Seaboard – Concepts for creating a message for MEC manually

# Purpose

Normally most if not all of this document won’t be used. Typically messages come into MEC already having a xml schema definition (XSD) and they will have an XML formatted with an envelope that contains a header and body section. This document attempts to walk through the process of creating those things manually. For example sake, this document makes use of the GL Trial balance message that Seaboard initially used for a manual test of MEC. Even though the examples use GL Trial balance as an example, the principles of this document can be applied to similar situations where the sender of a message hasn’t provided an xsd or their xml document doesn’t contain a typical xml format (no envelop, header – body format).

First it is important to understand the higher level view of why we are starting at this point with the XML. Typical flow of a message is to be received by MEC through a defined channel, then identified with a partner agreement, then use that agreement to determine the programmed processes that need to be executed on the message based on which agreement it has associated to the message. So, a new user may want to start with the channel configuration. While this is not really a wrong approach it isn’t the best approach. The last thing that happens in the flow is the executing of the programmed processes. Regardless of a message being Inbound (into M3) or outbound (out of M3) there is typically a data translation process. This process is key to the whole flow and this is the point which is the best place to start with a MEC message. The reason for this is that at the point when the partner admin is identifying the agreement, the mapping process will need to point to a mapping. If there is no mapping created then it will be impossible to do this without going back to the mapping manager and creating the mapping. Therefore it is best to start by creating the mapping first so that all the elements will be ready for creating the agreement in the partner admin. In our document we will work with xml from a system which provides only the data in xml format.

## XML structure

The normal xml structure is to have a wrapper around the entire xml message. This wrapper is called an envelope. Theoretically 1 xml file could contain multiple envelopes which could contain multiple messages. The Envelope would then be a definition that indicates the start and end of each message. Even though this is theoretically possible, it is not supported by MEC and many EDI systems. Therefore we will use the envelope structure but it will only wrap around 1 message within the xml.

Within the envelope tags are typically 2 other structures commonly called ‘The Header’ and ‘The detail’ or ‘The body’. The header section typically contains information regarding which system the message was originally sent by (source system), the system the message is meant for (destination system), and what the message topic is (outbound PO, inbound GL Trial balance, etc). Also in this section there can be other higher level pieces of data like System which has transformed or delivered the message (EDI broker, middle wear systems, etc), Suppliers code/name, customers code/name. Although these extra pieces of data are good to have they are generally not necessary. In the detail or body structure is where the message keeps all the data regarding the message. This includes the header and detail portions of any data. Therefore messages regarding orders which have header and detail data would have all this data in the Detail section of the xml message. Below is an conceptual example

<Envelope>

<Header>

<source system>

Source system name

</source system>

<destination system>

Destination system name

</destination system>

<message>

GL Trial balance

</message>

<translation system>

Informatica

</translation system>

</Header>

</detail>

<Document header>

<partner name>

Some partner

</parner name>

<order id>

OrderID

</order id>

</Document header>

</Document detail>

<OrderLine>

Orderlinenumber

</OrderLine>

<OrderLineDescription>

Orderlinedescription

</OrderLineDescription>

</detail>

</Envelope>

In some cases the source system will only generate the detail information. In those cases it will be necessary to understand the above structure in order to design a proper xml message. The purpose of designing a proper xml message as demonstrated above is so we can use it for creating an XML Schema diagram (XSD). The schema diagram is required for MEC’s mapping tool. Therefore we need to go through this exercise in order to be able to use MEC. An XSD is an XML file that describes the structure of an XML message. Since this document is a description of the message it can be used by programs to generate further messages in the exact same structure. Generally, an XSD is an important document.

## Creating an XML in a proper structure.

As mentioned above, the xml coming from a source system might not be in a normal xml message structure. A lot of times source systems just dump the data into a generic data xml format which only contains body information. When this happens we must create a proper xml structure and then copy the data into our body. Once we have the message created we can use some tools to auto create the Schema (xsd).

There are 2 normal ways to create a structured xml. The first way is manually. Based on knowledge and experience the xml envelope and header would be typed in manually. Obviously this is not the typical way because it requires fairly deep intimate knowledge of creating xml messages. The second and more typical way is to copy an existing xml message that already contains the proper structure, then adjust the information and a few of the tags that will make our new message structure. This is how we accomplished the xml message for GL Trial balance.

First we were provided an outbound Purchase order message XML from Frank. We took that message and we removed everything in the detail section. Then we looked at the header and adjusted all the tags to say “GLTrialBalance” instead of “PurchaseOrder”. We also adjusted the data that was being sent for sender since the PO message was outbound, the sender said M3. In these cases we changed all that data to say “AS400”. Below is the example of the original PO message and the converted GLTrialBalance message that was created from the PO message

### Original Purchase Order

<?xml version="1.0" encoding="UTF-8" ?>

<SyncPurchaseOrder xmlns="http://schema.infor.com/InforOAGIS/2" xmlns:udt="http://www.openapplications.org/oagis/9/unqualifieddatatypes/1.1" xmlns:oacl="http://www.openapplications.org/oagis/9/codelists" xmlns:ccts="urn:un:unece:uncefact:documentation:1.1" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:clmIANAMIMEMediaTypes="http://schema.infor.com/InforOAGIS/2/IANAMIMEMediaTypes:2003" xmlns:clm5639="http://schema.infor.com/InforOAGIS/2/languagecode/5639:1988" xmlns:clm54217="http://schema.infor.com/InforOAGIS/2/currencycode/54217:2001" xmlns:clm66411="http://schema.infor.com/InforOAGIS/2/unitcode/66411:2001" xmlns:qdt="http://www.openapplications.org/oagis/9/qualifieddatatypes/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://schema.infor.com/InforOAGIS/2 http://schema.infor.com/2.7.0/InforOAGIS/BODs/Developer/SyncCustomerPartyMaster.xsd" releaseID="9.2" versionID="2.7.0" systemEnvironmentCode="Production" languageCode="GB">

<ApplicationArea>

<Sender>

<LogicalID schemeVersionID="15.1.0.0">lid://infor.m3be.edu\_001</LogicalID>

<ComponentID schemeVersionID="15.1.2">M3BE</ComponentID>

</Sender>

<CreationDateTime>2014-02-17T16:56:42.016Z</CreationDateTime>

<BODID>a0188543-8bde-41e5-9a16-8caee9859aee</BODID>

</ApplicationArea>

<DataArea>

<Sync>

<TenantID>001</TenantID>

<LocationID>001</LocationID>

<ActionCriteria>

<ActionExpression actionCode="Add" />

</ActionCriteria>

</Sync>

<PurchaseOrder>

<PurchaseOrderHeader>

<PruchaseOrderHeaderData>

Data

</PurchaseOrderHeaderData>

</PurchaseOrderHeader>

<PurchaseOrderLine>

data

</PurchaseOrderLine>

</PurchaseOrder>

</DataArea>

</SyncPurchaseOrder>

### GL Trial Balance

<?xml version="1.0" encoding="UTF-8" ?>

<SyncGLTrialBalance releaseID="9.2" versionID="2.7.0" systemEnvironmentCode="Production" languageCode="GB">

<sqle36638 xmlns:dt="urn:schemas-microsoft-com:datatypes">

<ApplicationArea>

<Sender>

<LogicalID schemeVersionID="15.1.0.0">lid://Seaboard.AS400.Prod</LogicalID>

<ComponentID schemeVersionID="15.1.2">AS400</ComponentID>

</Sender>

<CreationDateTime>2015-04-29T12:25:42.016Z</CreationDateTime>

<BODID>a0188543-8bde-41e5-9a16-8caee9859aee</BODID>

</ApplicationArea>

<DataArea>

<Sync>

<TenantID>001</TenantID>

<LocationID>001</LocationID>

<ActionCriteria>

<ActionExpression actionCode="Add" />

</ActionCriteria>

</Sync>

<GLTrialBalance>

<GLTrialBalanceHeader>

GLTrialBalanceHeaderData

</GLTrialBalanceHeader>

<GLTrialBalanceDetail>

GLTrialBalanceDetailData

</GLTrialBalanceDetail>

</GLTrialBalance>

</DataArea>

</sqle36638>

</SyncGLTrialBalance>

## Inserting the Data into the new structure

Once this structure was created we saved this file. Now once the source system (AS400) generates the header and detail information, we would open this file and insert that data into the body section (<DataArea></DataArea>)

For example the with the GLTrialBalance, the AS400 generated 2 files. One with header data and one with detail data.

### Header (plb9cpp.xml)

<?xml version="1.0" encoding="ISO-8859-1"?>

<sqle36638 xmlns:dt="urn:schemas-microsoft-com:datatypes">

<viewfmt>

<b9wja1 dt:dt="number">1</b9wja1>

<b9qxcd>360</b9qxcd>

<b9ait1>201502</b9ait1>

<b9ajt1>TB\_UPLD</b9ajt1>

<b9wga1 dt:dt="number">0</b9wga1>

<b9akt1>JDE Balance Transfer</b9akt1>

<b9wia1 dt:dt="number">0</b9wia1>

<b9wha1 dt:dt="number">0</b9wha1>

<b9addx dt:dt="number">0</b9addx>

<b9aedx dt:dt="number">20150429</b9aedx>

<b9aavn>ISBGUTI</b9aavn>

<b9aadt dt:dt="number">0</b9aadt>

<b9amtm dt:dt="number">0</b9amtm>

<b9ajst></b9ajst>

<b9ahvn></b9ahvn>

<b9agvn>M3JDE200</b9agvn>

<b9abvn></b9abvn>

<b9abdt dt:dt="number">1150429</b9abdt>

<b9abtm dt:dt="number">81359</b9abtm>

</viewfmt>

</sqle36638>

### Detail (plcacpp.xml)

<?xml version="1.0" encoding="ISO-8859-1"?>

<sqle36695 xmlns:dt="urn:schemas-microsoft-com:datatypes">

<viewfmt>

<cawja1 dt:dt="number">1</cawja1>

<caqxcd>360</caqxcd>

<caait1>201502</caait1>

<cawka1 dt:dt="number">1</cawka1>

<cawga1 dt:dt="number">0</cawga1>

<caplt1>1410,,,,,,,0,484996.81,JDE Balance Transfer</caplt1>

<cawha1 dt:dt="number">0</cawha1>

<cawia1 dt:dt="number">0</cawia1>

<caaddx dt:dt="number">0</caaddx>

<caaedx dt:dt="number">20150429</caaedx>

<caaavn>ISBGUTI</caaavn>

<caaadt dt:dt="number">0</caaadt>

<caamtm dt:dt="number">0</caamtm>

<caajst></caajst>

<caahvn></caahvn>

<caagvn>M3JDE200</caagvn>

<caabvn></caabvn>

<caabdt dt:dt="number">1150429</caabdt>

<caabtm dt:dt="number">81359</caabtm>

</viewfmt>

<viewfmt>

<cawja1 dt:dt="number">1</cawja1>

<caqxcd>360</caqxcd>

<caait1>201502</caait1>

<cawka1 dt:dt="number">2</cawka1>

<cawga1 dt:dt="number">0</cawga1>

<caplt1>1507,360,45716,,,,,0,107022.21-,JDE Balance Transfer</caplt1>

<cawha1 dt:dt="number">0</cawha1>

<cawia1 dt:dt="number">0</cawia1>

<caaddx dt:dt="number">0</caaddx>

<caaedx dt:dt="number">20150429</caaedx>

<caaavn>ISBGUTI</caaavn>

<caaadt dt:dt="number">0</caaadt>

<caamtm dt:dt="number">0</caamtm>

<caajst></caajst>

<caahvn></caahvn>

<caagvn>M3JDE200</caagvn>

<caabvn></caabvn>

<caabdt dt:dt="number">1150429</caabdt>

<caabtm dt:dt="number">81359</caabtm>

</viewfmt>

<viewfmt>

etc

</viewfmt>

With these 2 files we insert them into our structure to generate a new message.

### New Message example

<?xml version="1.0" encoding="UTF-8" ?>

<SyncGLTrialBalance releaseID="9.2" versionID="2.7.0" systemEnvironmentCode="Production" languageCode="GB">

