Ohio MMS Services Integration Proxy Technical Design

# Introduction

## Purpose

The purpose of this Design Document is to state technical requirements, and technical solutions for Internal Service Integration Proxy.

The intended purpose of this design doc is to:

* As written consents about technical requirements and implementation between architect and development teams
* Provide development specification on how to implement the Proxy.
* Intentional designed as product solution, for generic clients, beyond OH SI.

## Scope

The scope of this document includes include all technical specifications of the proxy.

## High-level Technical Requirements

### Security/Auditing

#### Requirements:

**Secure service over HTTPS using client certificate authentication**

The service will be secured with service certificate authentication and accessible only over HTTPS.

**SOAP Header Validation**

Using SOAP headers to authenticate the client and process or reject request when they are not presented with proper values.

Each request header from client is expected to include unique Transaction ID **per request** for tracking purpose. Transaction ID will be used as logging file name.

Message Header parameters name required for Proxy should be configurable to be able to support different client/integrator.

**How to write same Transaction ID to response**

Possible solutions:

* Enable Session service in Proxy and WCF, the session ID per call to be initiated from Proxy to WCF service.

https://debugmode.net/2009/11/28/session-id-in-wcf/

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Header** | | | | |
| BUSINESSFLOW | Restriction: “SearchClaims” | 1:1 | Yes | Identifies the operation invoked. For e.g. Provider Enrollment, Update Provider etc. |
| STATECODE | Default: OH | 1:1 | Yes | Unique Code identifying the State for which the transaction applies. |
| REQUESTORSYSTEM | 50 Default: “PNM” | 1:1 | Yes | Identity of the caller system initiating this request. |
| SUBSCRIBERSYSTEM | 50 (e.g.MITS) | 0:1 | No | Identities of the receiver systems receiving this request. (e.g. MITS) |
| MODULETRANSACTIONID | 20 | 0:1 | No | OMES assigned module transaction ID (or SourceID) |
| ADDITIONALMODULETRANSACTIONID | 50 | 0:1 | No | Captures the unique transaction ID received during Pratial information exchange.  (E.g. CID123, PID123) |
| REQUESTTIMESTAMP | YYYY-MM DDTHH:MM:SS | 1:1 | Yes | Date stamp when the request was received by OMES. |
| SITRANSACTIONKEY | 32 | 0:1 | No | System Integrator Transaction Key assigned |

After validation, logging complete request and response SOAP message using [Log4Net](https://www.c-sharpcorner.com/blogs/how-to-use-log4net-in-asp-net-core-application) **File Appender** write to text file, save into shared file system for auditing purpose.

Request/Response messages should be stored as XML with following file structure and naming convention:

**Folder:**

Operation Name->Date->TransactionID

**FileName:**

<TransactionID>-IN.xml

<TransactionID>-OUT.xml

#### Implementation To inspect or modify messages

<https://docs.microsoft.com/en-us/dotnet/framework/wcf/extending/how-to-inspect-or-modify-messages-on-the-client?redirectedfrom=MSDN>

Inspect or modify the incoming or outgoing messages across a WCF client by implementing a [System.ServiceModel.Dispatcher.IClientMessageInspector](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.iclientmessageinspector) and inserting it into the client runtime. For more information, see [Extending Clients](https://docs.microsoft.com/en-us/dotnet/framework/wcf/extending/extending-clients). The equivalent feature on the service is the [System.ServiceModel.Dispatcher.IDispatchMessageInspector](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.idispatchmessageinspector). For a complete code example see the [Message Inspectors](https://docs.microsoft.com/en-us/dotnet/framework/wcf/samples/message-inspectors) sample.

1. Implement the [System.ServiceModel.Dispatcher.IClientMessageInspector](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.iclientmessageinspector) interface.
2. Implement a [System.ServiceModel.Description.IEndpointBehavior](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.description.iendpointbehavior) or [System.ServiceModel.Description.IContractBehavior](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.description.icontractbehavior) depending upon the scope at which you want to insert the client message inspector. [System.ServiceModel.Description.IEndpointBehavior](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.description.iendpointbehavior) allows you to change behavior at the endpoint level. [System.ServiceModel.Description.IContractBehavior](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.description.icontractbehavior) allows you to change behavior at the contract level.
3. Insert the behavior prior to calling the [ClientBase<TChannel>.Open](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.clientbase-1.open) or the [ICommunicationObject.Open](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.icommunicationobject.open) method on the [System.ServiceModel.ChannelFactory<TChannel>](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channelfactory-1). For details, see [Configuring and Extending the Runtime with Behaviors](https://docs.microsoft.com/en-us/dotnet/framework/wcf/extending/configuring-and-extending-the-runtime-with-behaviors).

### Divert/Dispatch operation

#### Requirements

**Dispatch operation:** After validation, basic on BUSINESSFLOW value from request header, proxy will dispatch operations in various scenarios:

* Pass thru operation directly to corresponding downstream WCF services operation.
* Consolidate operations. In the case of sequencing operations needed for some business functions, Proxy functions as aggregator to handle sequencing operations internally, reduce round-trip between client and downstream services, and reduce complexity of operations.

#### Implement with Custom Message Filter

https://docs.microsoft.com/en-us/dotnet/framework/wcf/samples/custom-message-filter

When the first message on a channel arrives at the server, the server must determine which (if any) of the endpoints associated with that URI should receive the message. This process is controlled by the [MessageFilter](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.messagefilter) objects attached to the [EndpointDispatcher](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.endpointdispatcher).

Each endpoint of a service has a single [EndpointDispatcher](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.endpointdispatcher). The [EndpointDispatcher](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.endpointdispatcher) has both an [AddressFilter](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.endpointdispatcher.addressfilter) and a [ContractFilter](https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.dispatcher.endpointdispatcher.contractfilter). The union of these two filters is the message filter used for that endpoint.

### Monitoring downstream services

Implement backend service run periodically check downstream services status. Send out notification if any service down/unhealth.