Nasdaq Calypso

Messaging Framework Integration Guide

Version 17 - 18

Revision 6.0
February 2022
Approved

Document History

Revision	Published	Summary of Changes
1.0	January 2015	First edition for version 3.10.0 of Data Uploader.
2.0	October 2015	Updated Trade Workflow FCM.
3.0	April 2017	Add Support to start all components in process.
4.0	October 2017	Added Messaging Server Configuration.
5.0	December 2018	Updates
6.0	February 2022	Updated for version 17.0.

Table of Contents

Introduction

This section describes the configuration setup for any external module using Messaging Framework.

Messaging Framework is based on real time asynchronous messaging using JMS and Camel. Messaging Framework can be used in the

- Intra Day Support for Clearing Member with Credit check (limit and initial margin) (aka as Head Room Check – FCMHRC Flow)
- Intra Day Support for Clearing Member without Risk (aka as FCM Flow).

Messaging Framework uses the following components.

- Messaging Server: This component acts as a Message Bus for the other components of the Messaging Framework, Active MQ is used as the Message Bus. All the components of the messaging framework use this Message bus to exchange messages and process them. Messaging server can be started using the scripts as explained below.
- Incoming Feed: Incoming Feed is the first component, which consumes messages from the CCP. This component ensures that the message complete, it validates and extracts the Unique ID of the message to ensure that the messages with the same id are processed in the order they are received and then passes to the bus to be consumed by the next component which is Feed Translator.
- Feed Translator: Feed Translator component translates the incoming message using the Data Uploader framework and creates an in memory trade, which is now ready for Credit Check. Feed Translator sends the message to the HRC or to the Persistor based on whether Credit Check is part of the solution.
- Data Persistor: The job of this component is to save the objects into Calypso's Transactional Core. If for some technical reason the persistor is not able to persist the object successfully, the persistor creates a BO Message of type DATAPERSISTORMSG so that the message can be reprocessed at a later time. In this scenario, if Persistor receives any subsequent messages for the same Trade, it will create PENDING BO Messages for these messages as well and queues them to be processed in order.
- Outgoing Feed: Outgoing Feed generates the acknowledgement (Consent Granted/Consent Refused) and sends it back to the CCP.
- Update Manager Engine: This component is a Calypso Engine which listens to Real Time Events from Calypso. The job of this component track Transactional Core events and replay them to the Message Bus. If the Update Manager is not able to send the message to the Message Bus, a) if the event belongs to a trade, the Update Manager Engine moves the trade to SUBMIT_FAILED status so that the trade can be submitted at a later time. b) If the event belongs to an object which does not have a workflow (like Limit Update / Collateral Update) then the Update Manager engine creates a BO Message with type (UPLOADEREXCEPTIONMSG) so that the message can be reprocessed at a later time.

[Note: The subsequent sections talk about configuration needed for enabling Messaging. In addition to these, you do need to carry out steps for individual Feeds as documented in the respective

documentation. E.g. for CME some business configuration e.g. mappings is still needed. Some areas are replaced. E.g. Trade Workflow and BO Message workflow should not be done as per the CME integration guide. These should be done from the Messaging Integration document.]

In short, read both the documents and if a config is provided in both (Exch Feed and Messaging), use the config from Messaging setup.

For more details on the FCM and FCMHRC flows please refer to the flow diagrams in the Appendix.

Configuration Files

The following configuration files are required by Messaging Framework, this section will describe all properties in the configuration file that messaging framework uses. All the configuration files should be deployed to client and server environments.

[Important Note

You need to copy all the configuration files you want to modify from `<calypso home>/client/resources` **to** `<calypso home>/tools/calypso-templates/resources`**.**

In some cases, you need to rename the configuration file from `<file name>.sample` **to** `<file name>`**.**

Then you need to re-deploy the application servers.]

2.1 <Service>Service.properties

The IncomingFeed component is run using a command line parameter '-service' and expects a parameter which is mandatory. The value of this parameter is used to identify the <>Service.properties with the value prefixed to the name of the properties file.

