UPLOADING A FILE FROM PO TO AWS s3 WITHOUT SIZE RESTRICTIONS

One of the many challenges that we face in On-Premise to Cloud migration is the difference in approach to integration. While solutions on the cloud have a great flexibility in making messages as granular as possible, on-premise solutions are bound to restricting factors like bandwidth, memory and message sizes. As a basic premise to SAP PO development, our aim as developers is to optimize the many factors of an integration requirement, message size being very important.

This document is aimed at providing one of the many solutions that we can adapt to transfer a file from PO to AWS s3 bucket without us having to worry about the size of the file. AWS has a list of APIs available for us to use to upload files into the s3 bucket. As mentioned above, AWS doesn't have a restriction on the size of the file received, but this issue is prominent in PO. You can find details of AWS API details here - https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html

Amazon gives us 3 options to upload files into s3 via APIs

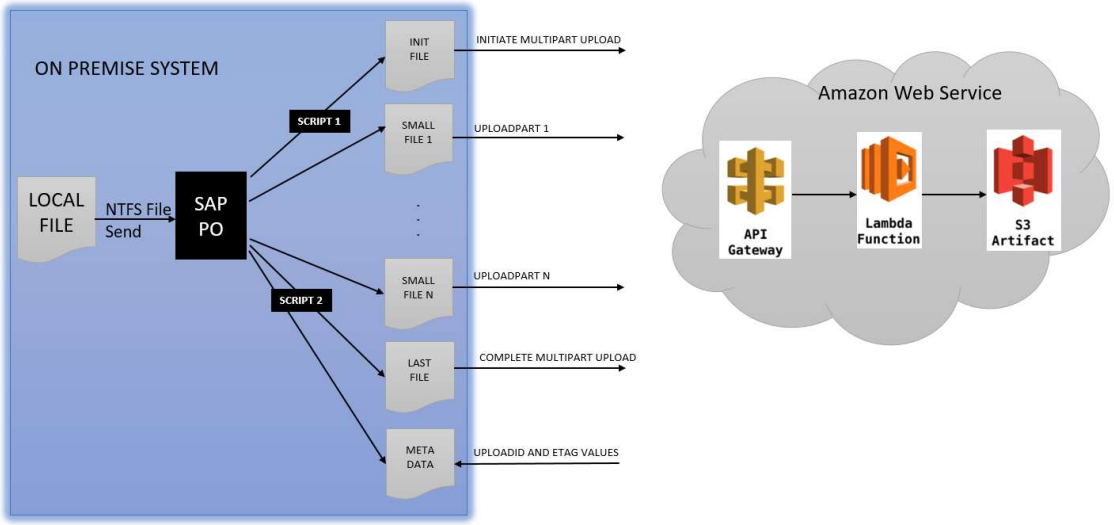
Single Chunk Upload - https://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-header-based-auth.html

Multi Chunk Upload - https://docs.aws.amazon.com/AmazonS3/latest/API/sigv4-streaming.html

Multi Part Upload - https://docs.aws.amazon.com/AmazonS3/latest/dev/mpuoverview.html

We had a file size restriction of ~900KB which wasn't very promising. Amazon's Single Chunk Upload was the best option. This is covered extensively in Rajesh PS's blog - https://blogs.sap.com/2019/05/31/integrating-amazon-simple-storage-service-amazon-s3-and-sap-ecc-v6.0-via-sap-pi-v7.5-using-aws-signature-v5-and-signing-algorithm-hmac-sha256/

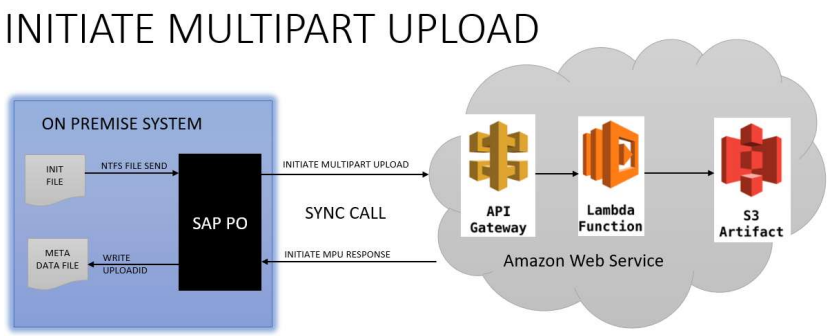
Multi-chunk upload was not feasible for PO since it requires an HTTP connection to be open until all the chunks have been transferred. That leaves us with the Multipart Upload option. I will attempt to cover how I transferred a huge file using multipart upload. The basic design is as follows:

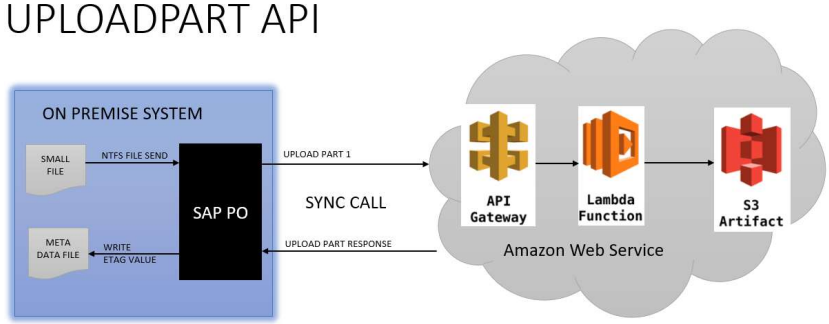


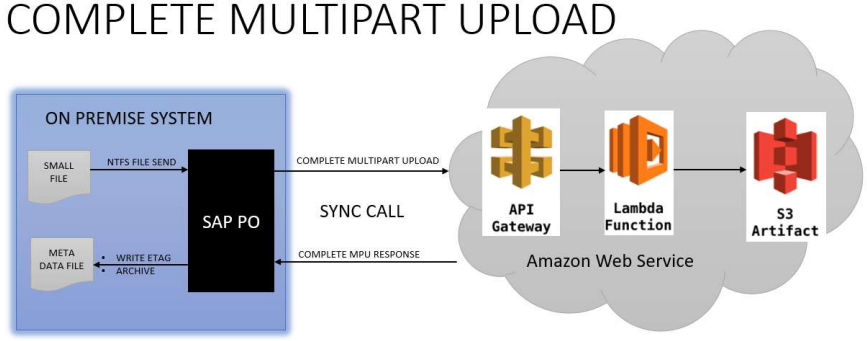
AWS MULTIPART UPLOAD APIs

1. Multipart Upload works by initiating an API. AWS returns us an UploadId. The smaller files are then sent one by one with the UploadId and part number. AWS allows for parallel sending of the multiple parts, but we will need to know the sequence number beforehand, which is not possible in this design. AWS returns an ETag for each file uploaded. After all the parts have been uploaded, you will then need to call the Complete Multipart Upload API with all the ETags and the Part Number. This enables AWS to 'stitch' back the multiple small files into one large file based on the details sent. I've had to develop the AWS signature from scratch: https://github.com/pnathan01/AWSSAPMultipartUploadLibrary. I also took inspiration from Rajesh's blog for single chunk upload(link in the last section)

The following is a diagrammatical representation of each of the 3 types of calls made to s3 API.







IMPLEMENTATION IN PO

As you can see all the above communication are Async-Sync bridges. I have created 4 ICOs – 1 for Request with routing to different APIs and 3 different ICOs for the response part. All 3 of the response ICOs will write into the metadata file.

Please note that I have not accounted for any data conversion, format checks or mapping of the data in the file. Technically they are pass through scenarios, except Complete Multipart Upload, but we have to use mappings to calculate AWS API signatures.

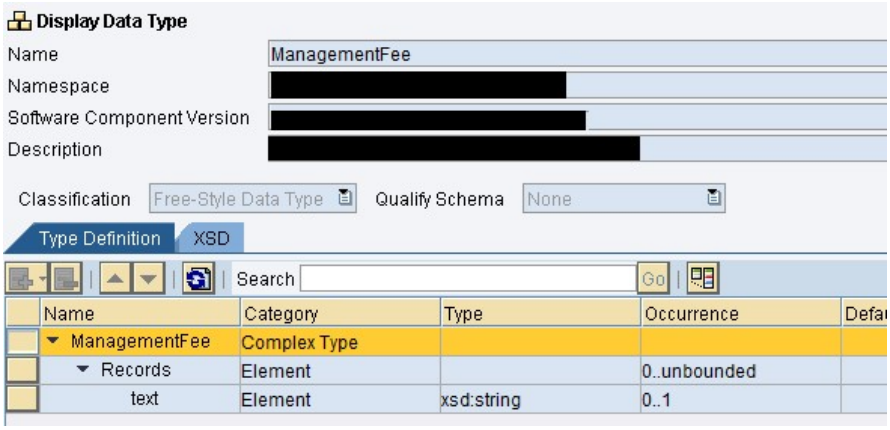
The following configuration will need to be done beforehand:

1. Implement certificates, if any
2. Enable large message handling. This is because AWS requires all smaller files of a multipart upload, except the last one, to be 5 MB or larger.

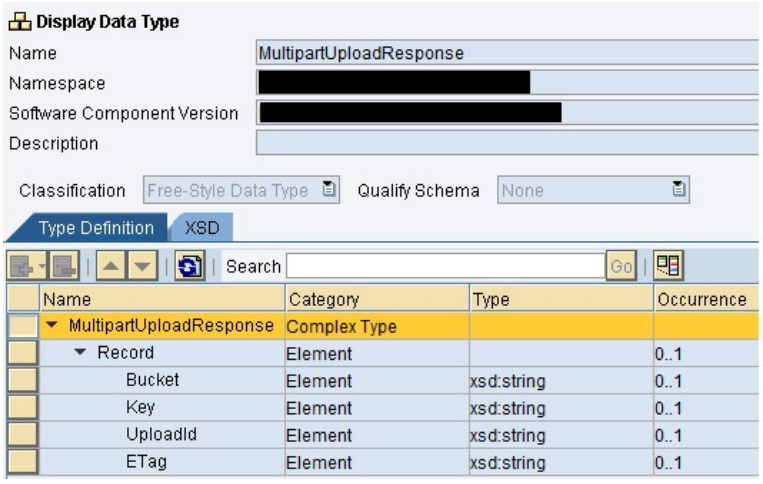
Create ESR Objects

We have 2 different software components for source and target systems. So this scenario is from Software Component 1 to Software Component 2. Also, we will be using a common structure to write the metadata content like UploadIDs and ETags into the metadata file.