<sqle36638 xmlns:dt="urn:schemas-microsoft-com:datatypes">

<ApplicationArea>

<Sender>

<LogicalID schemeVersionID="15.1.0.0">lid://Seaboard.AS400.Prod</LogicalID>

<ComponentID schemeVersionID="15.1.2">AS400</ComponentID>

</Sender>

<CreationDateTime>2015-04-29T12:25:42.016Z</CreationDateTime>

<BODID>a0188543-8bde-41e5-9a16-8caee9859aee</BODID>

</ApplicationArea>

<DataArea>

<Sync>

<TenantID>001</TenantID>

<LocationID>001</LocationID>

<ActionCriteria>

<ActionExpression actionCode="Add" />

</ActionCriteria>

</Sync>

<GLTrialBalance>

<GLTrialBalanceHeader>

<viewfmt>

<b9wja1 dt:dt="number">1</b9wja1>

<b9qxcd>360</b9qxcd>

<b9ait1>201410</b9ait1>

<b9ajt1>TB\_UPLD</b9ajt1>

<b9wga1 dt:dt="number">0</b9wga1>

<b9akt1>JDE Balance Transfer</b9akt1>

<b9wia1 dt:dt="number">0</b9wia1>

<b9wha1 dt:dt="number">0</b9wha1>

<b9addx dt:dt="number">0</b9addx>

<b9aedx dt:dt="number">20150430</b9aedx>

<b9aavn>ISBGUTI</b9aavn>

<b9aadt dt:dt="number">0</b9aadt>

<b9amtm dt:dt="number">0</b9amtm>

<b9ajst></b9ajst>

<b9ahvn></b9ahvn>

<b9agvn>M3JDE200</b9agvn>

<b9abvn></b9abvn>

<b9abdt dt:dt="number">1150430</b9abdt>

<b9abtm dt:dt="number">83224</b9abtm>

</viewfmt>

</GLTrialBalanceHeader>

<GLTrialBalanceDetail>

<viewfmt>

<cawja1 dt:dt="number">1</cawja1>

<caqxcd>360</caqxcd>

<caait1>201410</caait1>

<cawka1 dt:dt="number">1</cawka1>

<cawga1 dt:dt="number">0</cawga1>

<caplt1>1410,,,,,,,0,4182029.63,JDE Balance Transfer</caplt1>

<cawha1 dt:dt="number">0</cawha1>

<cawia1 dt:dt="number">0</cawia1>

<caaddx dt:dt="number">0</caaddx>

<caaedx dt:dt="number">20150430</caaedx>

<caaavn>ISBGUTI</caaavn>

<caaadt dt:dt="number">0</caaadt>

<caamtm dt:dt="number">0</caamtm>

<caajst></caajst>

<caahvn></caahvn>

<caagvn>M3JDE200</caagvn>

<caabvn></caabvn>

<caabdt dt:dt="number">1150430</caabdt>

<caabtm dt:dt="number">83224</caabtm>

</viewfmt>

etc

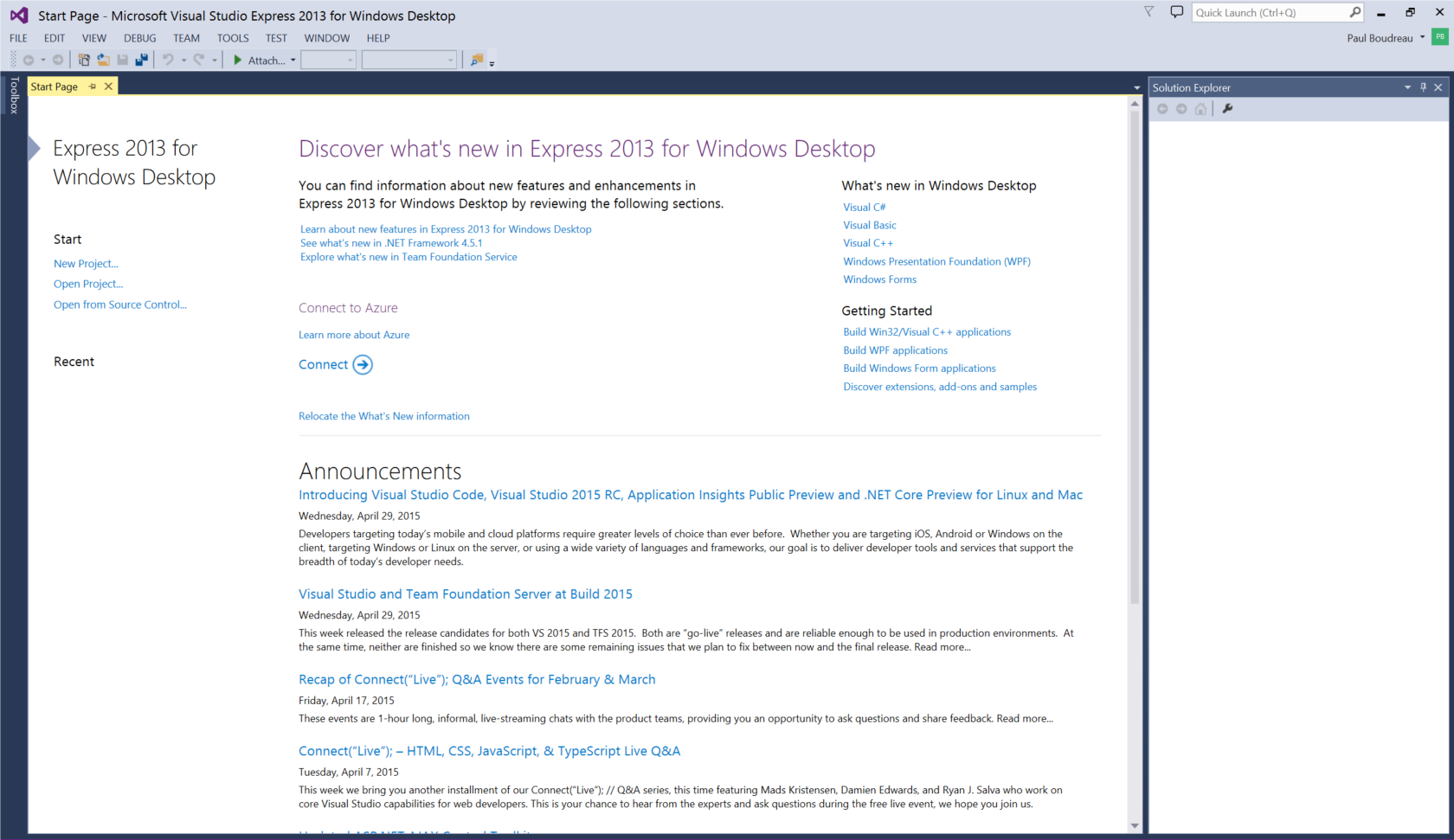
</viewfmt>

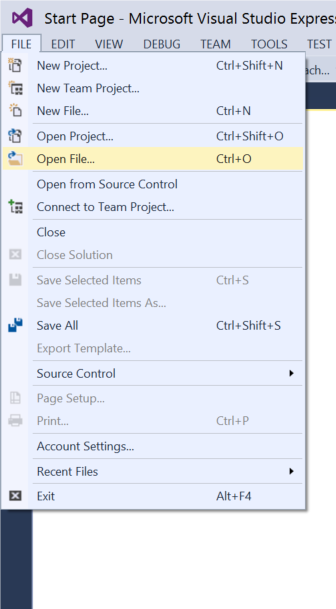
## Creating a Schema (XSD)

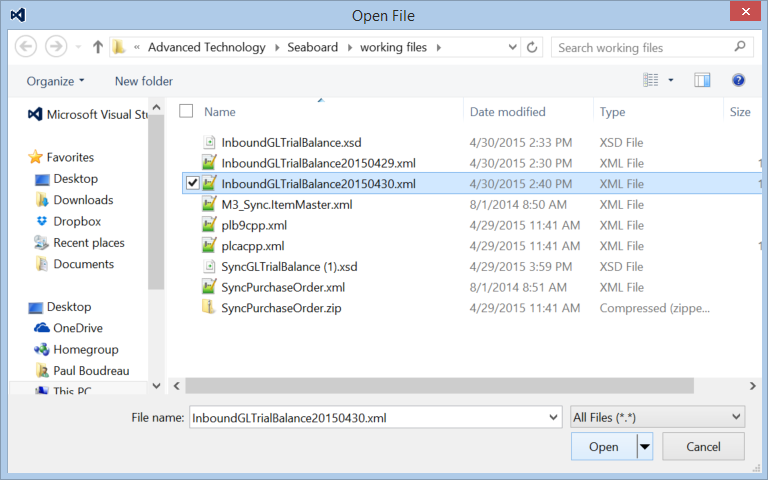
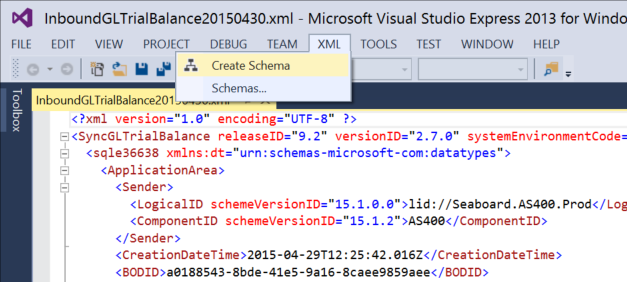
Now that we have a completed xml message in the proper format we can create a schema for it using it as a guide.

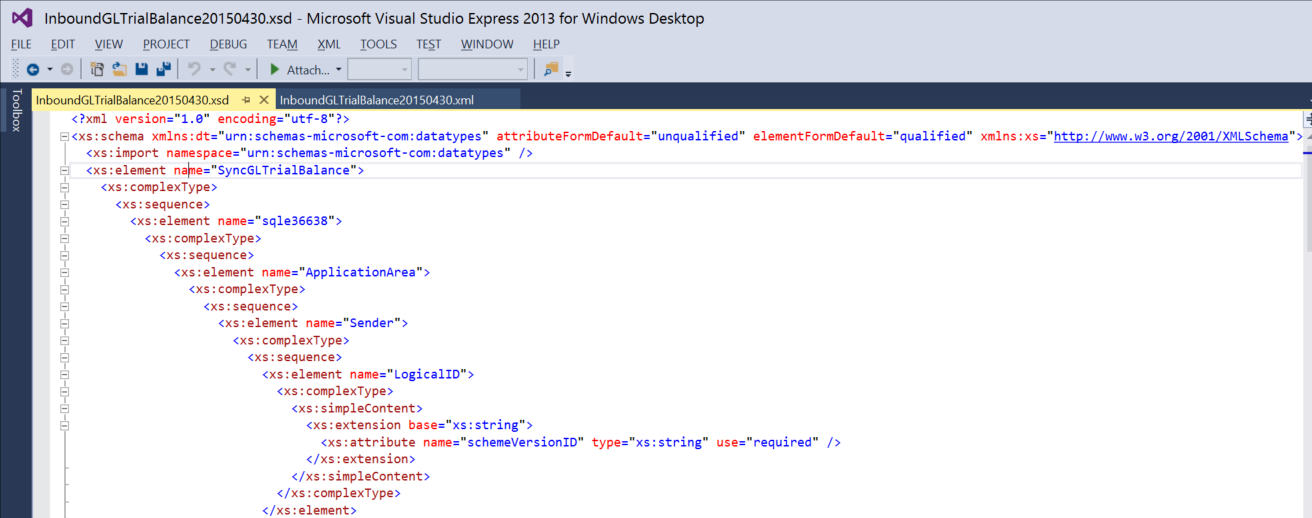
There are several free online tools that you can pass an XML document to and they will output a schema based on that xml. However, the Microsoft development tool ‘Visual Studio’ can also do the same thing and the express version can be downloaded from Microsoft for free.

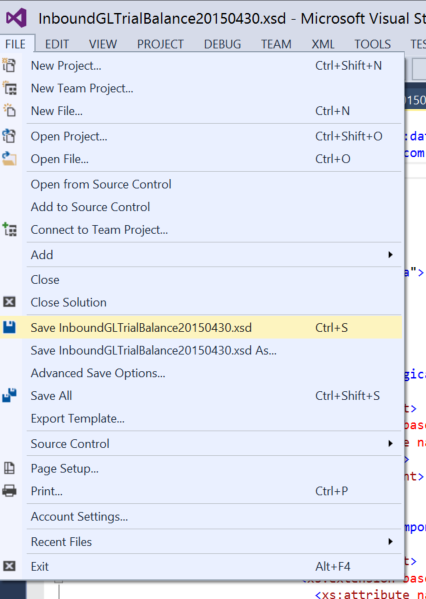
This document will assume we are using Visual Studio Express for this example.

1. Download and install Visual Studio Express if you don’t have it. Or if Seaboard allows, get a full copy of Visual Studio installed.
2. Open Visual Studio. 
3. Choose Open File from the File menu



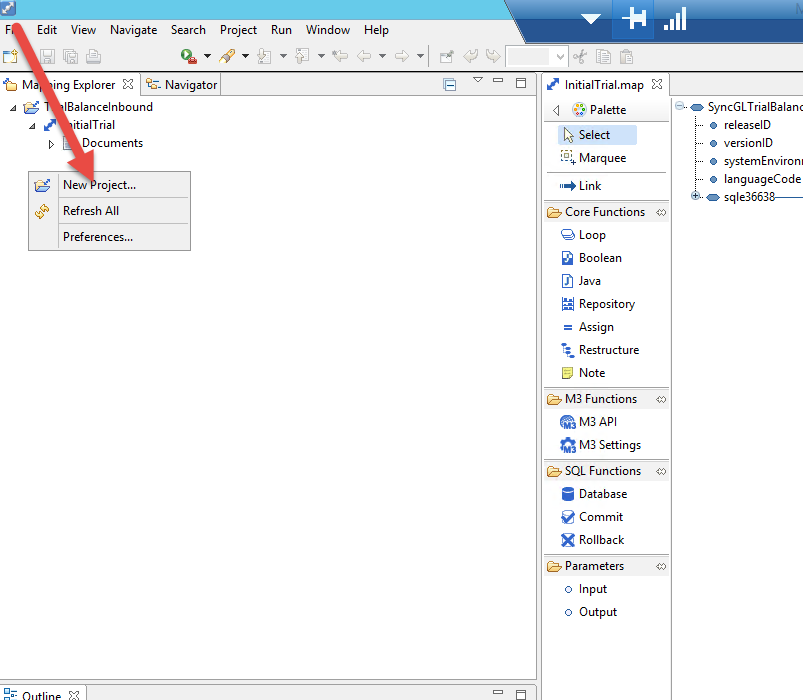
1. Point at the xml document to open
2. Choose the Generate XSD from the XML menu
3. A new tab should appear that looks like the following

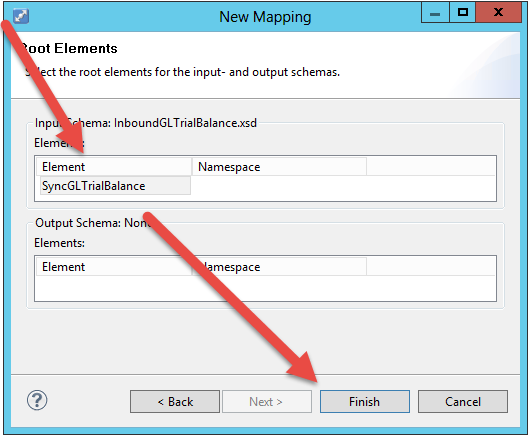
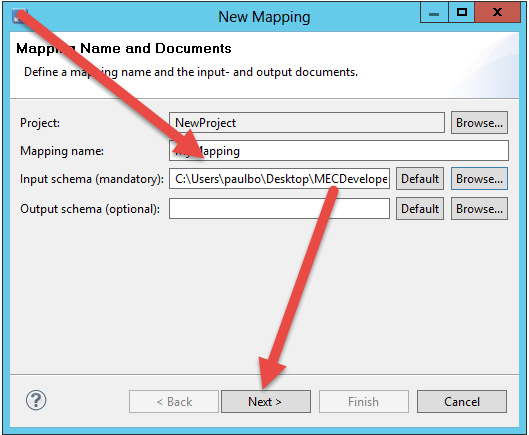
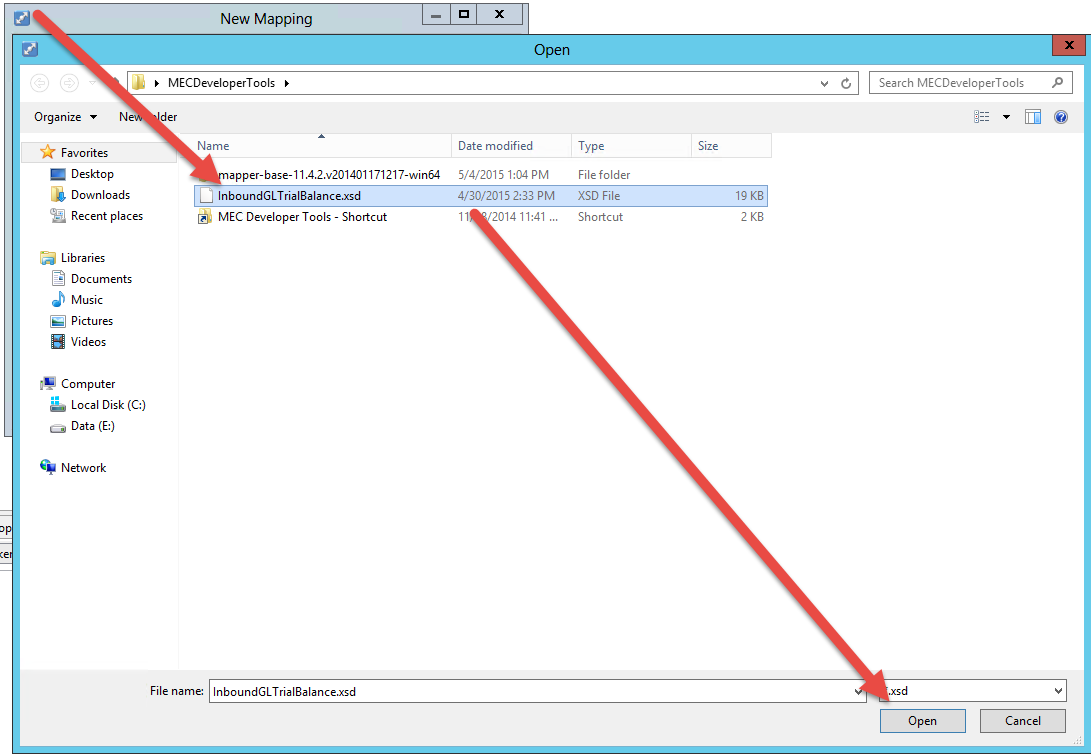
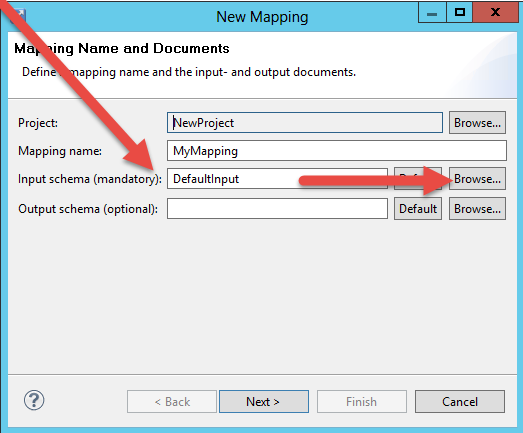
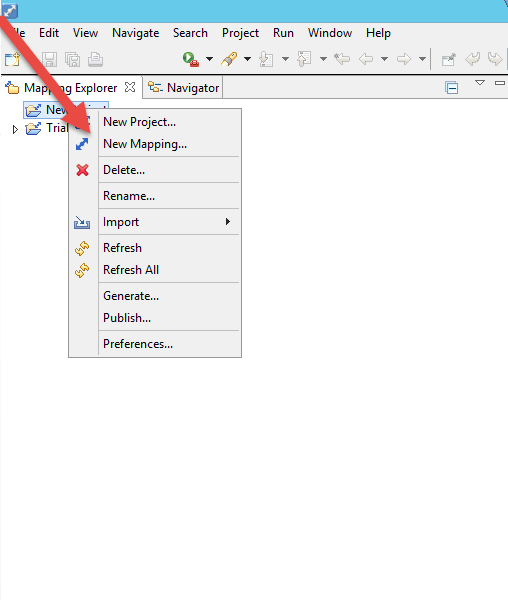
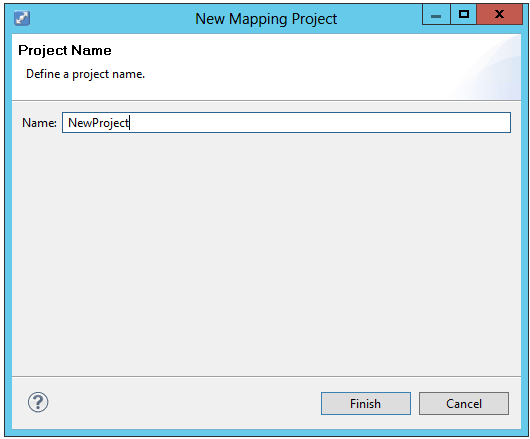