E.g., if the parameter is set as –service CME, IncomingFeed looks for a property file CMEService.properties with the following properties.

```
uploadSource=CME

# The Value of this attribute indicates from where the message originates; do not change the value of this attribute in the files supplied by Calypso. This value is used by the translator to identify the mappings in the Calypso Mapping Window.


uploadFormat=

# The Value of this attribute indicates the format; do not change the value of this attribute in the files supplied by Calypso. This value is used by the Data Uploader framework to identify the translator to translate the message. It is valid for this attribute to be left empty in certain cases.


ExternalURIPrefix=CMESource

# The Value of this attribute specifies the config name [Spring bean] to use from the file datauploaderExternalURI.xml This XML file contains Messaging definitions to connect to the CCP using JMS, IBM MQ, etc. The config name used in this XML file to create the Messaging definitions is the value to be assigned to this attribute. [See the next section for more details.]


ExternalIncomingURI=

# Please change the value of this attribute to the queue name from which the CCP sends the message to the Clearing Member.


## Provide a sample of what the queue would look like
```

```
ExternalOutgoingURI=

# Please change the value of this attribute to the queue name which the CCP expects the acknowledgement


BusinessFlow=HRC

# The Value of this attribute indicates whether the Credit Check (Collateral and Limit Check) is part of the
Clearing Member Solution. The Possible values are a) FCM (No Head Room Check) b) HRC (Head Room Check is used
for Risk). Note HRC will be replaced by FCMHRC in the next release (to allow for CCPHRC flows) This value is
important as the Trade workflow is different for these solutions. And based on this, the the Update manager
listens to different Trade Status Messages


Features=AuditOn,AutoRejectOff

# AuditOn specifies that a copy of all Incoming & Outgoing messages should be saved as BO Messages. When set to
AuditOff, in cases where an error or warning occurs during message processing, the BO Message will still be
saved. Please do not change this without advice from Calypso

# AutoRejectOff is an advanced feature and Calypso advice should be sought before changing this value.


ExternalIncomingURIParams=

# The value should be left empty unless a value is provided by Calypso out of the box. This attribute is
mandatory to be present even if this value is empty
```

2.2 datauploaderExternalURI.xml

This XML file contains the connectivity information to the Messaging service used by the CCP.

The following snippet shows a section of the file that is supplied by calypso which explains the connectivity to CME using IBM MQ.

Please Change only the following parameters (highlighted in the snippet below) and leave the rest of the properties as provided by calypso.

- Hostname: to the host name where the messaging server (IBM MQ) is running.
- Port: to the port on which the messaging server is running.
- queueManager: Name of the Queue Manager on the Messaging Server (IBM MQ)
- channel: Name of the Channel used to connect to the Messaging Server (IBM MQ)

```
<bean id="CMESource" parent="wmq">
		<property name="concurrentConsumers" value="${INCOMING_CONSUMERCOUNT:1}"/>
		<property name="maxConcurrentConsumers" value="${INCOMING_CONSUMERCOUNT:1}"/>
	</bean>


	<bean id="wmq" class="org.apache.camel.component.jms.JmsComponent">
		<property name="connectionFactory" ref="cachedConnectionFactory"/>
```

```
        </bean>

        <bean id="cachedConnectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
                <property name="targetConnectionFactory" ref="jmsConnectionFactory"/>
                <property name="sessionCacheSize" value="10"/>
        </bean>

        <bean id="jmsConnectionFactory" class="com.ibm.mq.jms.MQQueueConnectionFactory">
                <property name="transportType" value="1"/>
                <property name="hostName" value="PLEASE CHANGE TO THE HOST"/>
                <property name="port" value="PLEASE CHANGE TO THE PORT"/>
                <property name="queueManager" value="PLEASE CHANGE TO THE QUEUE MANAGER"/>
                <property name="channel" value="PLEASE CHANGE TO THE CHANNEL NAME"/>
</bean>
```

Note that the reference used to define the connectivity in the definition `<bean id="CMESource" parent="wmq">` should be set as the value of 'ExternalURIPrefix' property in <Service>Service.properties as explained in the previous section.

2.3 datauploaderQueueNames.properties

This configuration file contains the queue names which the messaging framework components such as Incoming Feed, Feed Translator and Data Persist or consume messages from, and is used by internal components only. The user should not change anything in the properties file and use it the way calypso makes it available.

```
InternalURIPrefix=upload
FeedTranslator.queue=calypso.queue.translator
OutgoingFeed.queue=calypso.queue.outgoing
DataPersistor.queue=calypso.queue.persistor
TradeVar.queue=calypso.queue.tradesToValue
```

2.4 gatewayservice.properties

All the components of the messaging framework use data uploader to translate or create BO Messages for audit purposes, hence all the configuration files required by data uploader framework should be properly deployed to both the client and server-side environments. Please refer to data uploader integration guide for setup instructions of the same.

Message Workflow

The following message workflows should be imported using the Calypso Workflow Configuration window. More details of these message types and their workflows are explained in the sections below.