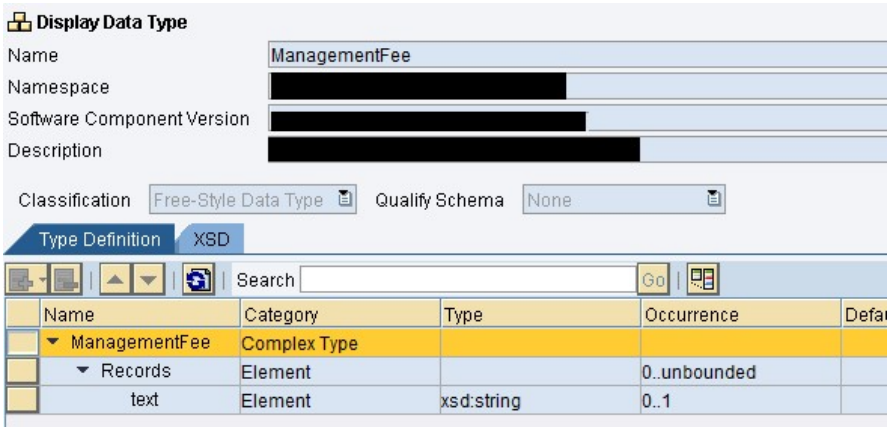
1. Data Types
    a. Request Data Types in both software components.

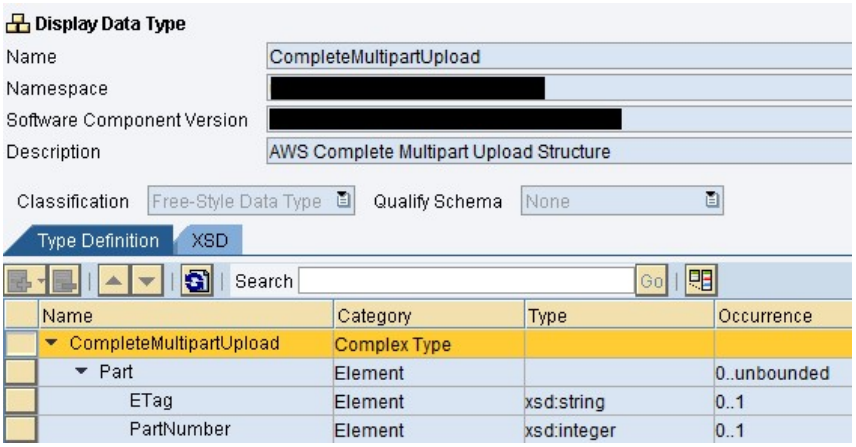


    b. Response Data Type in Software Component 1

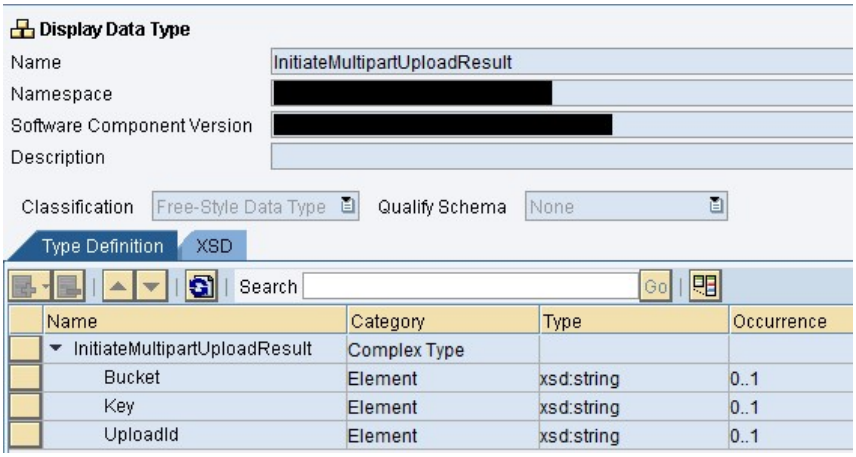


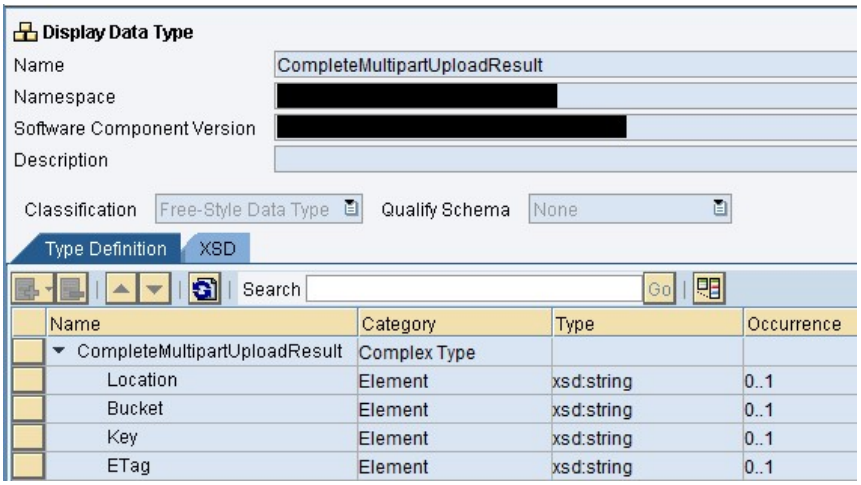
c. Request Data Type in Software Component 2. CompleteMultipartUpload Data Type will not be a pass through mapping because there is actual data that is required to be sent.



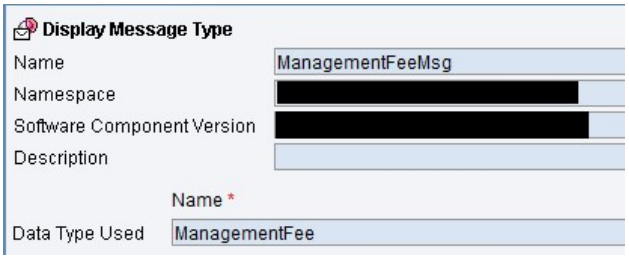


d. Response Data Type in Software Component 2. The InitiateMultipartUploadResult data type is used to capture the response of both Initiate MPU and Upload Part because the UploadID and ETags are part of the header.

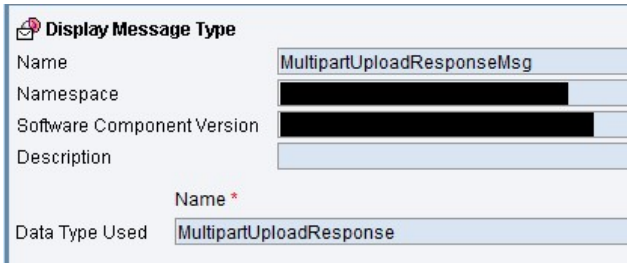




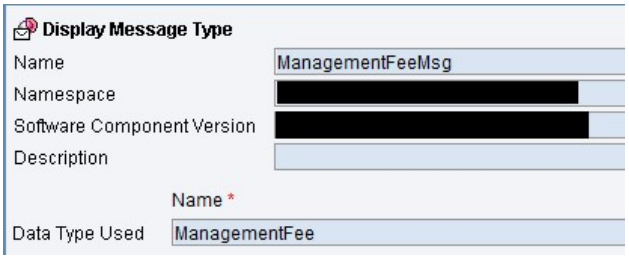
2. Message Types
    a. Request Message Type in Software Component 1

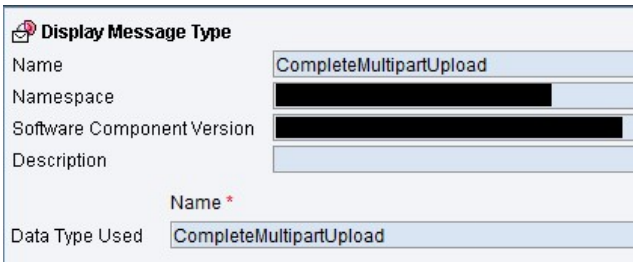


    b. Response Message Type in Software Component 1



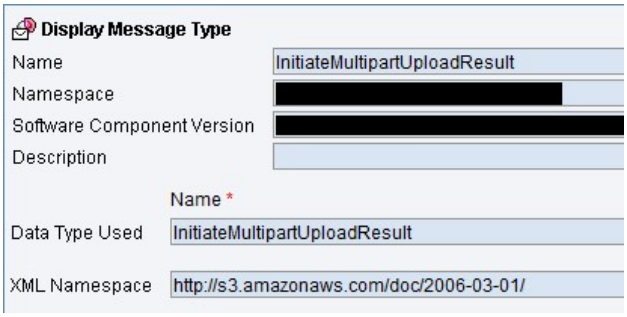
    c. Request Message Type in Software Component 2

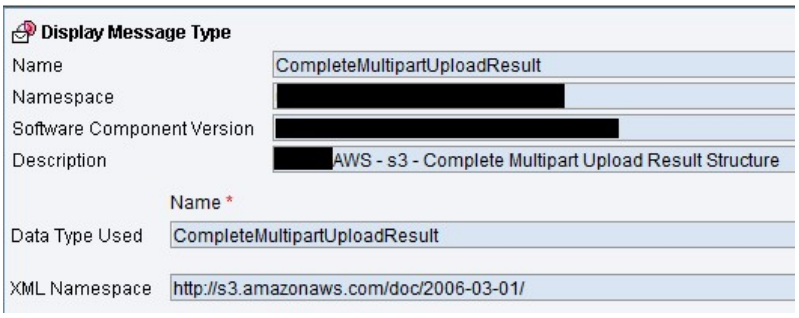




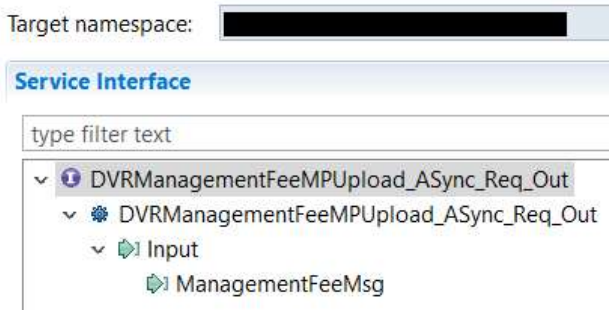
d. Response Message Type in Software Component 2 - I've changed the namespace for these structures because I didn't want to develop code/adapter module to handle this separately.

Just like the data type, this message type is common for both Initiate MPU and UploadPart API Response structures.