1. Save the Schema (or Save as.. if you want to rename it) 

## MEC Mapping Manager

Now that we have the Schema and the xml, we can begin the mapping within MEC Mapping Manager. As described in the training manual, open a new project and name it. Then open a new mapping. During this step the mapping manager will require an inbound schema. This is where you will point the mapping manager at the new schema that was just created by visual studio.





Once this is finished the mapping manager will now show the input schema as per the training manual on the left side.

Remember that the right side is only needed if the message is outbound out of M3 to some other destination system. In that case, the same process would need to be repeated above to create an example xml and xsd. Generally speaking though, the destination system will likely have an example xml and/or xsd they expect the data to come to them as. At the very least they will have a requirements document that will explain their data layout and that will leave the MEC owner to mock up a sample xml file from the requirements document manually

Since our example is an inbound message the API’s inside the map will write the data directly into the M3 Business Engine (M3BE). Therefore there is no output xml needed for inbound to M3

## Creating the Mapping in MEC

Now that you have this piece done the next is to design the map itself. In the map there should always be an initial note to document the initial creation and any changes to the map. This note should exist in EVERY map. Time and again maps will need to be adjusted and without this vital information the work can take up to twice as long without knowing who did what and when. Never skip this step

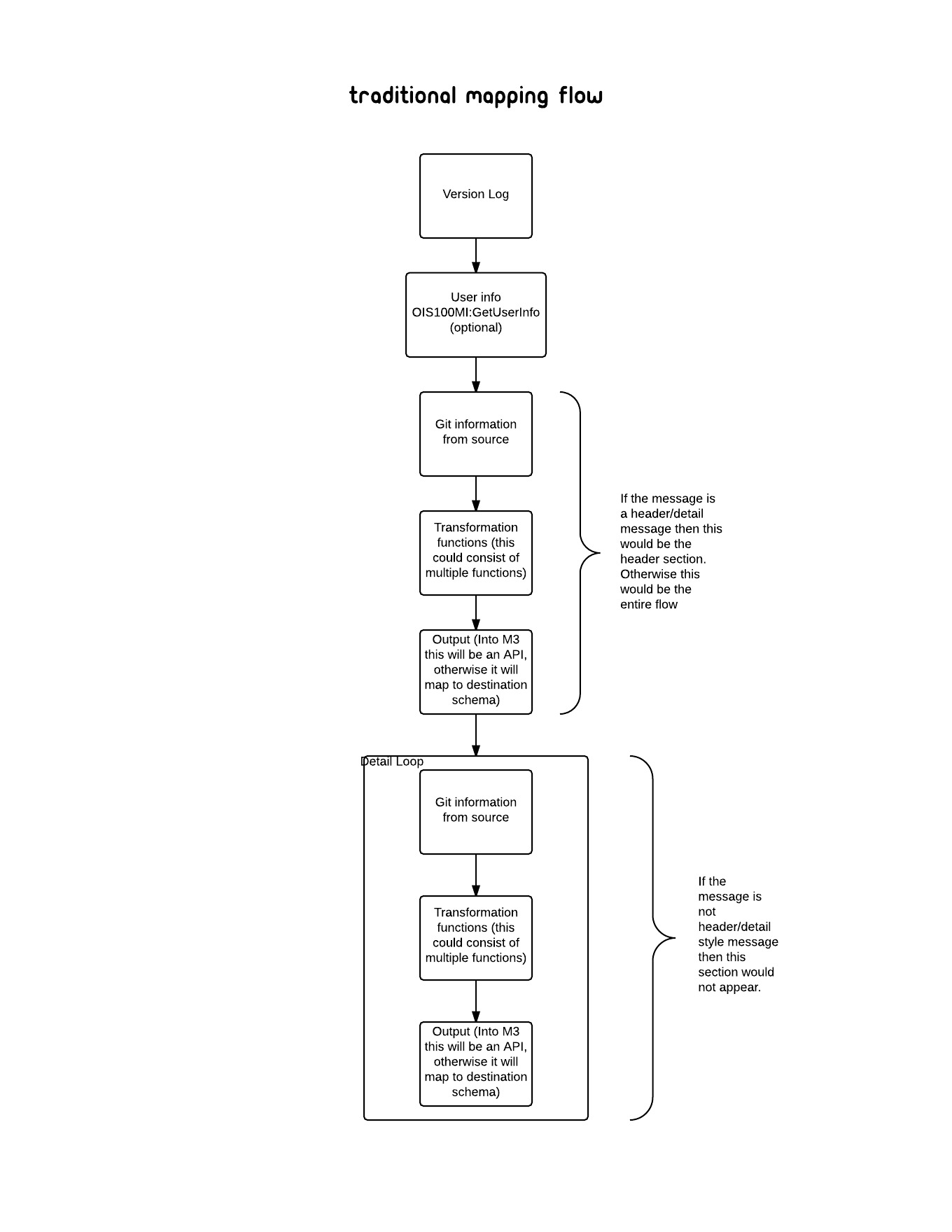
Also a very good practice is to rename all the objects you use in your mapping to something more English readable descriptive. By default the mapper will create functions <function\_1>, <function\_2>, etc. When trying to review this code it can become very confusing to remember if you were trying to track <function\_10> or was it <function\_11>? Instead, if the functions are named with something more descriptive it becomes much easier to review the code. (GiterGLTrialBalHeader vs GLTrialBalanceDetailLoop are easy to tell apart and know what each does)

Next the design of the mapping flow is created. Usually there is a function created to bring in the input data (from the Schema) and store the data values into “English Readable Descriptive” variables. For these input functions I normally prefix the function name with “Giter”, meaning that they “git” data from the input message. If the company number comes into the system as <b9wja1> then the value should come into an input parameter variable named something like (iCompNum) This notation says “i” is an input variable, CompNum describes what is data is being stored.

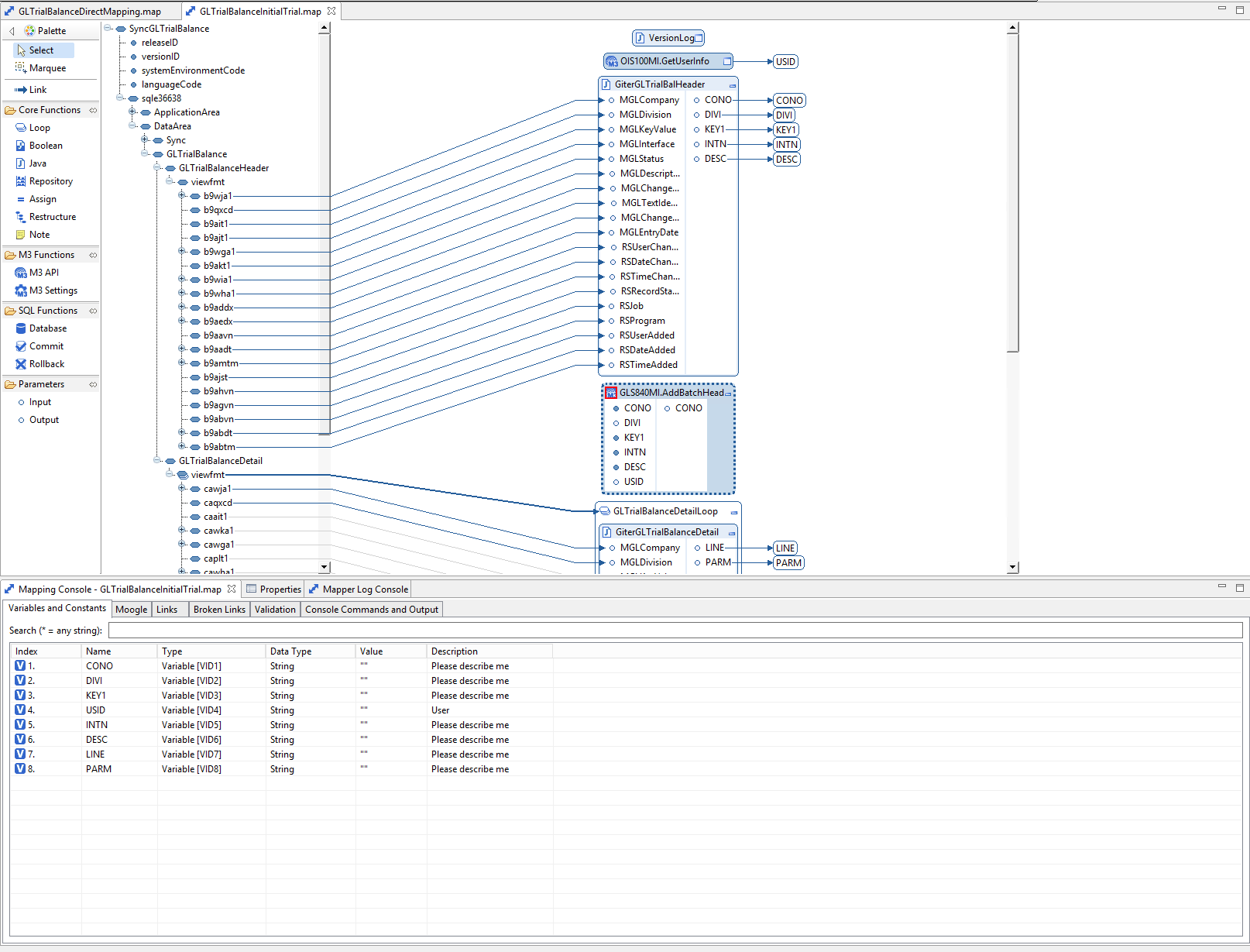
After the variables are brought into the code from the input message, the next function is normally a translation function. Within M3 Smart office there are panels that let M3 cross reference data from MEC input into M3 acceptable data codes. These translations are setup in CRS881 and CRS882. Inside the java function in the mapping the DataTranslator.toMessage method will be used to access the translation data that is setup in CRS881 and CRS882. For Seaboards specific use of MEC, there will be no need for this type of function in the mapping since MEC won’t be the application that does translation. For the most part, MEC will be passing out un-translated data and passed in data that is already translated via Informatica.

The next general function in the flow is usually an API call (assuming an inbound message). This function uses an API to send in the data directly to the M3 Business Engine so it can write the data into the M3 database.

