent mediators in the In Sequence.
* Use the Store mediator as the last mediator if you want to enqueue messages to a message store.

**Working with proxy services –**

* Use proxy services for SOAP service development instead of RESTful service development.
* **Offers Soap Message Formats.**

**Working with REST services –**

The following are some [best practices](https://s3.amazonaws.com/tfpearsonecollege/bestpractices/RESTful+Best+Practices.pdf) to keep in mind when designing your APIs for use with REST.

* **Use meaningful resource names** to clarify what a given request does. A RESTful URI should refer to a resource that is a thing instead of an action. The name and structure of URIs should convey meaning to those consumers.
* **Use HTTP methods -** Use POST , GET , PUT , DELETE  in requests to clarify the purpose of the request. The POST , GET , PUT and DELETE methods map to the CRUD methods Create, Read, Update, and Delete, respectively. **Each resource should have at least one method.**
* **Create at most only one default resource** (a resource with neither a uri-template nor a url-mapping) for each API.
* **Offer both XML,TEXT,HTML and JSON** whenever possible.
* **Version your APIs** as early as possible in the development cycle. At present, the ESB profile identifies each API by its unique context name. If you introduce a version in the API context (e.g., /Service/1.0.0), you can update it when you upgrade the same API (e.g., /Service/1.0.1).
* **Secure your services** using OAuth2, JWT, BASE 64 or another authentication/authorization, Do not use anonymous endpoints. Always use named [endpoints](https://ei.docs.wso2.com/en/latest/micro-integrator/references/synapse-properties/endpoint-properties). As anynymous endpoints have auto-generated names in the synapse configuration, it is difficult to identify which endpoint is causing the error in case of an error.

**Endpoint Configuration -**

* Configure timeout settings appropriately. Timeout configurations are required before you go into production with the system.  
  Here are the important timeout parameters you should configure before going into production:
* Set the socket timeout value and individual endpoint timeout values appropriately. Use this formula to set timeout values:

Socket Timeout > max (Global endpoint timeout, Timeout of individual endpoints)

* Be sure to set proper values to advanced configuration parameters, although they are optional.

**Behaviour of the FaultSequence** -

* If a sequence explicitly defines a fault handler using the onError attribute, WSO2 Micro Integrator invokes that specific onError sequence whenever an error occurs in the sequence. This is true even if the sequence is invoked by a [proxy service](https://ei.docs.wso2.com/en/latest/micro-integrator/references/synapse-properties/proxy-service-properties) or in an [API](https://ei.docs.wso2.com/en/latest/micro-integrator/references/synapse-properties/rest-api-properties).
* If a request arrives via the [main sequence](https://ei.docs.wso2.com/en/latest/micro-integrator/references/synapse-properties/sequence-properties) and fails within a sequence that does not explicitly define a fault handler, the default FaultSequence is invoked.

**Testing –**

**Unit Testing-**

* Use SoapUI for soap services and Rest client for Rest services for unit testing as well as performance testing.
* Create a test plan covering all functional scenarios and performance tests.
* Write a test case for all the services and make a proper document.
* Maintain health check service for all the instances need to know the status of the server.
* **Prepare a Release notes for SIT Team**

**Application Testing-**

**As per our project Scope Document OM is taking care of SIT.**

**Deployment -**

* Ensure that you tune the deployment environment based on the [performance tuning guidelines](https://ei.docs.wso2.com/en/latest/micro-integrator/setup/performance_tuning/tuning_jvm_performance) .
* Make a CI/CD plan for Automated deployment process for Upper Instances.
* Pre-test patches in a test environment before going into the production environment. You should use an automated test suite to do this. For example, you can use [JMeter Automation](http://www.testautomationguru.com/how-to-test-rest-api-using-jmeter/) or [SOAP UI](https://www.soapui.org/tutorials/rest-sample-project.html) .
* Pre-test artifacts in a test environment before deploying into the production environment.
* Pre-test configuration changes before applying them in the production environment.