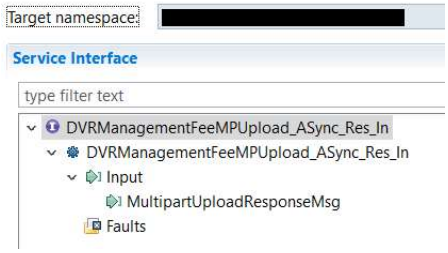




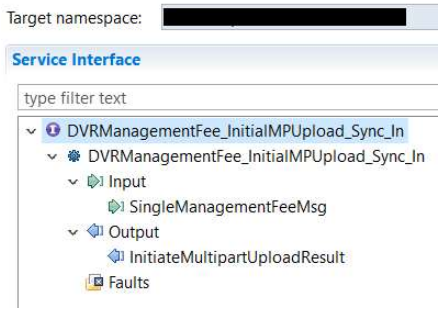
3. Service Interfaces
   a. One outbound service for all requests from Software Component 1

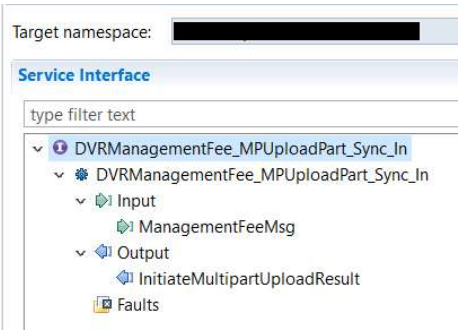


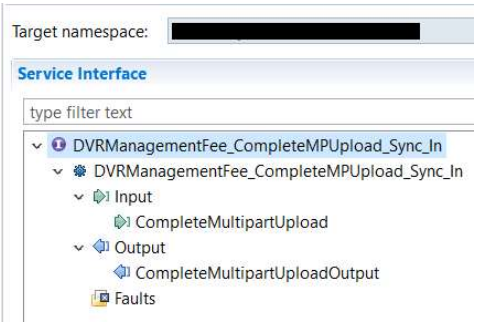
b. One inbound service for all responses from Software Component 2



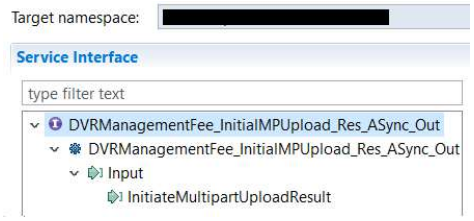
c. 3 Synchronous inbound services to send the request messages

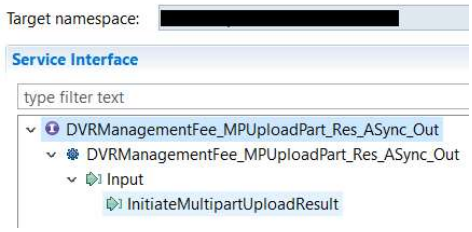


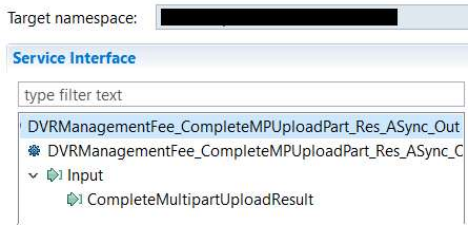




d. 3 outbound asynchronous services to receive the response – You will notice that we have separate services for InitialMPU and UploadPart even though the structure is the same. This is because the mappings for them are different.

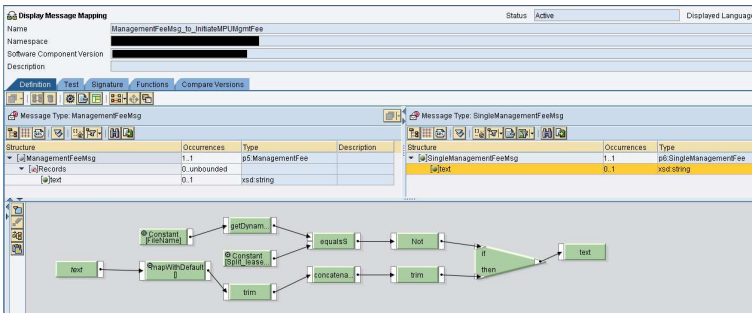




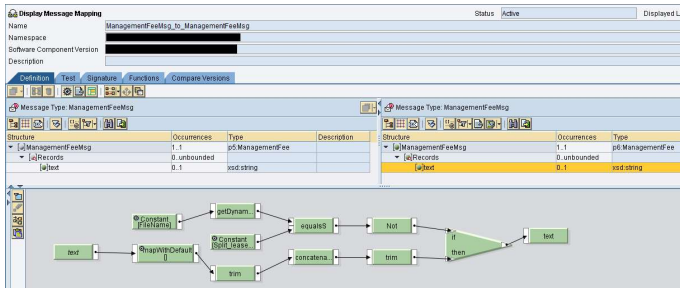


2. Mapping Programs – I've added a parameter called action because I reuse the same library code for both single chunk and multipart upload. The difference is required to calculate the AWS signature for the payload. The mappings themselves are pretty straightforward except for the signature calculation, which is handled entirely in the library. Code for AWS Signature Calculation: https://github.com/pnathan01/AWSSAPMultipartUploadLibrary

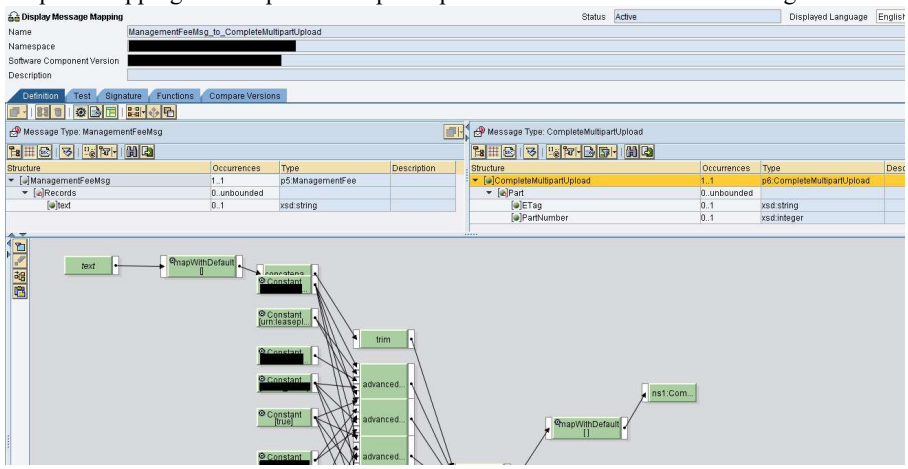
e. Request Mapping – Initiate Multipart Upload.



f. Request Mapping – Upload Part



g. Request Mappings – Complete Multipart Upload – Conversion and Formatting



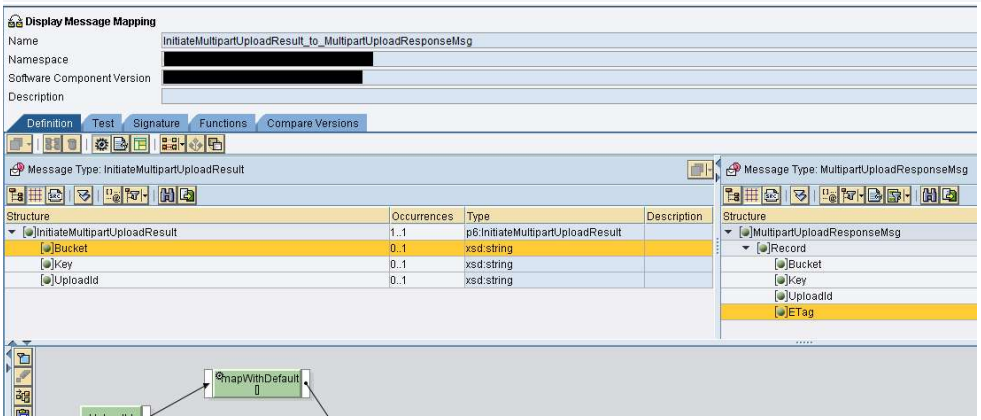

```
public void transform(TransformationInput transformationInput, TransformationOutput transformationOutput) throws StreamTransformationException {
    try {
     InputStream inputstream = transformationInput.getInputPayload().getInputStream();
     OutputStream outputstream = transformationOutput.getOutputPayload().getOutputStream();
     byte[] b = new byte[inputstream.available()];
     inputstream.read(b);
     String input = new String(b);
     input = input.replaceAll("&quot;", "\"");
     input = input.replaceAll("ns1:", "");
     input = input.replaceAll(" xmlns:ns1=\"urn:leaseplan.com:au:xi:unifii:dvr\"", "");
     String splitStr [] = input.split("<CompleteMultipartUpload");
     input = "<CompleteMultipartUpload"+splitStr[1];
     TransformerFactory tFactory = TransformerFactory.newInstance();
     Transformer transformer = tFactory.newTransformer();
     transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,"yes.");
     outputstream.write(input.getBytes());
    } catch (Exception exception) {
     getTrace().addDebugMessage(exception.getMessage());
     throw new StreamTransformationException(exception.toString());
    }
   }
```

h. Response Mapping - Initiate Multipart Upload – Java code to handle empty message

Display Message Mapping

Name: InitiateMulitpartUploadResult_Namespace_Mapping

```
/**
* You can create source text for the class body in area 'Attributes and Methods'.
* Exceptions are import statements; you must enter these in the import table. You
* can use the inner classes, methods, and attributes defined here in the user-defined
* functions and in the methods "init, cleanUp"
* You can also use classes from imported archives that are in the same software
* component version as the function library, or in a sub software component version.
* You must specify the imported archives in table 'Archives Used'
*/
public void transform(TransformationInput transformationInput, TransformationOutput transformationOutput) throws StreamTransformationException {
    try {
     InputStream inputstream = transformationInput.getInputPayload().getInputStream();
     OutputStream outputstream = transformationOutput.getOutputPayload().getOutputStream();

     byte[] b = new byte[inputstream.available()];
     inputstream.read(b);
     String encoding ="UTF-8";
     String  inputXML= "";
     String input = new String(b);
     getTrace().addInfo("input"+input);
     input = input.replaceAll("<InitiateMultipartUploadResult", "<ns0:InitiateMultipartUploadResult");
     input = input.replaceAll("xmlns=", "xmlns:ns0=");
     input = input.replaceAll("</InitiateMultipartUploadResult", "</ns0:InitiateMultipartUploadResult");
     outputstream.write(input.getBytes());
    } catch (Exception exception) {
     getTrace().addDebugMessage(exception.getMessage());
     throw new StreamTransformationException(exception.toString());
    }
}
```

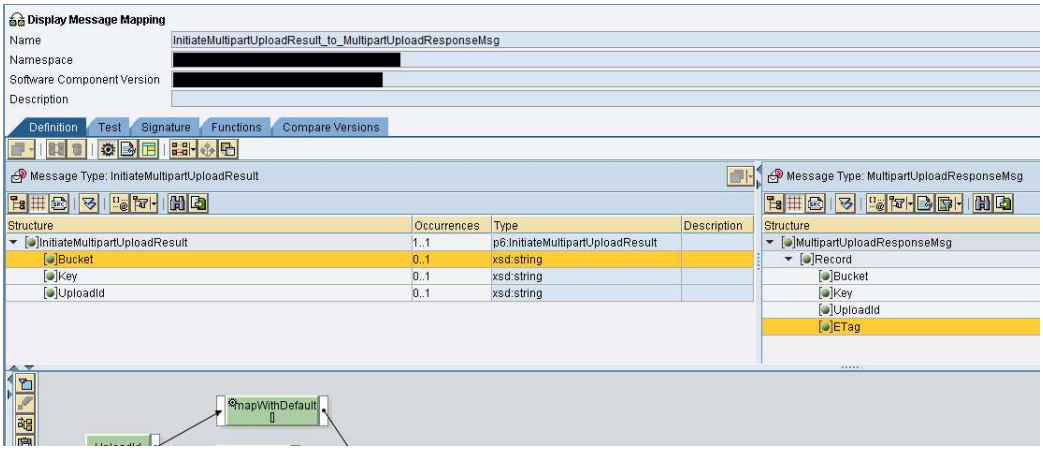


i. Response Mapping – Upload Part – Java code to handle empty message received.