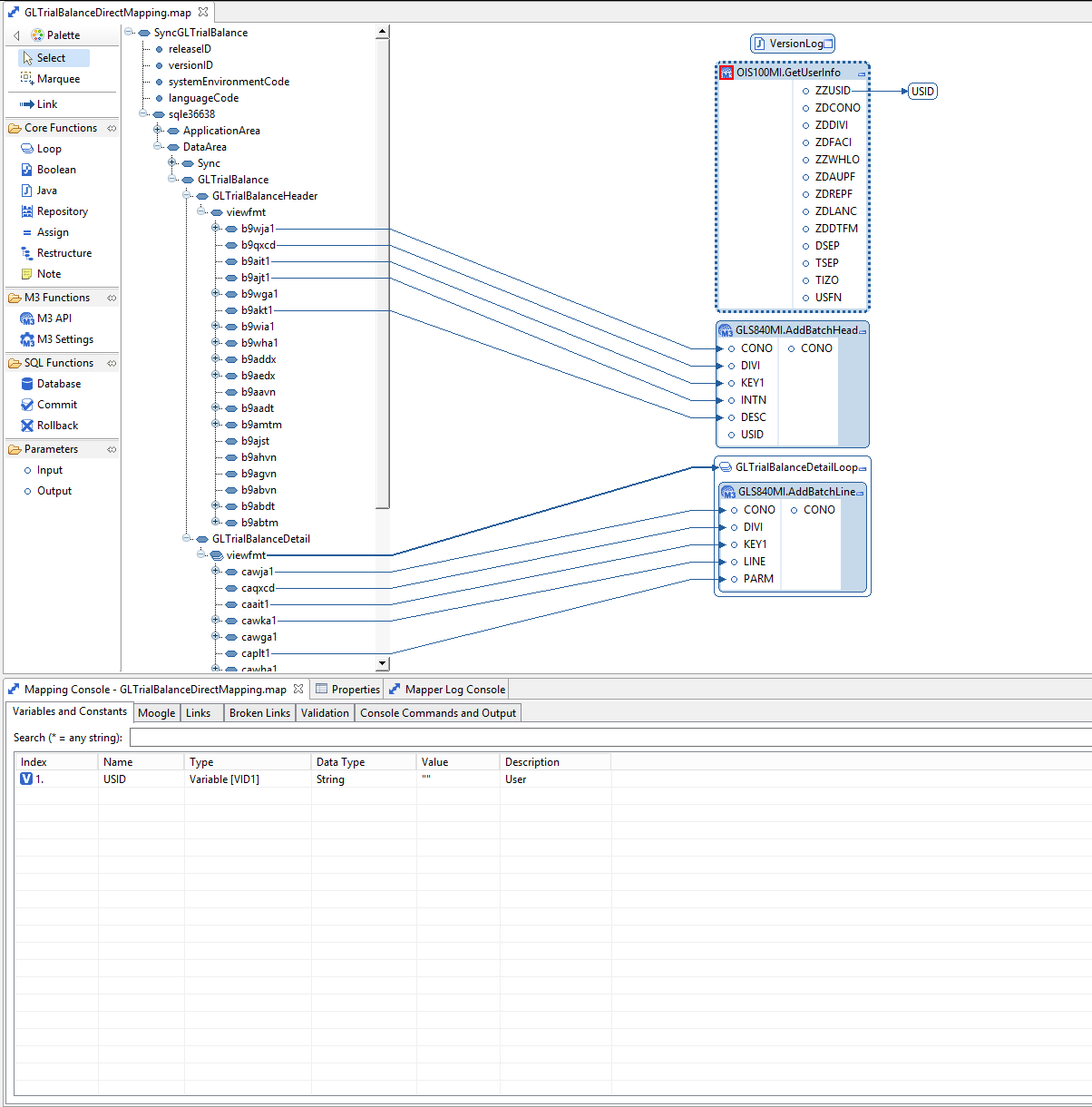
This structure is usually duplicated a second time for messages that have header & details. The first structure is used to record the header and the second is used



No transformation process in this example for GL Trial Balance



The mapping can be designed much simpler since there is no transformation needed. Input schema variables can be fed directly into the API input values



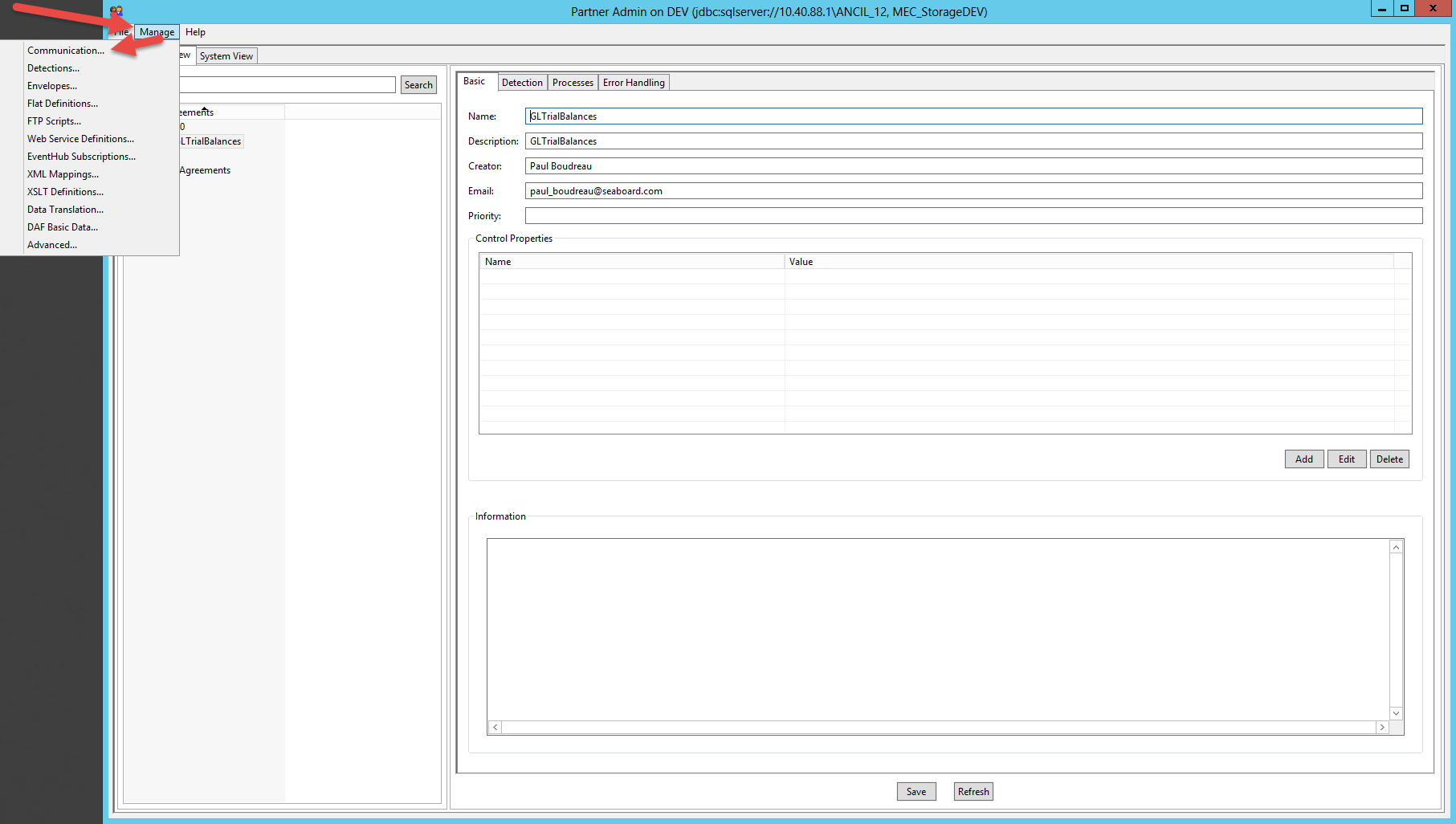
## Partner Admin Tool setup

Now that we have the sample xml message, the xml schema (XSD) and the map complete, we are ready to go through all the setup required in the PAT.

### Setup Communication

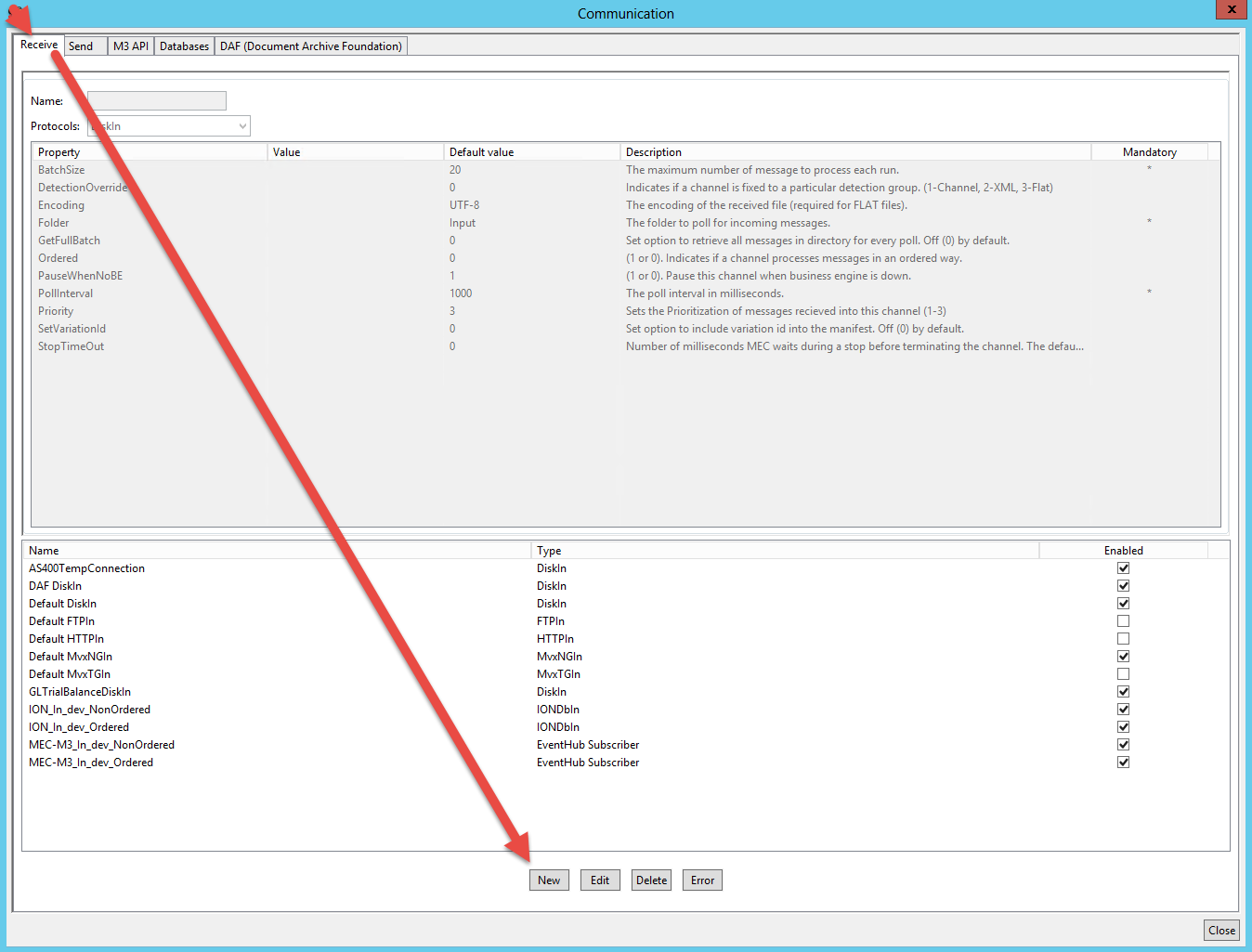
This is not always necessary. For example, if there is a generic “In Box” folder which all the messages will be dumped, then a new agreement would use the previously setup “In Box” channel. Therefore in that situation a new receiving channel would not need to be setup. In our document we assume that a new channel is needed in order to document the process of setting up a new channel.

From PAT, click the Manage drop down menu and choose the Communication option



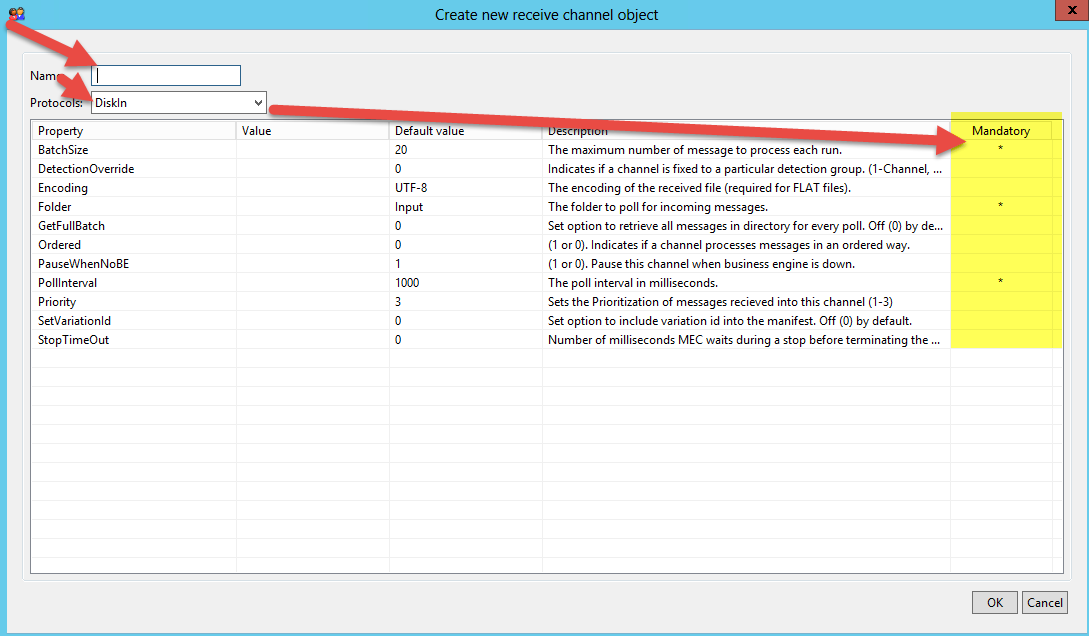
Since our example is an inbound message we will demonstrate the setup of a receiving channel. However, the setup of a send channel is a similar process.

Once the communications panel is open, choose the receive tab. At the bottom of the panel click the New button

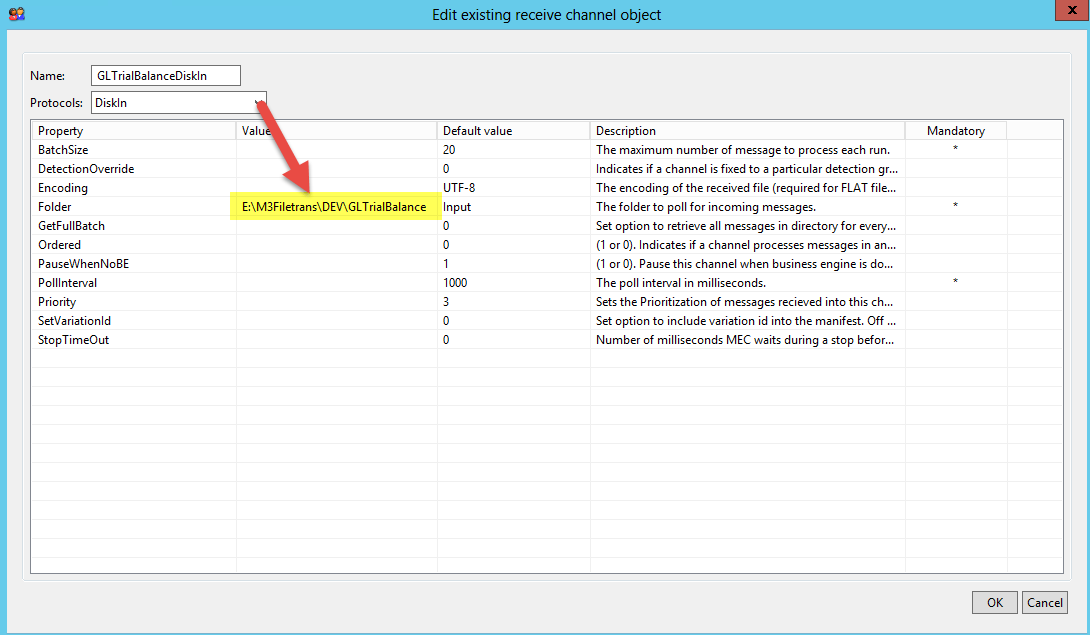


Type in a name. This will be a “Tag” identifier for the channel you are about to setup. There is no validation on the name so you can call the channel anything you like. In this example we called our channel GLTrialBalanceDiskIn. After the name you can choose a protocol for this receive channel. Depending on the protocol you choose the parameters needed to define the channel will change. In our example we will use a simple DiskIn Protocol. For explanation on other protocols please read the PAT user guide.