Display Message Mapping

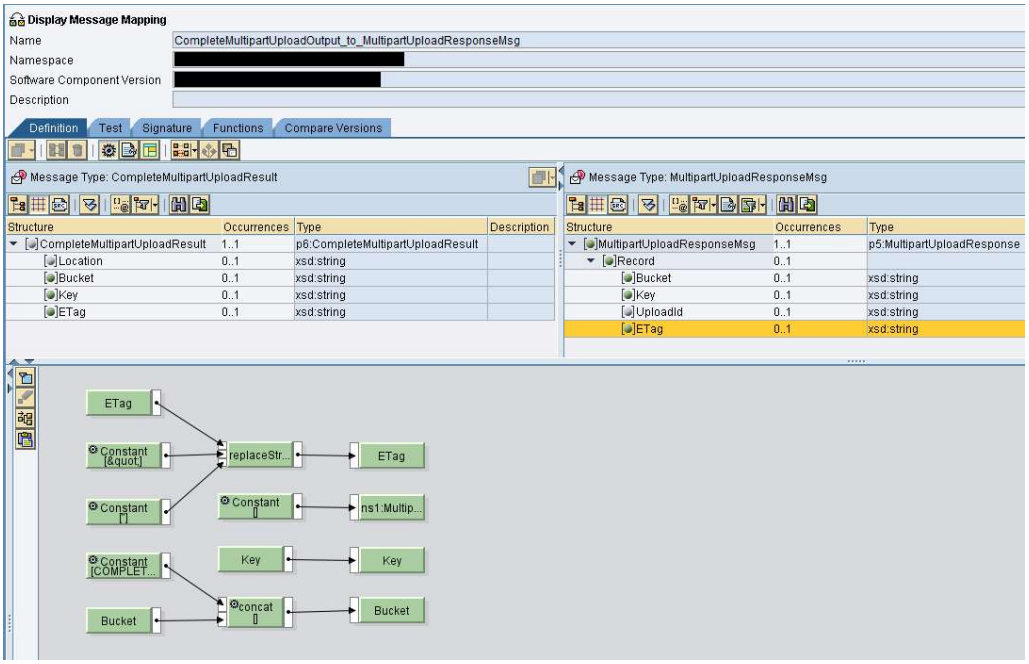
Name: Empty_InitiateMultipartUploadResult_Mapping

```
public void transform(TransformationInput transformationInput, TransformationOutput transformationOutput) throws StreamTransformationException {
    try {
     InputStream inputstream = transformationInput.getInputPayload().getInputStream();
     OutputStream outputstream = transformationOutput.getOutputPayload().getOutputStream();
     String input = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"+"\n" +
               "<ns0:InitiateMultipartUploadResult xmlns:ns0=\"http://s3.amazonaws.com/doc/2006-03-01/\">"+"\n" +
               "<Bucket/>"+"\n" +
               "<Key/>"+"\n" +
               "<UploadId/>"+"\n" +
               "</ns0:InitiateMultipartUploadResult>";
     outputstream.write(input.getBytes());
    } catch (Exception exception) {
     getTrace().addDebugMessage(exception.getMessage());
     throw new StreamTransformationException(exception.toString());
    }
}
```

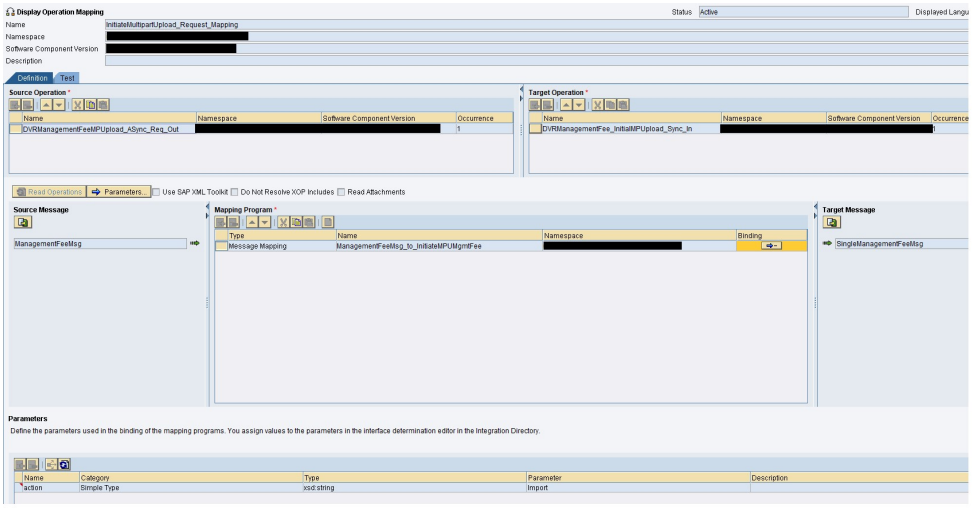


j. Response Mapping – Complete Multipart Upload. The response is namespace-agnostic, causing issues in PI.

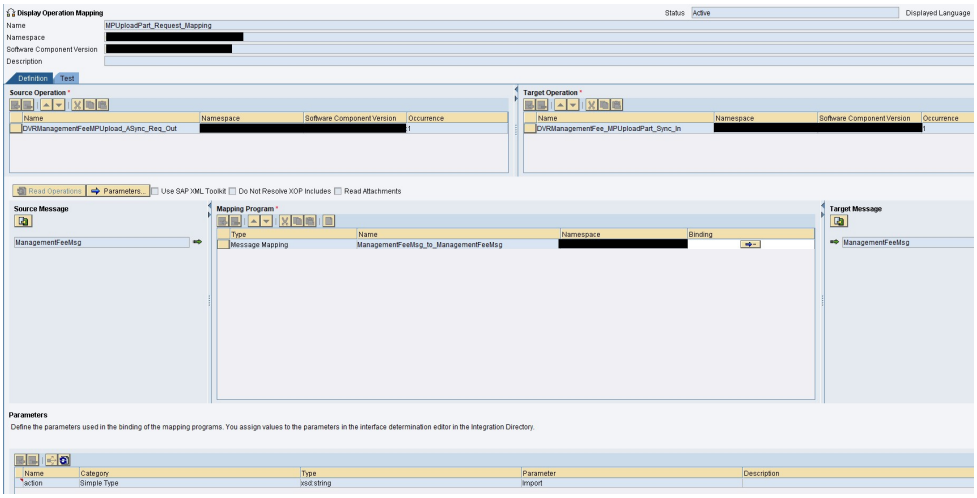
```
public void transform(TransformationInput transformationInput, TransformationOutput transformationOutput) throws StreamTransformationException {
   try {
     InputStream inputstream = transformationInput.getInputPayload().getInputStream();
     OutputStream outputstream = transformationOutput.getOutputPayload().getOutputStream();

     byte[] b = new byte[inputstream.available()];
     inputstream.read(b);
     String encoding ="UTF-8";
     String  inputXML= "";
     String input = new String(b);
     getTrace().addInfo("input"+input);
     input = input.replaceAll("xmlns=", "xmlns:ns0=");
     input = input.replaceAll("CompleteMultipartUploadResult","ns0:CompleteMultipartUploadResult");
     input = input.replaceAll("&quot;", "\"");
//   input = input.replaceAll("CompleteMultipartUploadOutput>","</ns0:CompleteMultipartUploadOutput>");
     outputstream.write(input.getBytes());
    } catch (Exception exception) {
     getTrace().addDebugMessage(exception.getMessage());
     throw new StreamTransformationException(exception.toString());
    }
  }
```



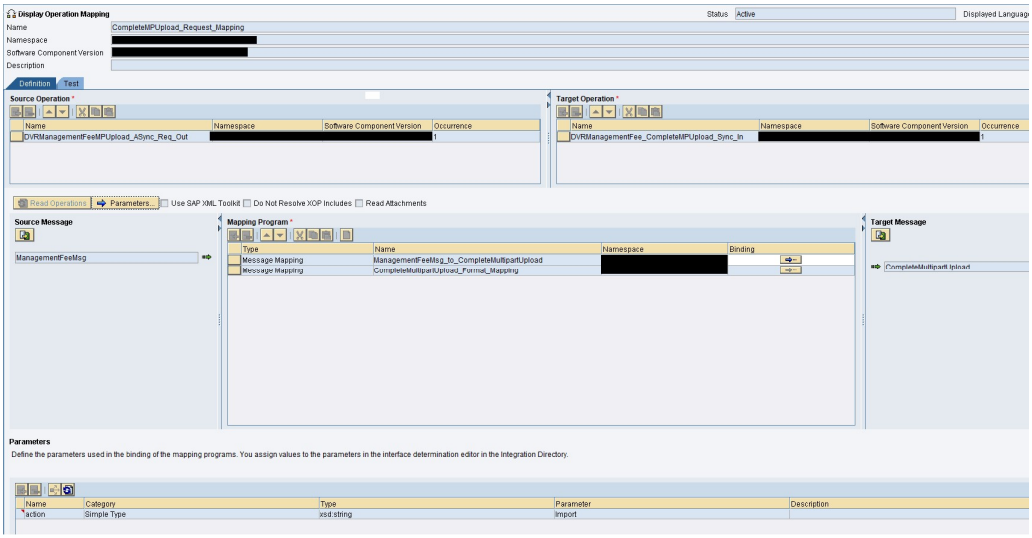
4. Operation Mappings
    a. Request OM – Initiate Multipart Upload



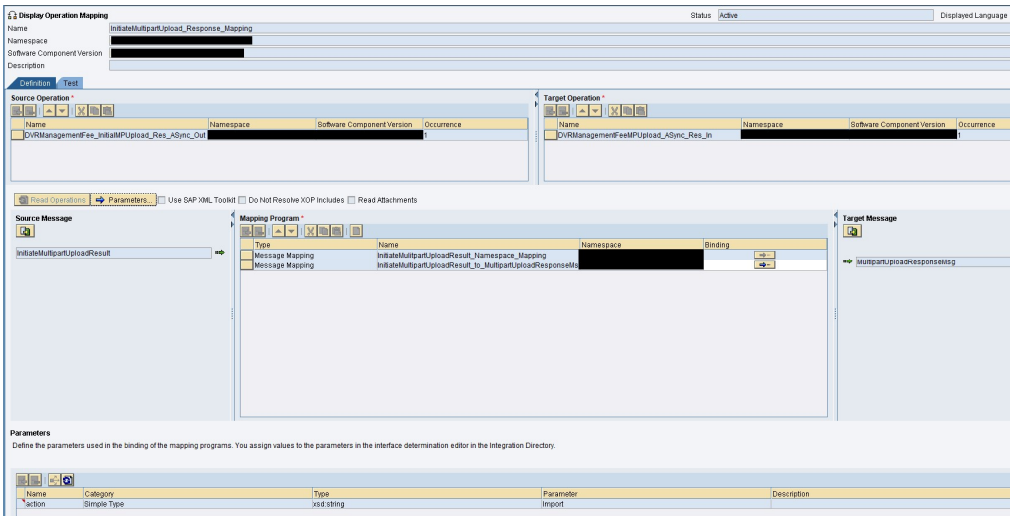
    b. Request OM – Upload Part



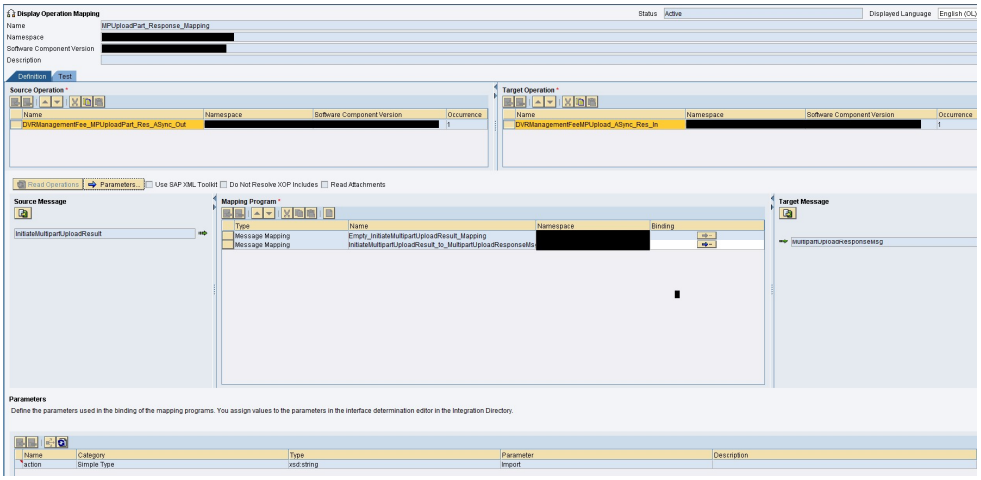
c. Request OM – Complete Multipart Upload



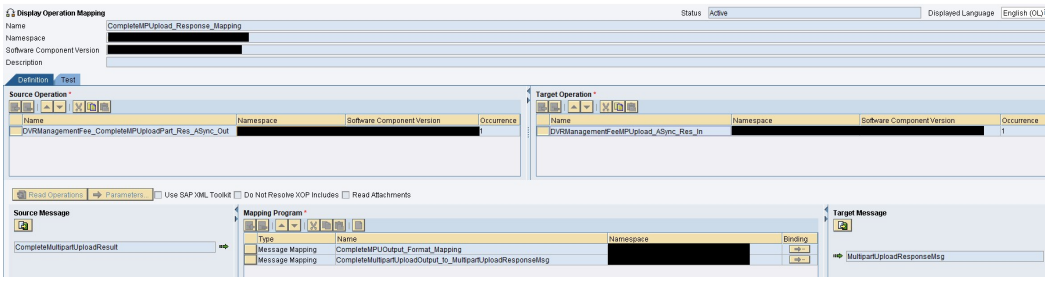
d. Response OM - Initiate Multipart Upload



e. Response OM – Upload Part



f. Response OM – Complete Multipart Upload



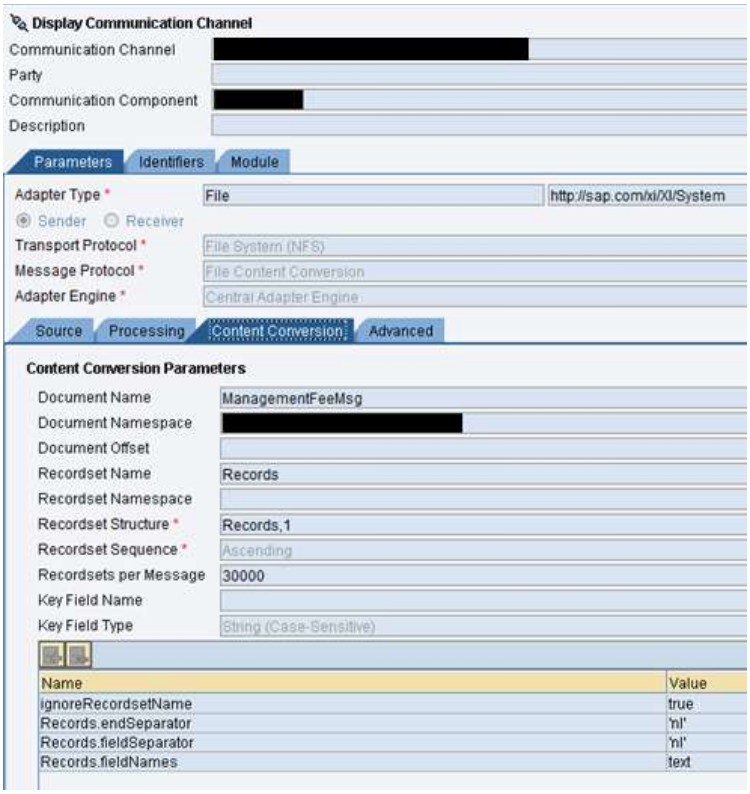
Create ID Objects

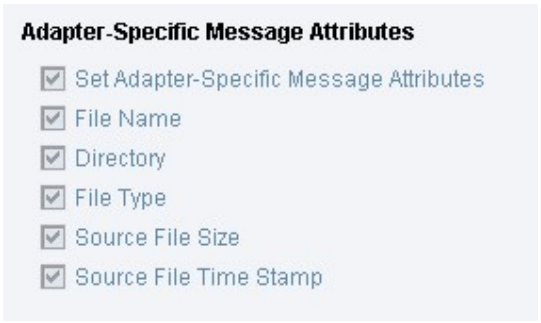
We will create the following ID objects

- 1 ICO for splitting the files
- 1 ICO for sending out the request part of the ASync-Sync bridge
- 3 ICOs for receiving the response

1. File Split Scenario

    a. Sender Channel – The file is split based on number of records because AWS MPU has a condition that all multipart files except the last one be larger than 5 MB.

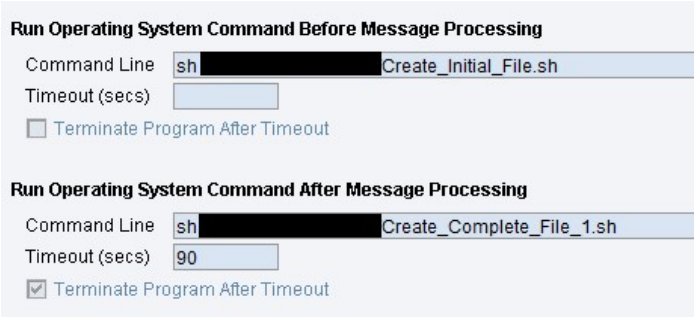




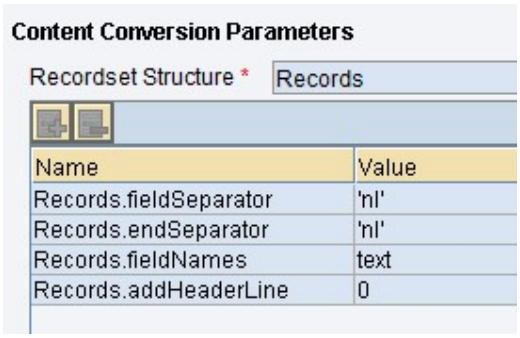
Here I call 2 scripts, before and after the message processing. The first one creates a file of the format <FILE_NAME>_00000000-000000-000.txt. This will be the first file called which will trigger the Initiate Multipart Upload.

The second script creates the last file of the format <FILE_NAME>_99991231-235959-000.txt and Temp_Metadata.txt. This file is the one to initiate the Complete Multipart Upload. The metadata file is the one that will hold all the UploadIDs and ETags for logging, audit and monitoring purposes.

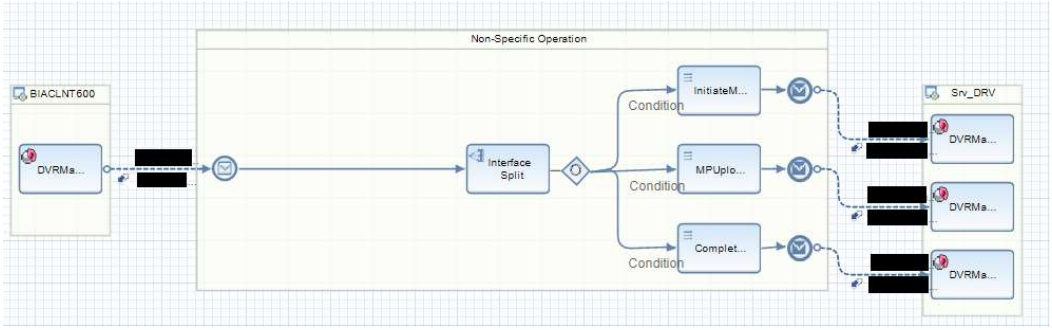
I haven't provided the script code because it is simple, and it is dependent on your OS.