Once the protocol is selected, the parameters need to be entered. Normally there are defaults for the parameters. The key parameters to watch closely are the ones that are tagged as Mandatory in the right most column. These must be set correctly for the channel to work. The other parameters may be adjusted from their defaults but typically this is only done to tweek the channel when issues arise.

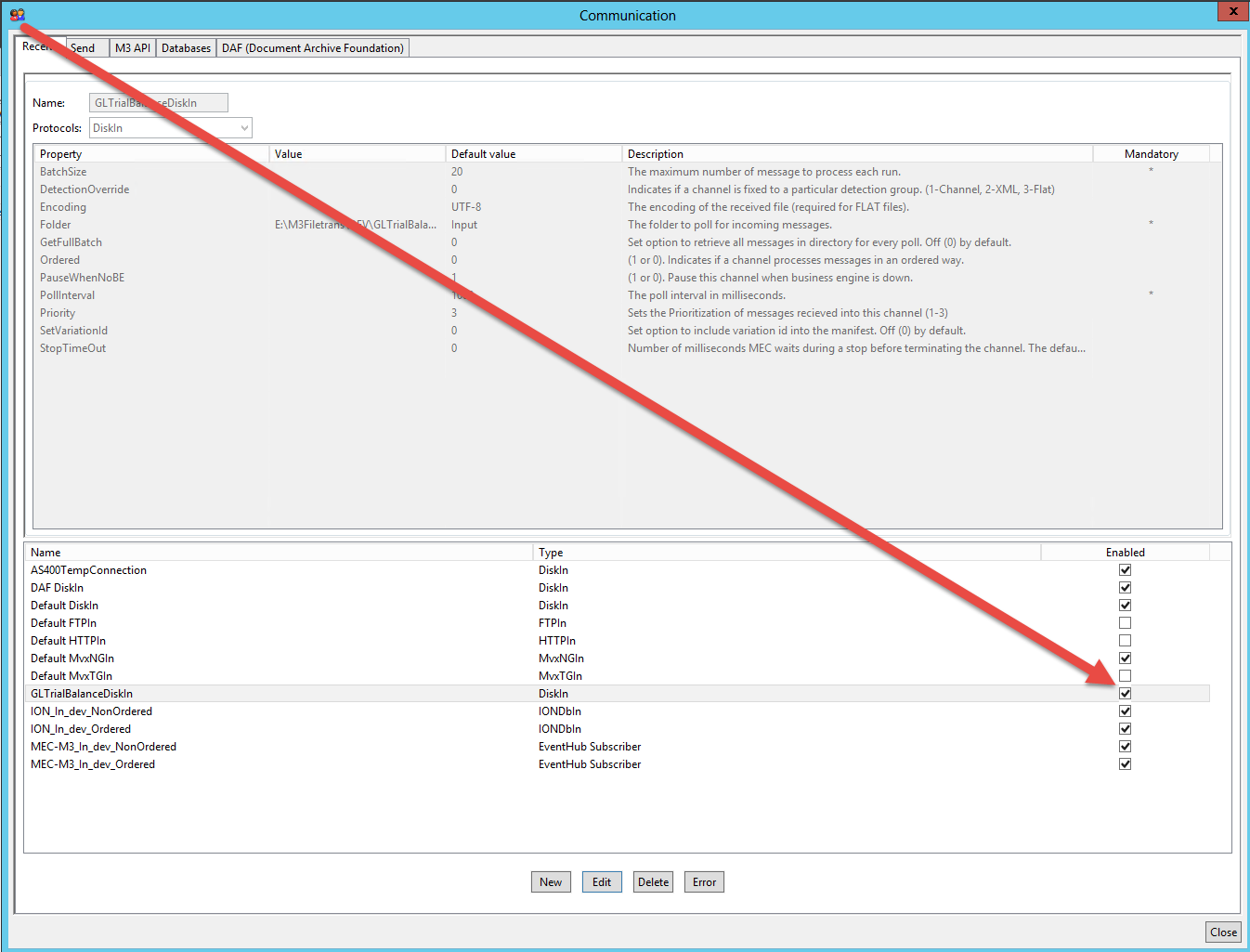


With our example the only value that needs to change from the default is the folder property



Once the change is done, click the ok button at the bottom of the panel

Even though the new channel is defined, it is not working on the server until 2 more steps are completed. First on the communications panel the channel has to be checked as enabled and the MEC service in LCM needs to be stopped and re-started.



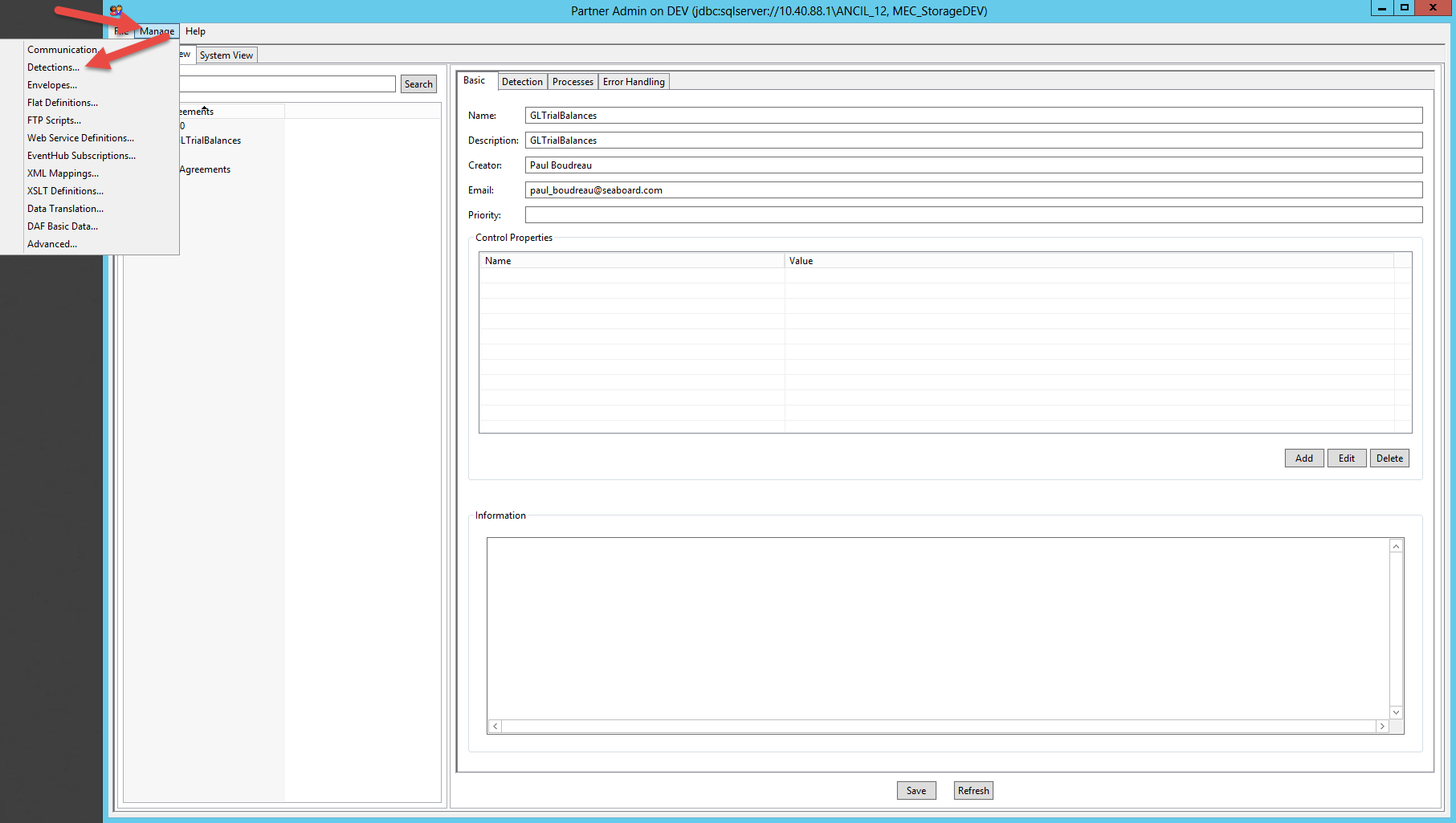
Once this is completed, click close on the communications panel

### Setting up the detection parameters

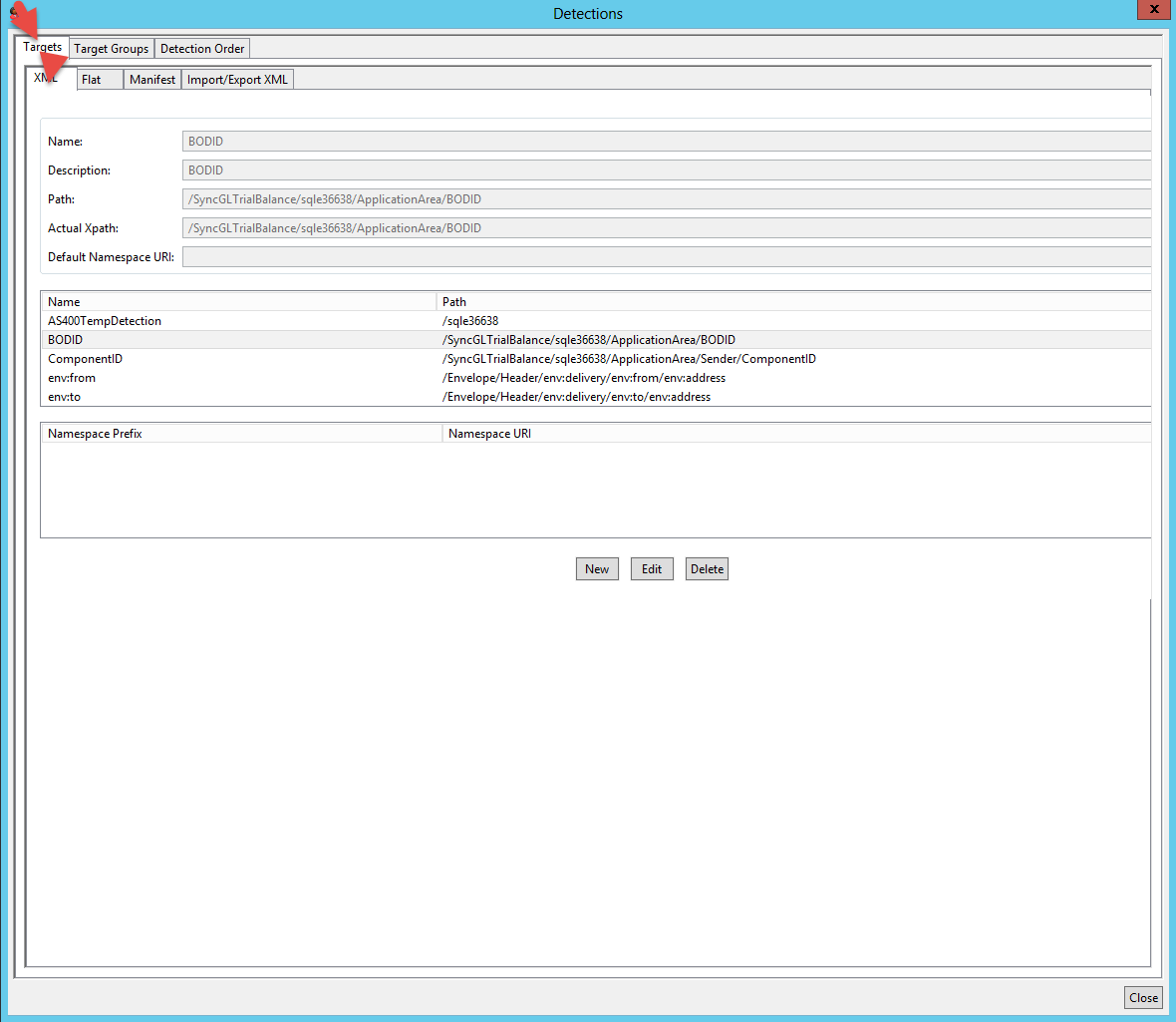
The purpose of this setup is to create a definition of how the agreement can be matched with an incoming (or outgoing) message. The end result is that the agreement will know which tags it should look for and what values those tags should contain in the xml message to identify the message as this agreement. How the system does this is by first defining the individual tags and their path in the xml message. Once the individual tags are defined, the system then requires the definition of the group of tags. This uses the prior definitions of the individual tags and “groups” them together and names the group. The purpose is so that the agreement can refer to a group name which will then point to a defined list of tags. Finally, the system requires the new group to be setup in the detection order. This defines which tag groups are reviewed first. The detection order is sequential from top to bottom.

In our GLTrialBalance example we will define some temporary tags and then add them to a group and then insert the group into the detection order. This example is a temporary detection since the generic xml header was still in development at the time of this document. Once a generic xml header is developed, that definition will only need to be setup once in the detections panel. All agreements would then point to that detection group but would have different values in the tags to define which agreement a message is connected to.

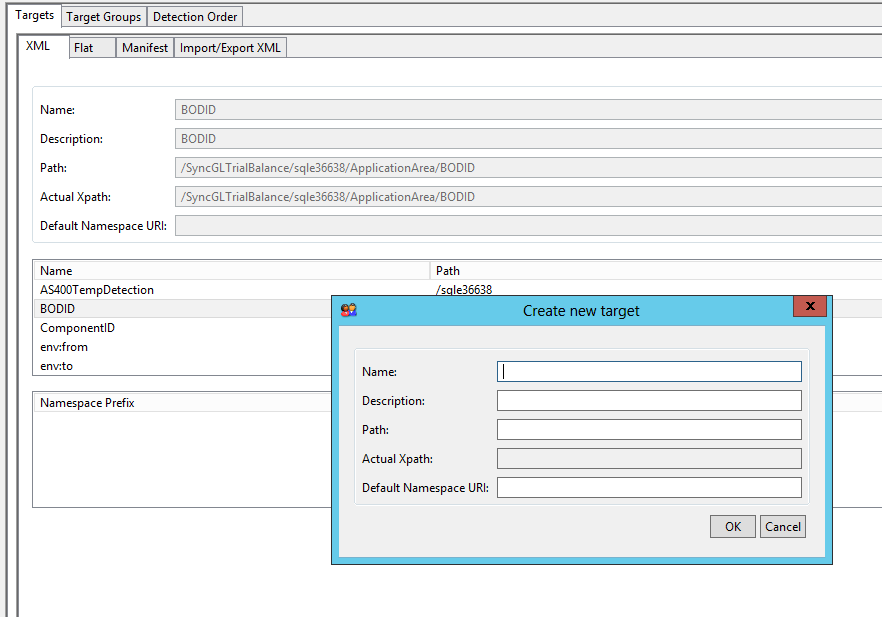
First click the manage menu and choose the detections option.



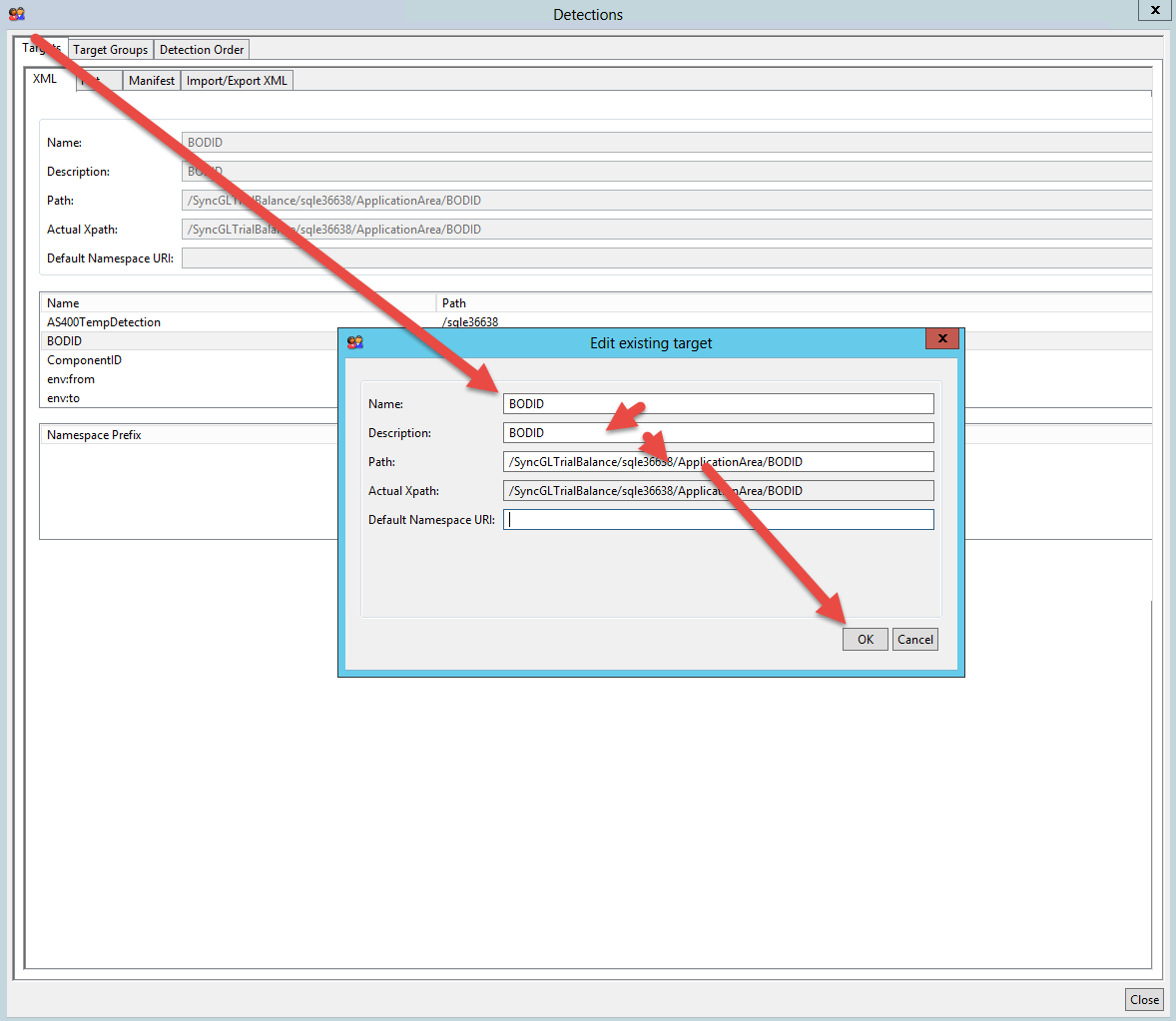
First thing that has to be setup is the individual tags that the agreement will use to identify itself to the xml message. In our example we will use the <BODID>, <ComponentID>, and <SenderLogicalID> tags to do this identification. We will setup these tags and their paths in the first tab, Targets. Since the message we are working with is an XML message we will work in the XML tab.



Next we will click the New button at the bottom of the panel and we will start to define the first tag

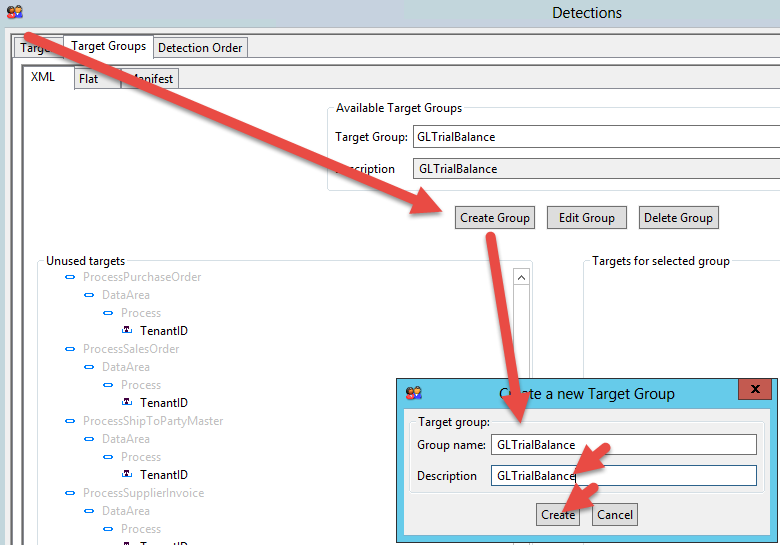


The name field can be anything you want. It is simply a descriptive tag that will identify this target. The description field can be more elaborate and give a deeper explanation of the target. The path field needs to be filled out to point to the path MEC should take to get to the xml tag we are identifying. For example, to get to the BODID tag in the xml, we need to first pass through the <SyncGLTrialBalance> tag, then <sqle36638> tag, then <ApplicationArea> tag, then finally <BODID> tag. So in this example the path would look like “/SyncGLTrialBalance/sqle36638/ApplicationArea/BODID”. Once all this is entered, click the ok button at the bottom of the panel

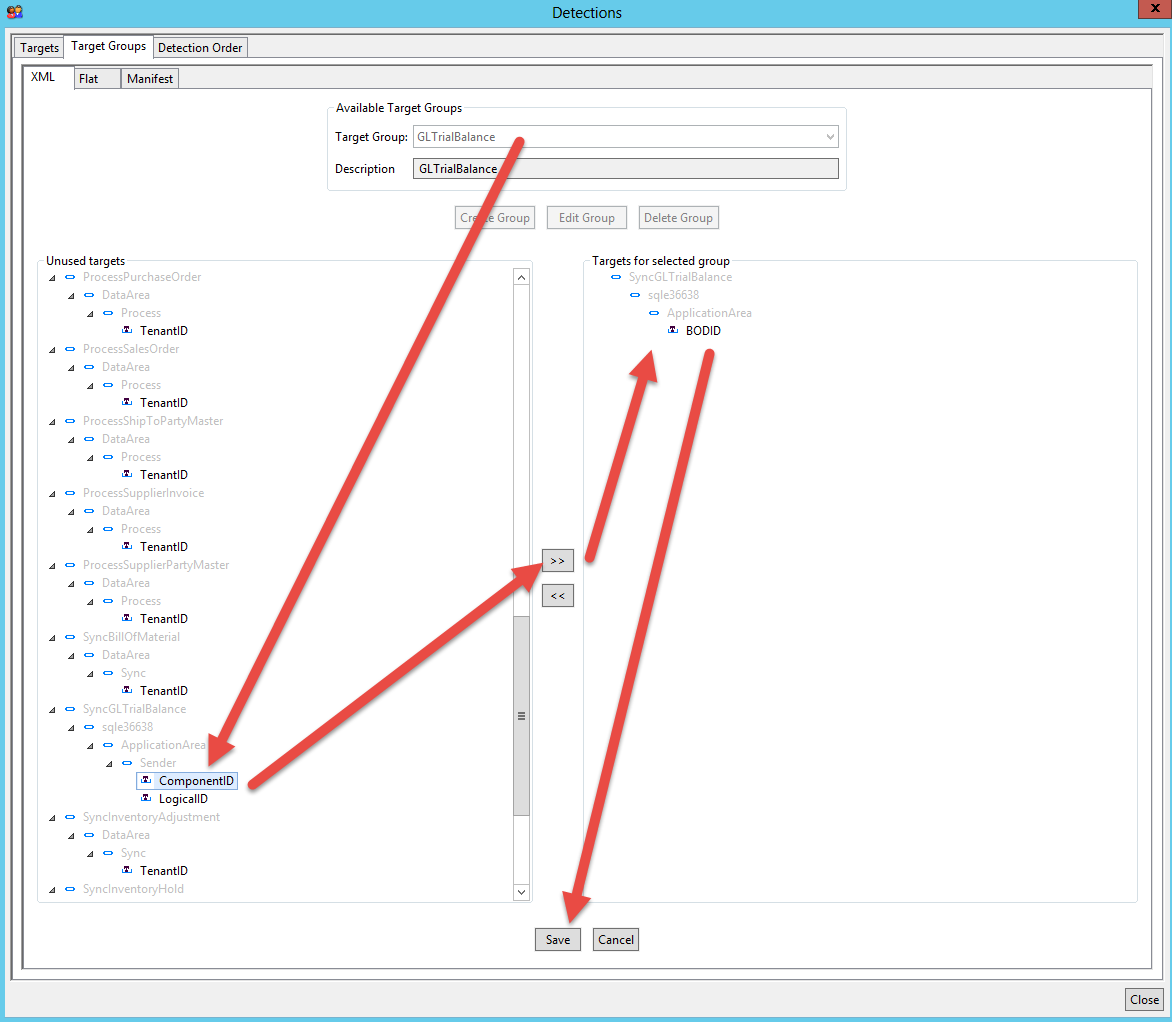


This process would be repeated for each of the xml tags that the agreement will use in the detection

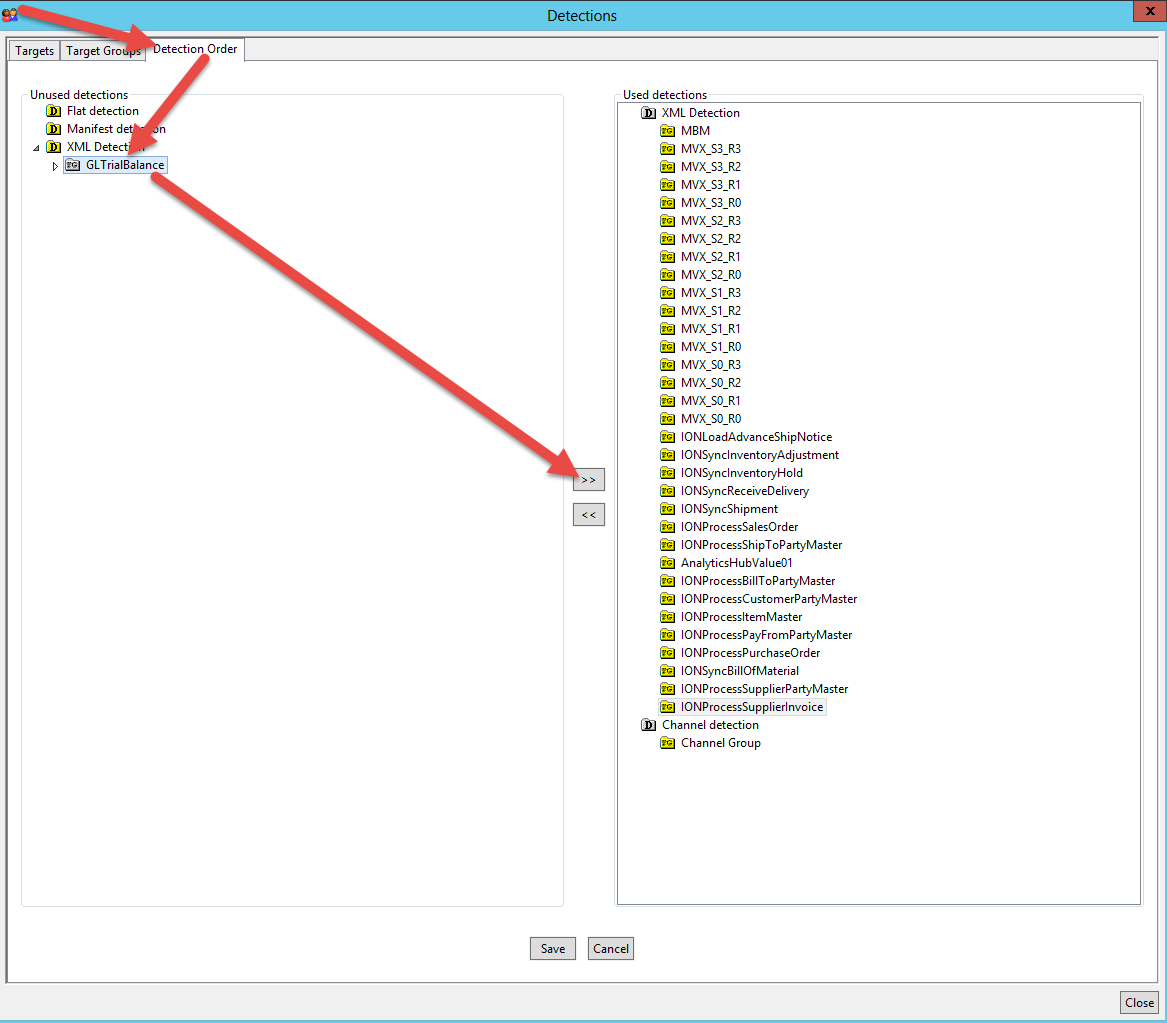
Once all the tags are defined, then we want to put the tags together into a group. For this we will choose the target group tab at the top of the detections panel. Again we will stay in the XML section since this is an xml message. The xml tags that were just setup in the target tab will show up on the left side of the panel under the unused targets. First click the Create Group button. Enter the Group name you want to use to identify this group. If you want you can enter a description that is more detailed. Then click the Create button.



Once this is done the new group will show up in the Target Group drop down. Select your new group from this drop down and Scroll through the list and find the newly defined tags and click to highlight them one at a time. Each time one is highlighted, click the double arrows (>>) to move the xml tag over to the Targets for selected group. Once all the tags are moved over then click the save button at the bottom of the panel.



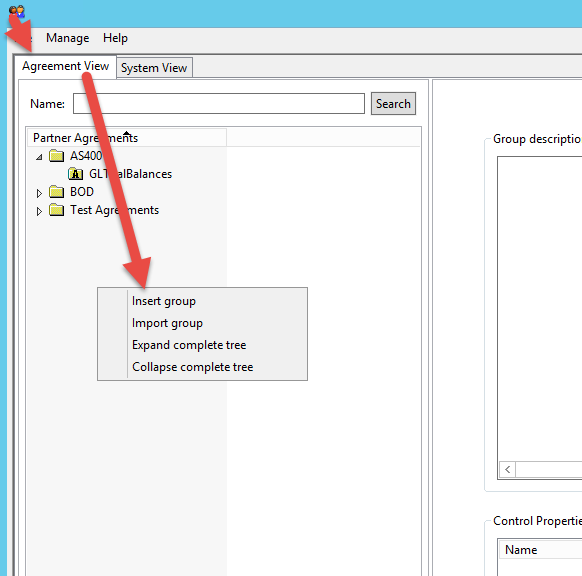
Once a group is setup then it needs to be put in the detection order. MEC uses this order to search for an agreement that matches the xml message. MEC will start at the top of the detection order and work its way down the list until it finds a match. Click on the Detection order tab. The new group will show up on the left had Unused detections list. Highlight the new group and click the double arrow button (>>) to move it to the used detection list. Once it is over in the used detection list, you can click the group and drag it up or down the list.