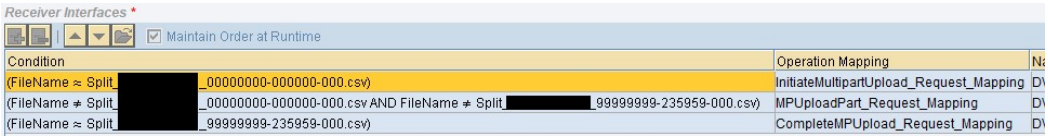


b. Receiver Channel

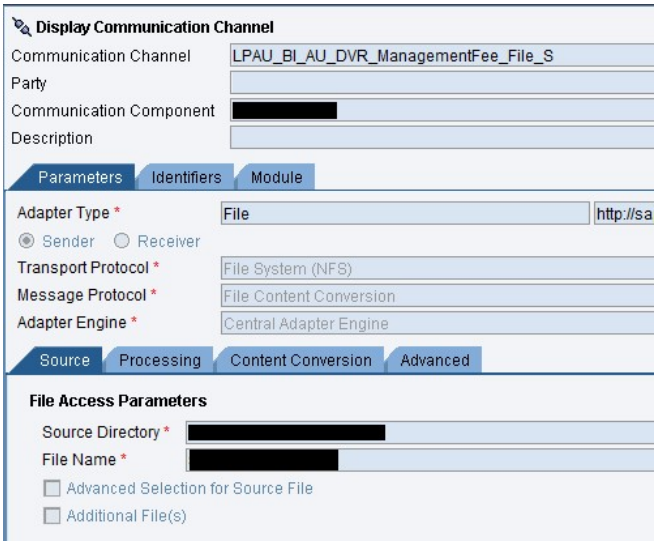


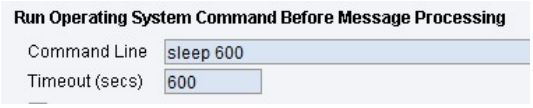
2. Request Send Scenario
   a. ICO

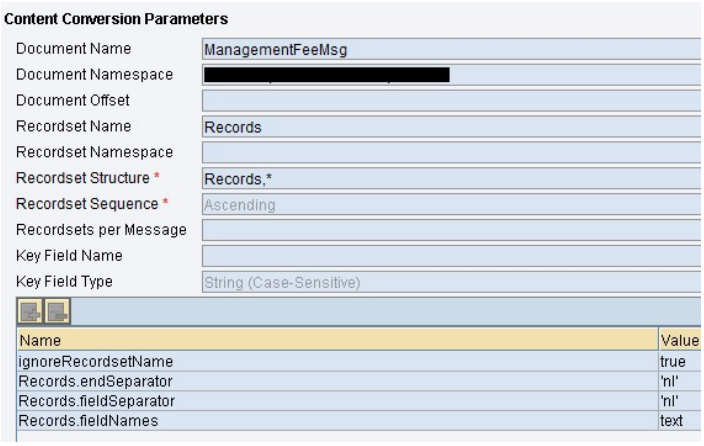




b. Sender Channel

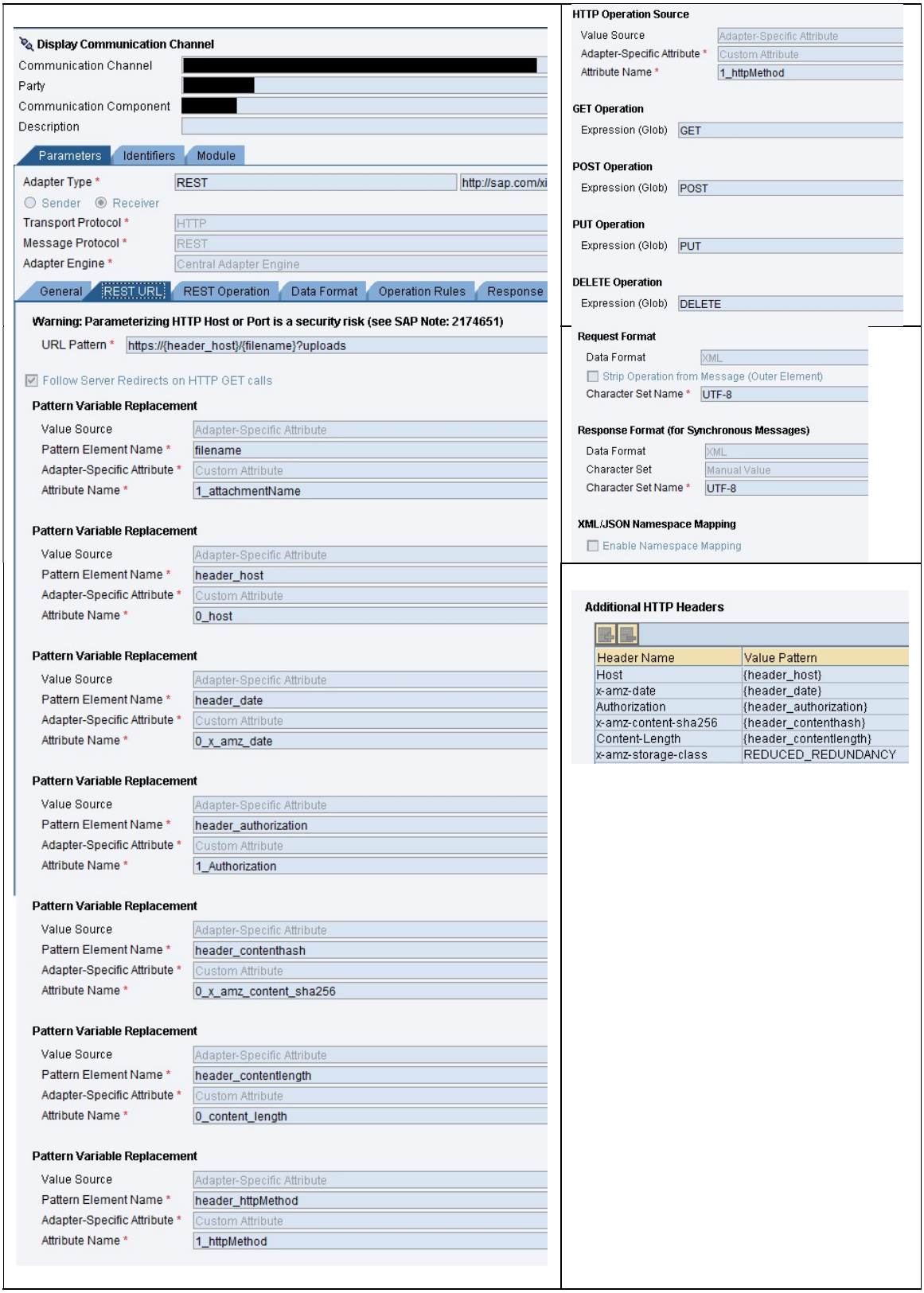






c. Receiver Channels

Channel 1 – Request to Initiate Multipart Upload



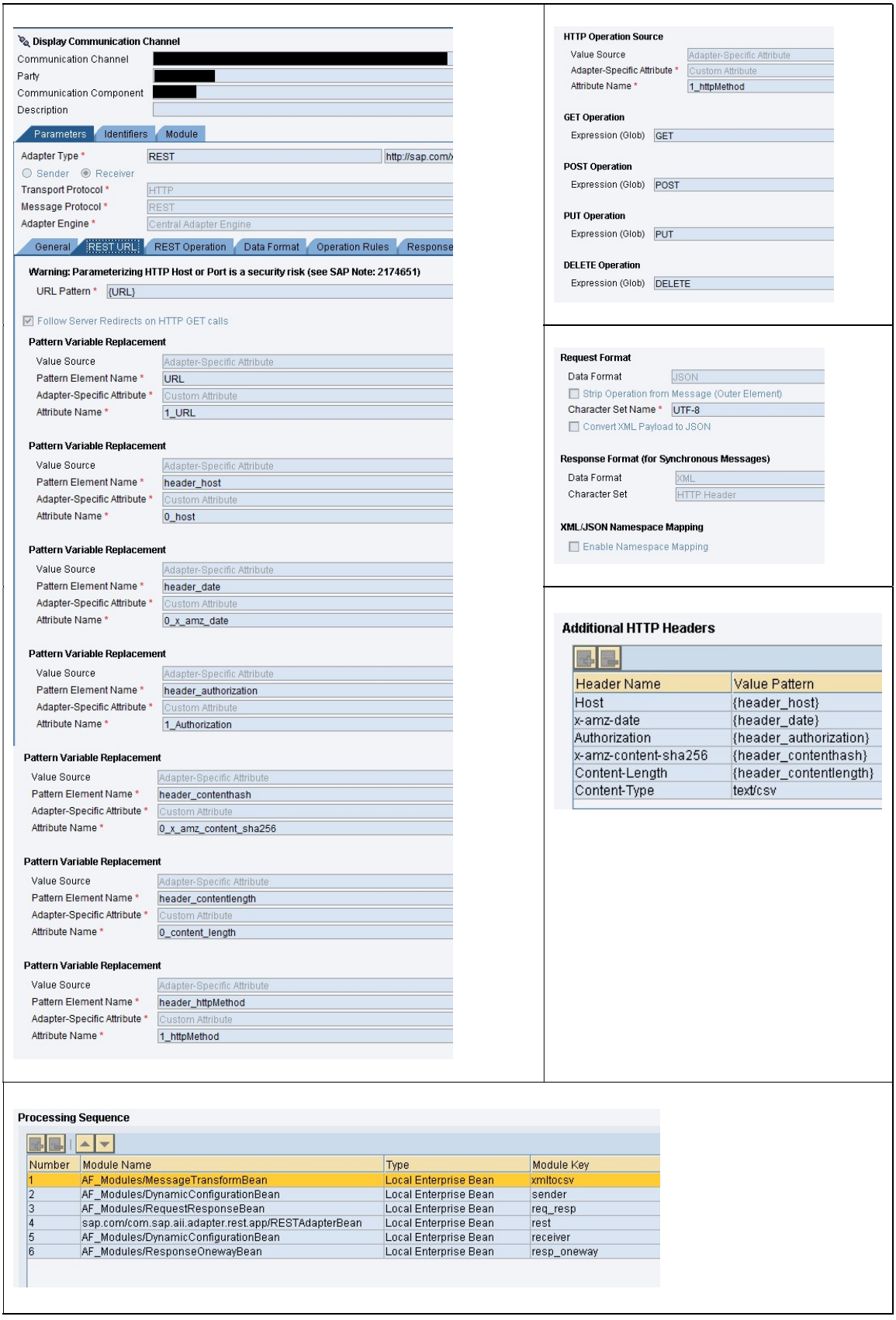
Processing Sequence

Number	Module Name	Type	Module Key
1	AF_Modules/StrictXml2PlainBean	Local Enterprise Bean	transform
2	AF_Modules/DynamicConfigurationBean	Local Enterprise Bean	sender
3	AF_Modules/RequestResponseBean	Local Enterprise Bean	req_resp
4	ReplaceString	Local Enterprise Bean	replace
5	sap.com/com.sap.aii.adapter.rest.app/RESTAdapterBean	Local Enterprise Bean	rest
6	AF_Modules/DynamicConfigurationBean	Local Enterprise Bean	receiver
7	AF_Modules/ResponseOnewayBean	Local Enterprise Bean	resp_oneway

Module Configuration

Module Key	Parameter Name	Parameter Value
receiver	key.1	read http://sap.com/xi/XI/System/REST 1_fileName
receiver	value1	module.http://sap.com/xi/XI/System/REST
replace	param1	null;<?xml version=\"1.0\" encoding=\"UTF-8\"?> <ns0:InitiateMulti
replace	separator	;
req_resp	passThrough	true
resp_oneway	interface	DVRManagementFee_InitialMPUpload_Res_ASync_Out
resp_oneway	interfaceNamespace	[redacted]
resp_oneway	replaceInterface	true
rest	setIgnoreEmptyArrayValues	false
rest	setIgnoredElements	ManagementFeeMsg
sender	key.1	write http://sap.com/xi/XI/System/REST 1_fileName
sender	key.2	write http://sap.com/xi/XI/System/REST Content-Type
sender	value.1	module.http://sap.com/xi/XI/System/REST
sender	value.2	text/csv;charset=UTF-8
transform	Records.endSeparator	'nl
transform	Records.fieldNames	text
transform	Records.fieldSeparator	'nl'
transform	Transform.ContentType	text/plain
transform	recordTypes	Records

Channel 2 – Request to Upload Part



Module Configuration

Module Key	Parameter Name	Parameter Value
receiver	key.1	read http://sap.com/xi/XI/System/REST 1_fileName
receiver	value.1	module.http://sap.com/xi/XI/System/REST
req_resp	passThrough	true
resp_oneway	interface	DVRManagementFee_MPUploadPart_Res_ASync_Out
resp_oneway	interfaceNamespace	[redacted]
resp_oneway	replaceInterface	true
rest	xml.processFieldNames	fromConfiguration
sender	key.1	write http://sap.com/xi/XI/System/REST 1_fileName
sender	value.1	module.http://sap.com/xi/XI/System/REST
xmltocsv	Transform.Class	com.sap.aii.messaging.adapter.Conversion
xmltocsv	Transform.ContentType	text/csv;charset=utf-8
xmltocsv	xml.addHeaderLine	0
xmltocsv	xml.conversionType	SimpleXML2Plain
xmltocsv	xml.documentName	ManagementFeeMsg
xmltocsv	xml.documentNamespace	urn:leaseplan.com:au:xi:unifii:dvr
xmltocsv	xml.fieldSeparator	'nl'
xmltocsv	xml.recordsetStructure	Records

Channel 3 – Request to Complete Multipart Upload

HTTP Operation Source

Value Source	Adapter-Specific Attribute
Adapter-Specific Attribute *	Custom Attribute
Attribute Name *	1_httpMethod

GET Operation

Expression (Glob)	GET

POST Operation

Expression (Glob)	POST

PUT Operation

Expression (Glob)	PUT

DELETE Operation

Expression (Glob)	DELETE

Request Format

Data Format	JSON
Strip Operation from Message (Outer Element)	
Character Set Name *	UTF-8
Convert XML Payload to JSON	

Response Format (for Synchronous Messages)

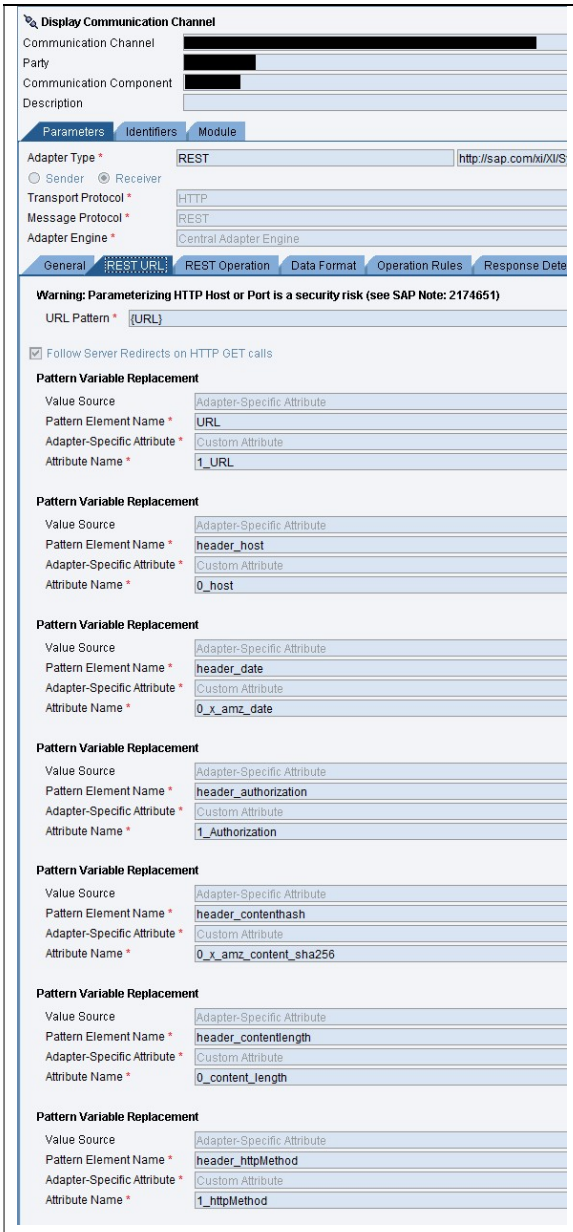
Data Format	XML
Character Set	HTTP Header

XML/JSON Namespace Mapping

Enable Namespace Mapping

Additional HTTP Headers

Header Name	Value Pattern
x-amz-date	{header_date}
Authorization	{header_authorization}
x-amz-content-sha256	{header_contenthash}
Host	{header_host}
Content-Length	{header_contentlength}
Content-Type	application/xml



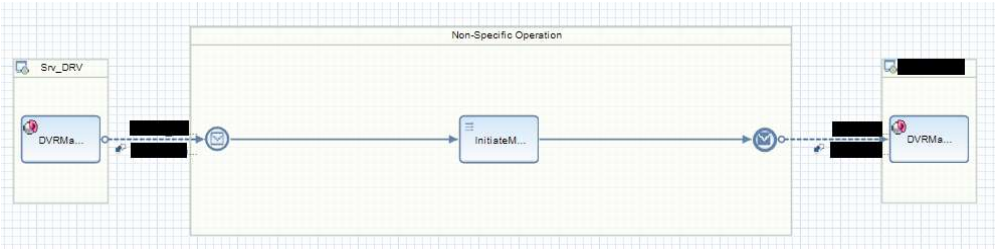
Processing Sequence

Number	Module Name	Type	Module Key
1	localejbs/ReplaceString	Local Enterprise Bean	replace
2	AF_Modules/DynamicConfigurationBean	Local Enterprise Bean	sender
3	AF_Modules/RequestResponseBean	Local Enterprise Bean	req_resp
4	sap.com/com.sap.aii.adapter.rest.app/RESTAdapterBean	Local Enterprise Bean	rest
5	AF_Modules/DynamicConfigurationBean	Local Enterprise Bean	receiver
6	AF_Modules/ResponseOnewayBean	Local Enterprise Bean	resp_oneway

Module Configuration

Module Key	Parameter Name	Parameter Value
receiver	key.1	read http://sap.com/xi/XI/System/REST 1_fileName
receiver	value.1	module.http://sap.com/xi/XI/System/REST
replace	param2	ns0:#emptyString
replace	separator	#
req_resp	passThrough	true
resp_oneway	interface	DVRManagementFee_CompleteMPUploadPart_Res_ASync_Out
resp_oneway	interfaceNamespace	[redacted]
resp_oneway	replaceInterface	true
sender	key.1	write http://sap.com/xi/XI/System/REST 1_fileName
sender	value.1	module.http://sap.com/xi/XI/System/REST

3. Response Receive Scenario – Initiate Multipart Upload
   a. ICO



   b. Sender Channel

Display Communication Channel

Communication Channel	[redacted]
Party	[redacted]
Communication Component	[redacted]
Description	

Parameters | Identifiers | Module

Adapter Type *	REST http://sap.com/xi/XI/Systen
	Sender / Receiver
Transport Protocol *	HTTP
Message Protocol *	REST
Adapter Engine *	Central Adapter Engine

General | Channel Selection | REST Resources | REST Operation | Operation Determination

Input Message Format

Data Format	XML
Character Set	HTTP Content-Type Header

Quality of Service

Quality of Service	Exactly Once

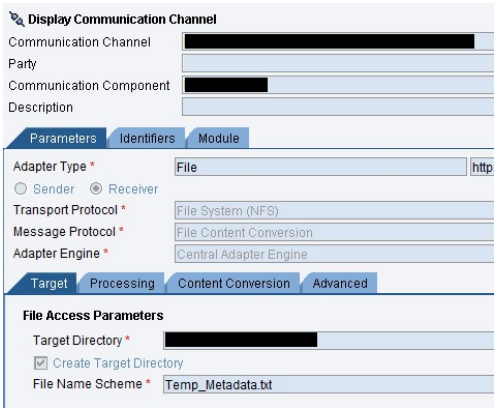
Processing Sequence

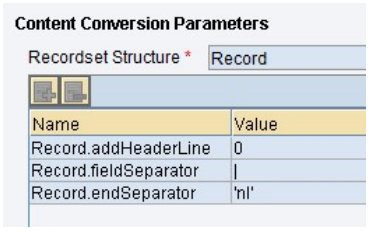
Number	Module Name	Type	Module Key
1	AF_Modules/XMLAnonymizerBean	Local Enterprise Bean	0
2	CallSapAdapter	Local Enterprise Bean	rest

Module Configuration

Module Key	Parameter Name	Parameter Value
0	anonymizer.acceptNamespaces	http://s3.amazonaws.com/doc/2006-03-01/ ns0