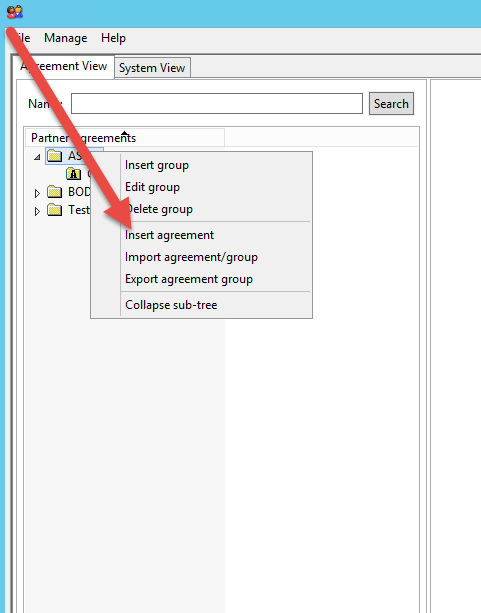


Now that the detection order is setup there is no more configuration needed in the detection panel. In order for these changes to take affect on the MEC service, the service will need to be stopped and started.

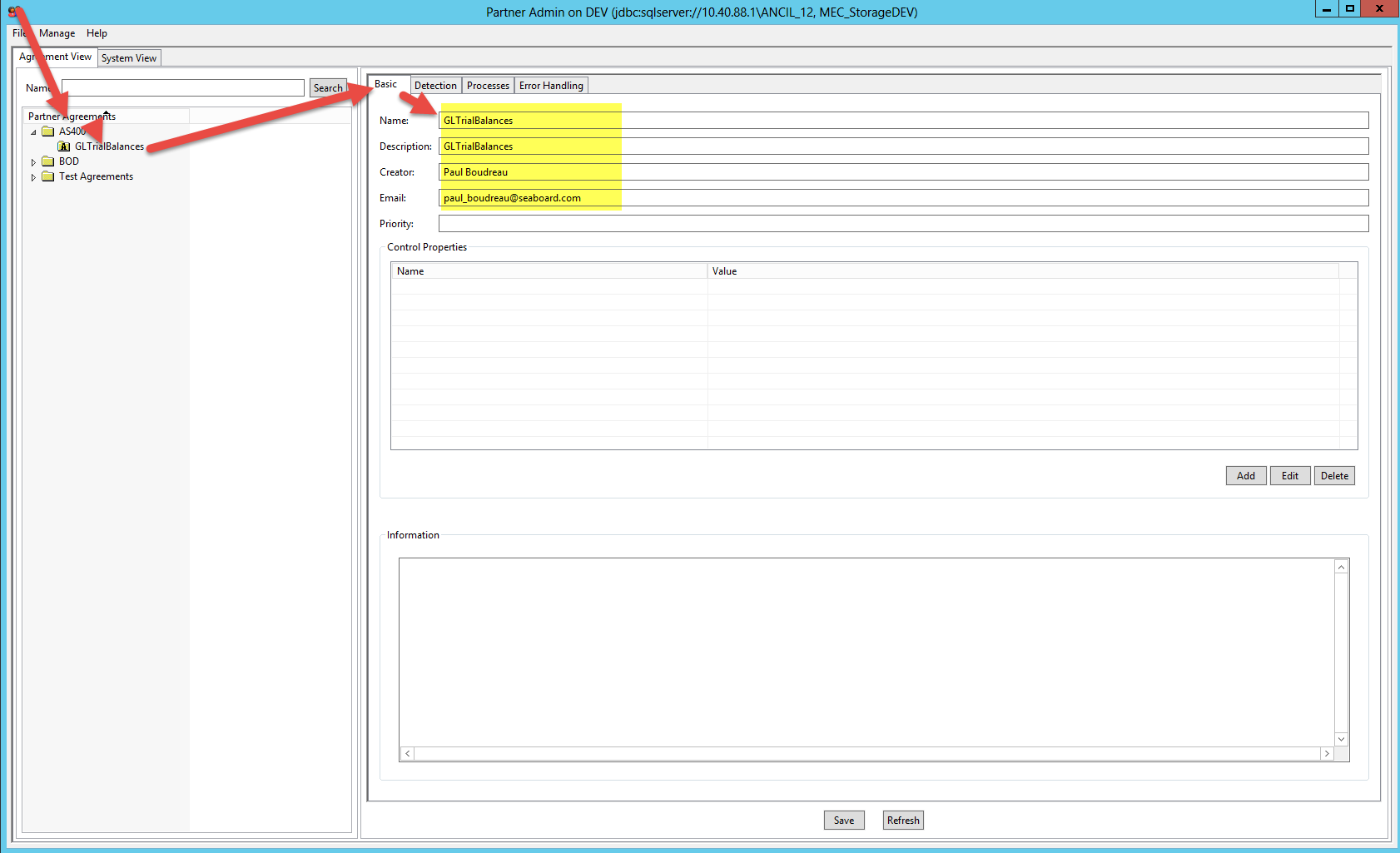
### Setup of the Agreement

Once the detections are defined we can start creating the agreement. We will need to go through the Basic, Detection and finally the process tabs before we save the new agreement. First we need to create a new agreement group. This is not necessary but it is recommended to group agreements by the partner (or integration system) that the agreements are related to. Right click within the left Partner agreements list and choose Insert group. Once the Group is created, right click on the new group and choose Insert Agreement.

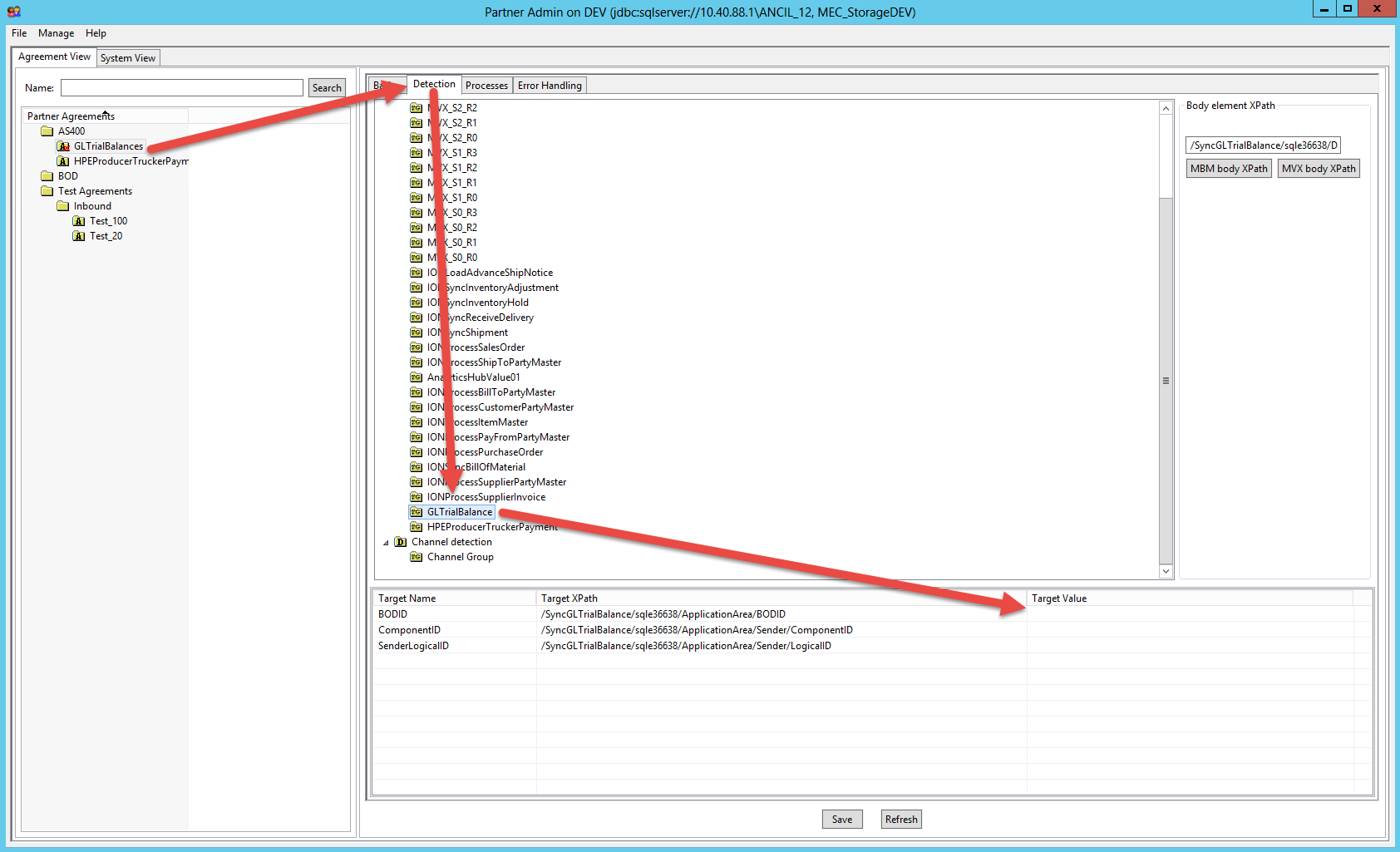




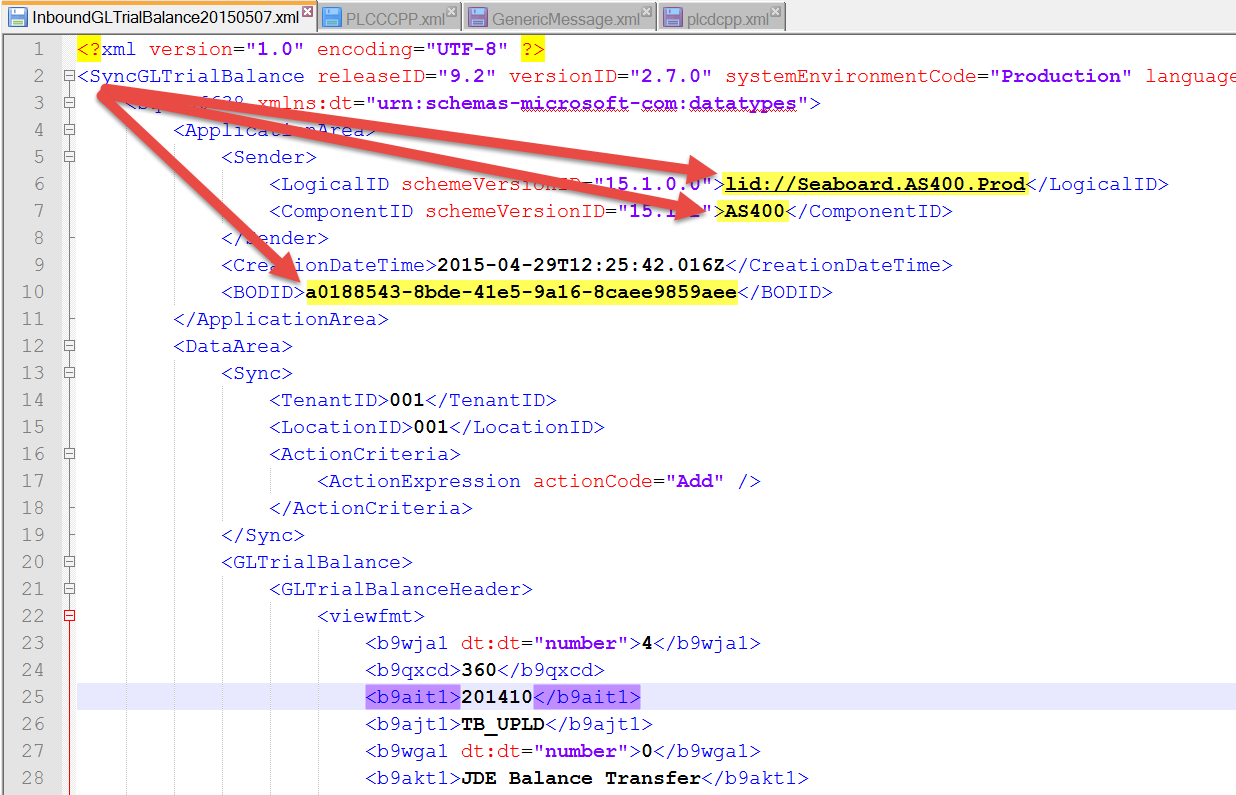
Once the agreement is created in the Partner Agreements left list panel, highlight the agreement and type name and description for the new agreement in the Basic Tab on the right of the partner agreement list. Additionally it is always helpful for future maintenance to enter the name and email address of who created the agreement.

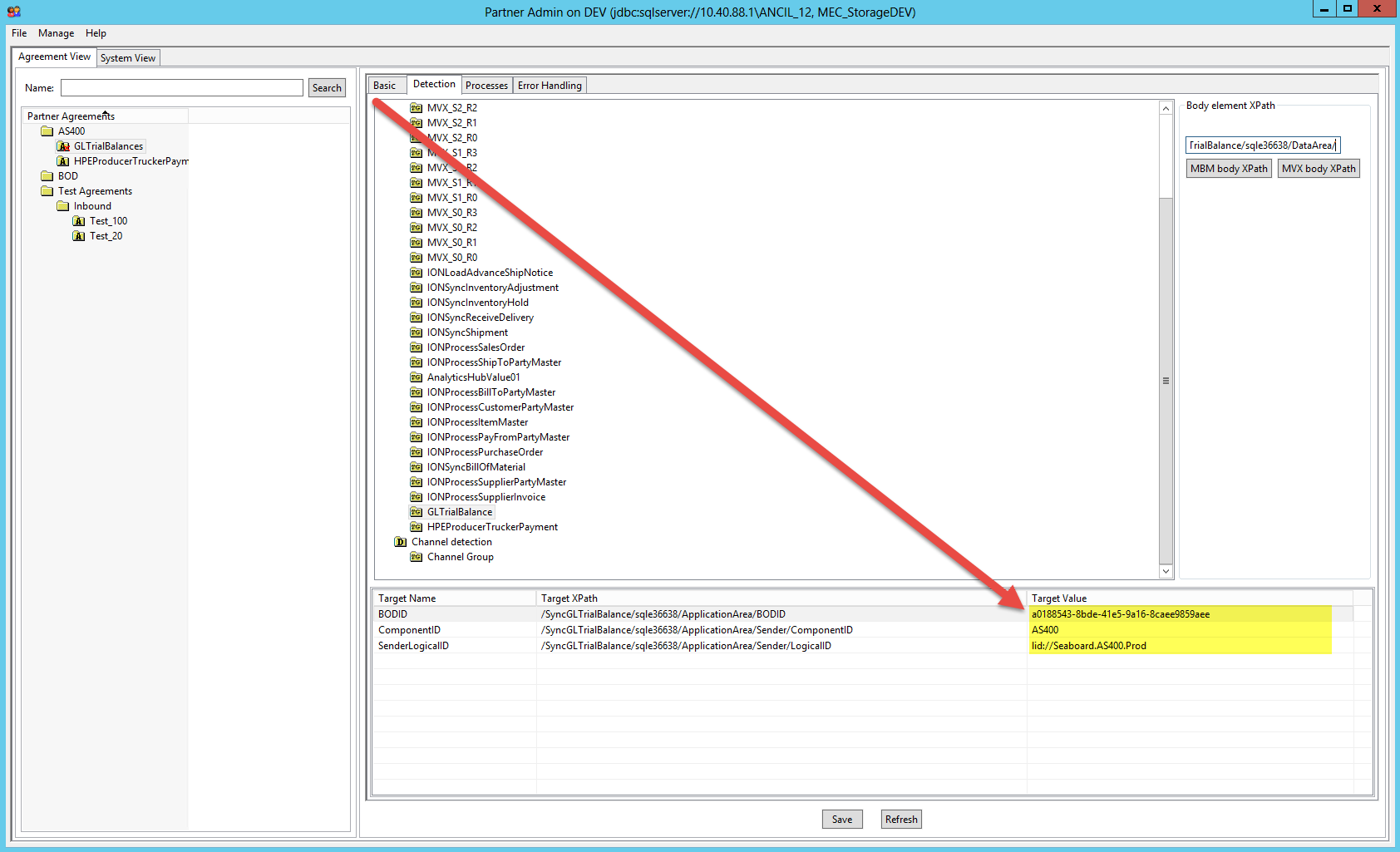


After that information is entered, click the detection tab. Pick the new target group that was just defined. The values for the xml targets need to be entered in the Target Value column.