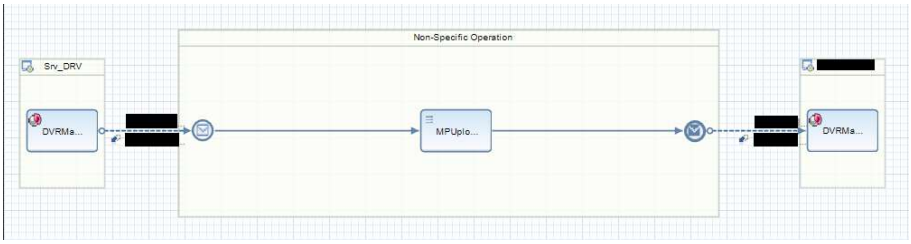
c. Receiver Channel



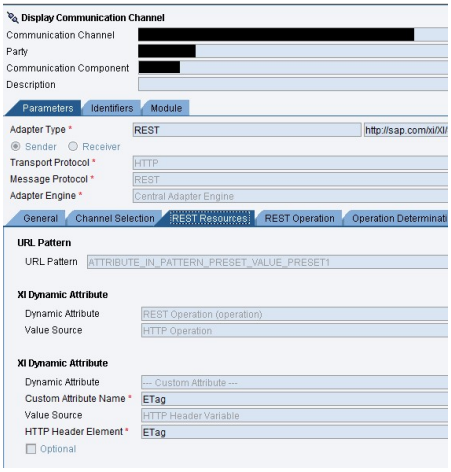


4. Response Receive Scenario – Upload Part

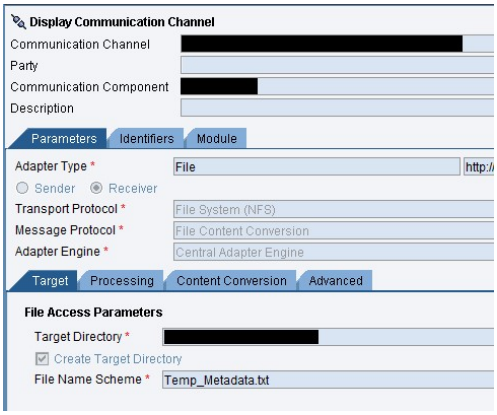
a. ICO

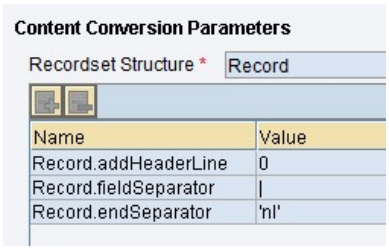


b. Sender Channel

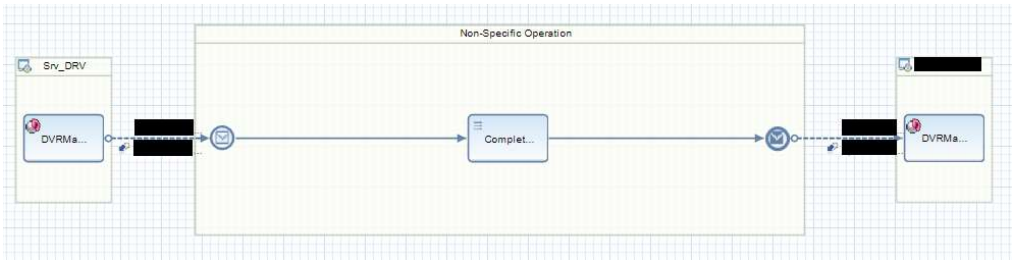


c. Receiver Channel

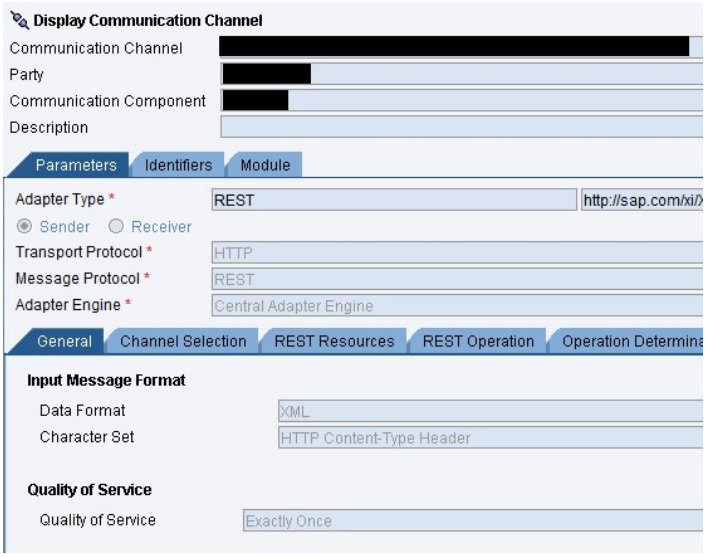




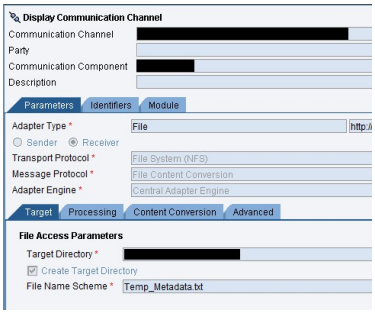
5. Response Receive Scenario – Complete Multipart Upload
   a. ICO

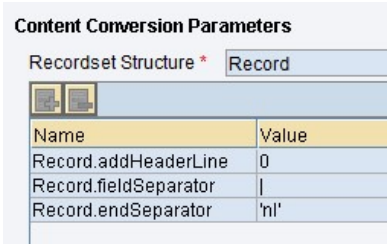


b. Sender Channel



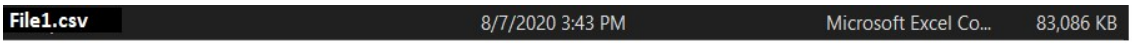
c. Receiver Channel



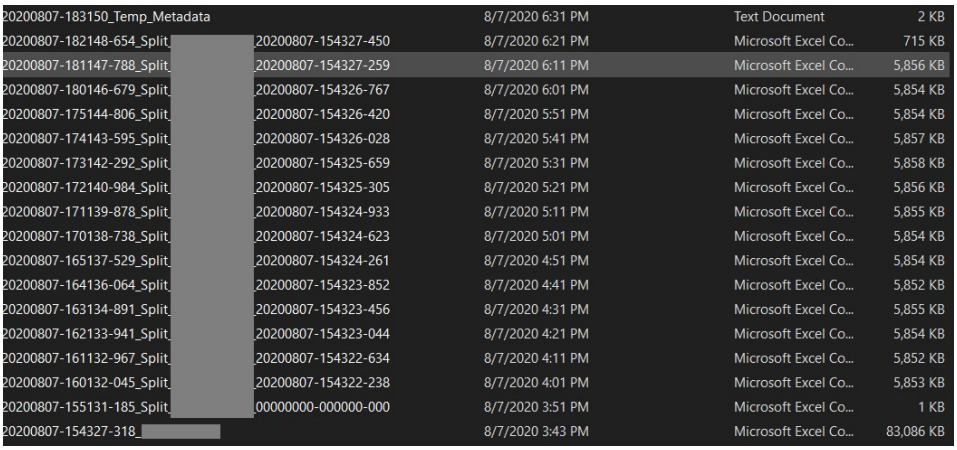


I have provided only the skeleton and the basic information to get this requirement. Because there is routing to multiple interfaces in the request part, I find it difficult to implement EOIO quality in this scenario. Hence I had to introduce a mandatory delay of 600 seconds. This was calculated based on the time it took to upload one part of the file, which was 6-7 minutes.

Also, after writing the Complete Multipart Upload, the metadata file is archived. This is a sample file to be uploaded.



A file of this size is split into multiple files of size slightly larger than 5 MB.



The metadata file contains the following data.

```
CREATEMULTIPARTUPLOAD|BUCKET:[redacted]|KEY:[redacted]|UPLOADID:PGZ0wZNmmPsRlyo7N1
UPLOADPART|ETAG:"e6b4f2e14139ce51f97aa71cc07ccec8"
UPLOADPART|ETAG:"230850302f0d58fc1f2bd98df3e92e71"
UPLOADPART|ETAG:"a5ff3351f1d076136f2b7a80bb7b573d"
UPLOADPART|ETAG:"1ff5aca1a0ea9a49c629891d0b1eb34f"
UPLOADPART|ETAG:"bfacbbe1c53a6248eca4a16cf03f6829"
UPLOADPART|ETAG:"104dc1017e693b0d2b738063f3d341db"
UPLOADPART|ETAG:"a0901397e97a98e13848cbb9da232c38"
UPLOADPART|ETAG:"84b7b275ec0fa8bae7002bbbd49363c2"
UPLOADPART|ETAG:"22d640fde392acf61045f7ae17827de7"
UPLOADPART|ETAG:"7d48879825fb8a9d6dafa3c3c5cc457a"
UPLOADPART|ETAG:"b5c721468e0ce5eda93b1d11eae783f5"
UPLOADPART|ETAG:"f8746f3abc8f26dd270f111a5427b8f0"
UPLOADPART|ETAG:"1e2273cc79f95d0ea20d29039fa505d2"
UPLOADPART|ETAG:"143c3647bbd12370054f8a62e6b3b40a"
UPLOADPART|ETAG:"1731513ecbadee9ae65caab8e31483fb"
COMPLETEMULTIPARTOUTPUT|[redacted]|"7d316b20ca2be814654fb462d494fb3b-15"
```

The above is written into this file by the 3 response mappings.

Conclusion:

This way we don't have to worry about files of any size to be uploaded into AWS s3. But there are a few points to be considered here:

1. SAP PO has no provision for greater than or lesser than in the condition editor. This prevented us from routing the file to a single chuck or multipart upload based on file size.
2. AWS has no central repo for the structures they use. It would've been easier for us that way to just download and use them. There was quite some time spent on resolving namespace issues.
3. Java mapping could be used to eliminate the need for separate mappings to resolve namespace issues. Also, EOIO quality for file sender to multiple receivers is possible with java mapping.
4. The performance of this scenario seems exponential to the number of parts it divides into. This could be optimized if we process files of different sizes based on threshold values. For example: files of size 5 MB or lesser can be uploaded in a single chunk. Files larger than that could be uploaded using multipart.
5. The absence of EOIO necessitates that only one file be present in the folder at a time. In case of multiple files, there will be multiple metadata files created and the process will find writing back the correct ETag values to the correct metadata file confusing.

Feel free to suggest any improvements/suggestions/feedback because this was done due to a time crunch. There can be many more ways to optimize this.

Lastly, I've listed the links/people I referred to during the development of this requirement.

3. https://blogs.sap.com/2019/07/18/idoc-rest-async-sync-scenario-approach-with-aleaud/
4. https://blogs.sap.com/2019/05/31/integrating-amazon-simple-storage-service-amazon-s3-and-sap-ecc-v6.0-via-sap-pi-v7.5-using-aws-signature-v5-and-signing-algorithm-hmac-sha256/
5. https://docs.aws.amazon.com/AmazonS3/latest/dev/mpuoverview.html
6. https://github.com/pnathan01/AWSSAPMultipartUploadLibrary
7. https://wiki.scn.sap.com/wiki/display/XI/Where+to+get+the+libraries+for+XI+development
8. https://launchpad.support.sap.com/#/notes/0001763011
9. Robbie Cooray from AWS – He was my go-to person for valuable inputs and great ideas. Thanks Robbie! 😊