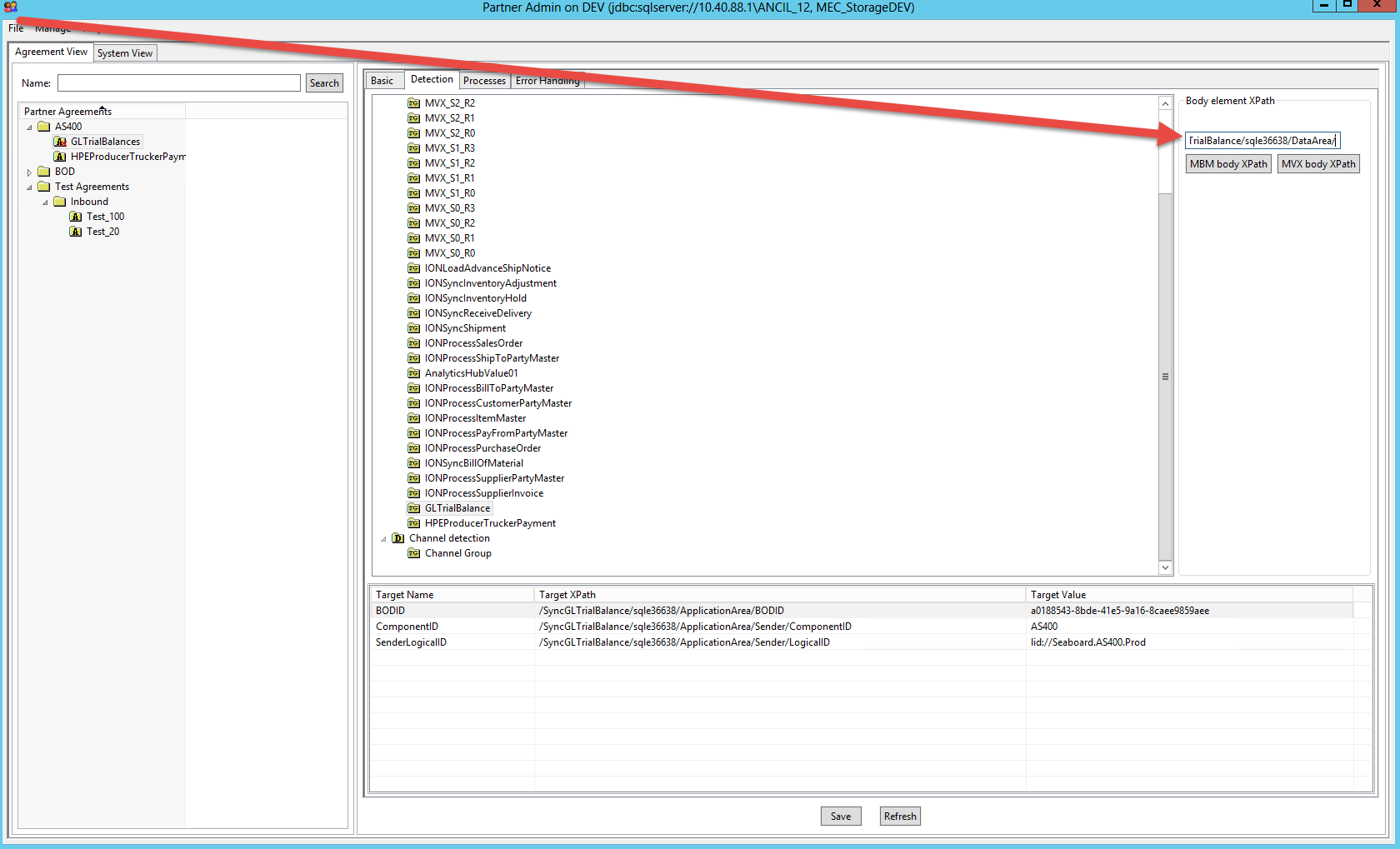


Open the sample xml message and look at what it contains in the target tags. It is best to copy and paste the values from the xml message directly into the PAT fields in order to avoid typo’s

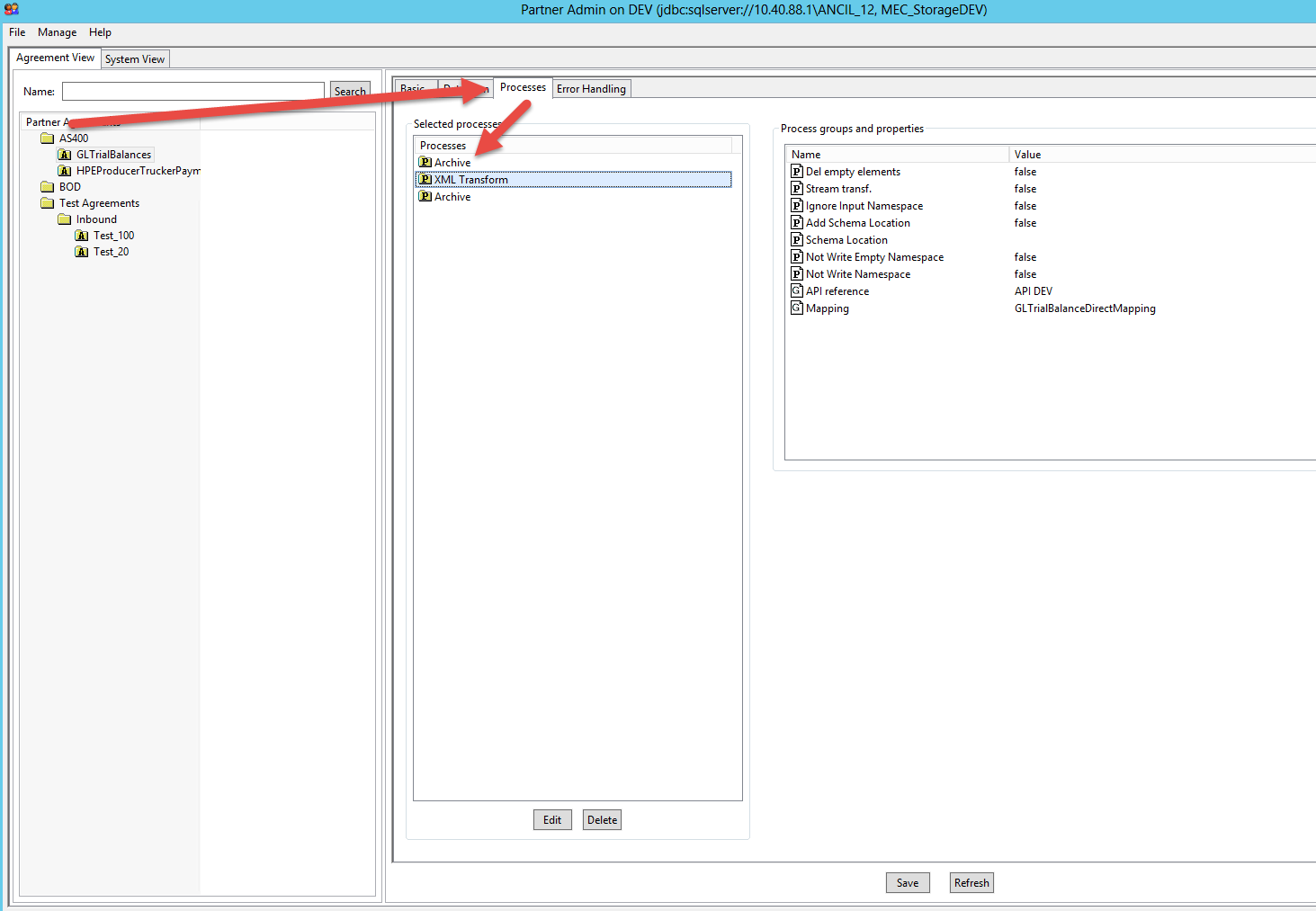




After this is entered the path to the detail xml tag in the body element xpath field

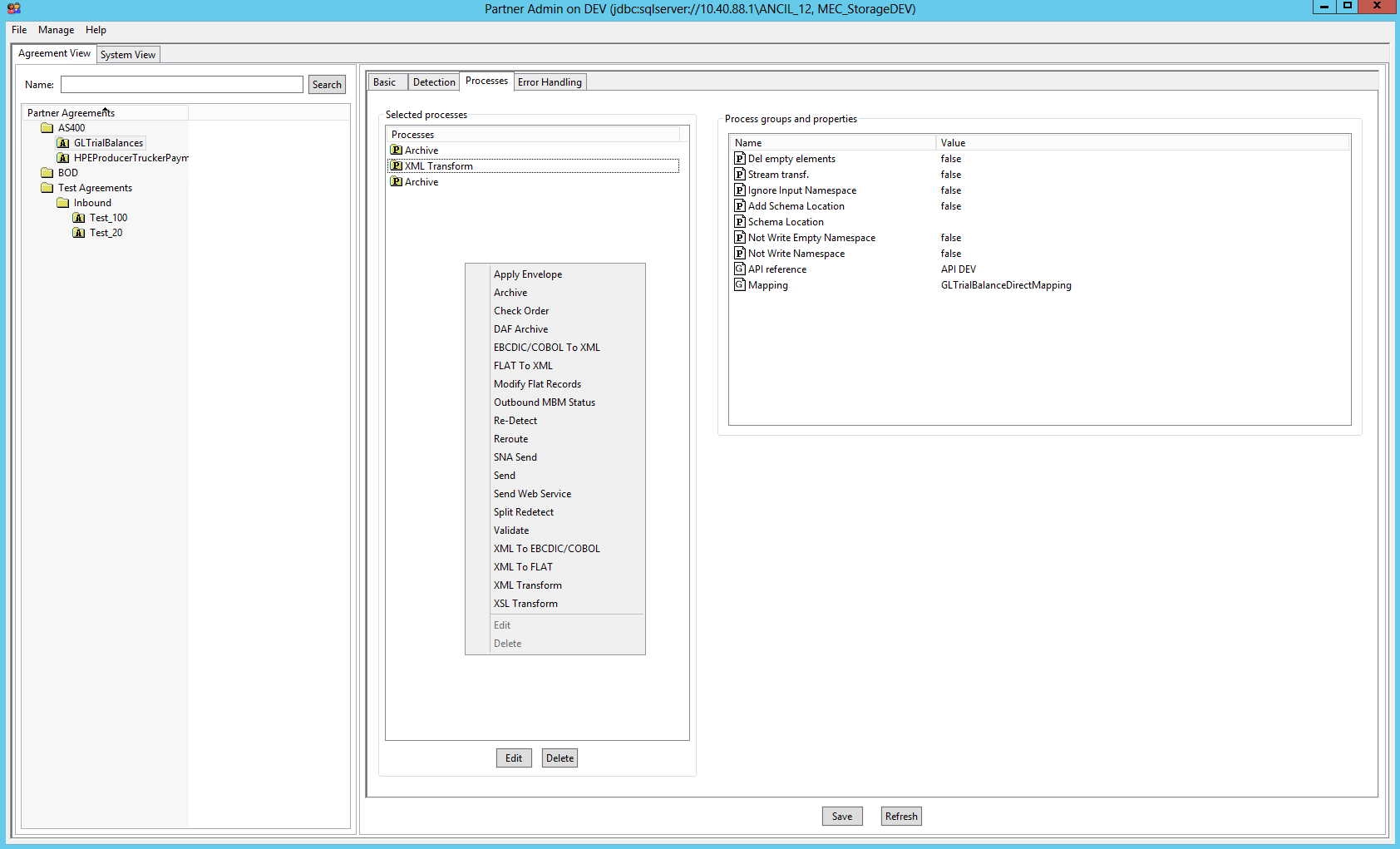


Next click on the Processes Tab. This is where you will tell MEC what to do once it has determined which agreement applies to the XML message. There are many different process that can be used on a message but for an inbound message the most important one is the xml transformation. In this process you will define for MEC which mapping it will use on the message. In our example we added an archive process before and after the xml transformation process.

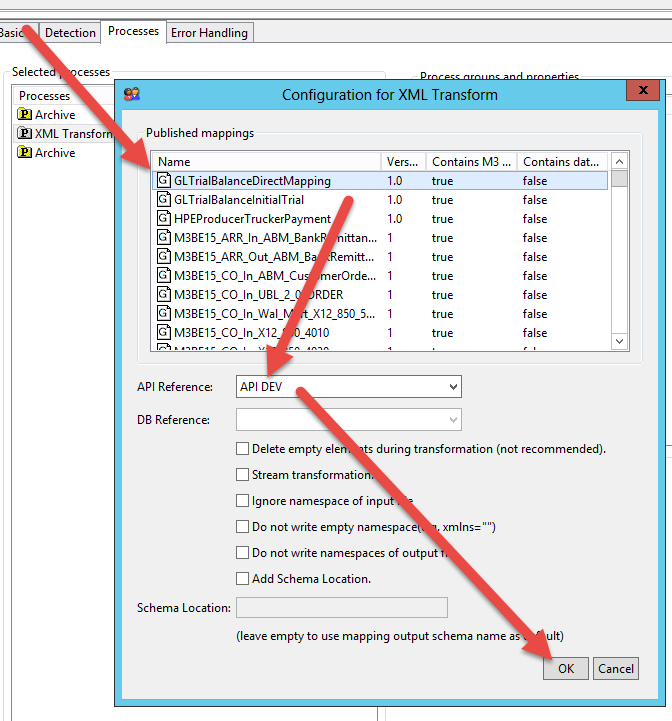


Right click in the blank area in the Selected Process list area and choose the process you want to add. The processes are executed from top to bottom.

When adding the xml transformation, you need to identify which mapping this process will use.



Select the mapping from the published mapping list and make sure the API reference is pointing to the correct M3 server.



## Testing a sample message