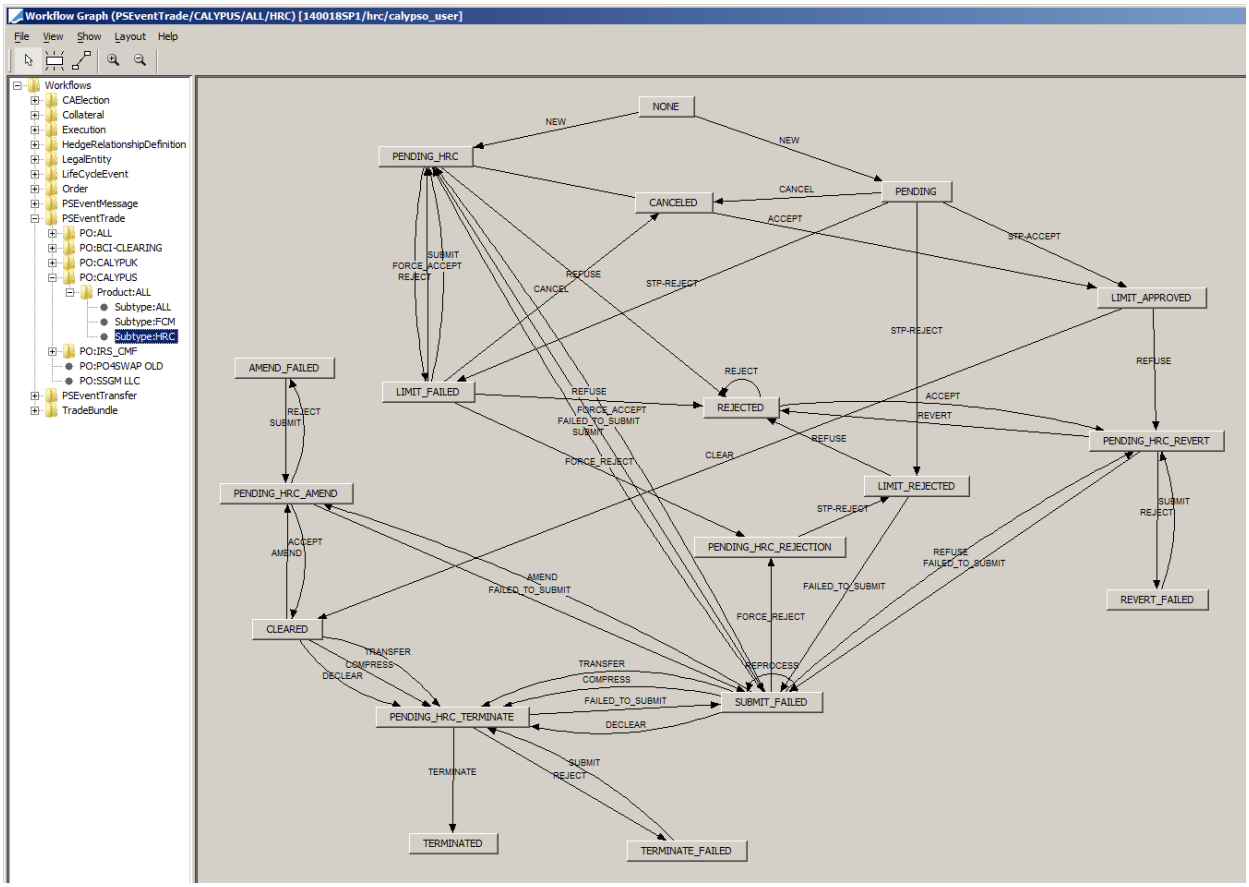
- INCOMINGFEEDMSG.wf (Used to create BO Message for every incoming message, allows the user to reprocess if the message cannot be processed)
- OUTGOINGFEEDMSG.wf (Used to create BO Message for every outgoing message, allows the user to reprocess if the message cannot be processed)
- UPLOADEREXCEPTIONMSG.wf (Used to create BO Message when the Update Manager Engine was not able to send the Collateral and Limit Update events back to the Message Bus)
- DATAPERSISTORMSG.wf (Used by the Data Persistor to create Bo Message when the Persistor was not able to persist the object due to technical reasons)

Trade Workflow

The following trade workflows should be imported using Calypso Workflow Configuration window.

4.1 TRADE_WORKFLOW_FCMHRC.wf

Used in the Clearing Member Solution with Head Room Check used for Risk.

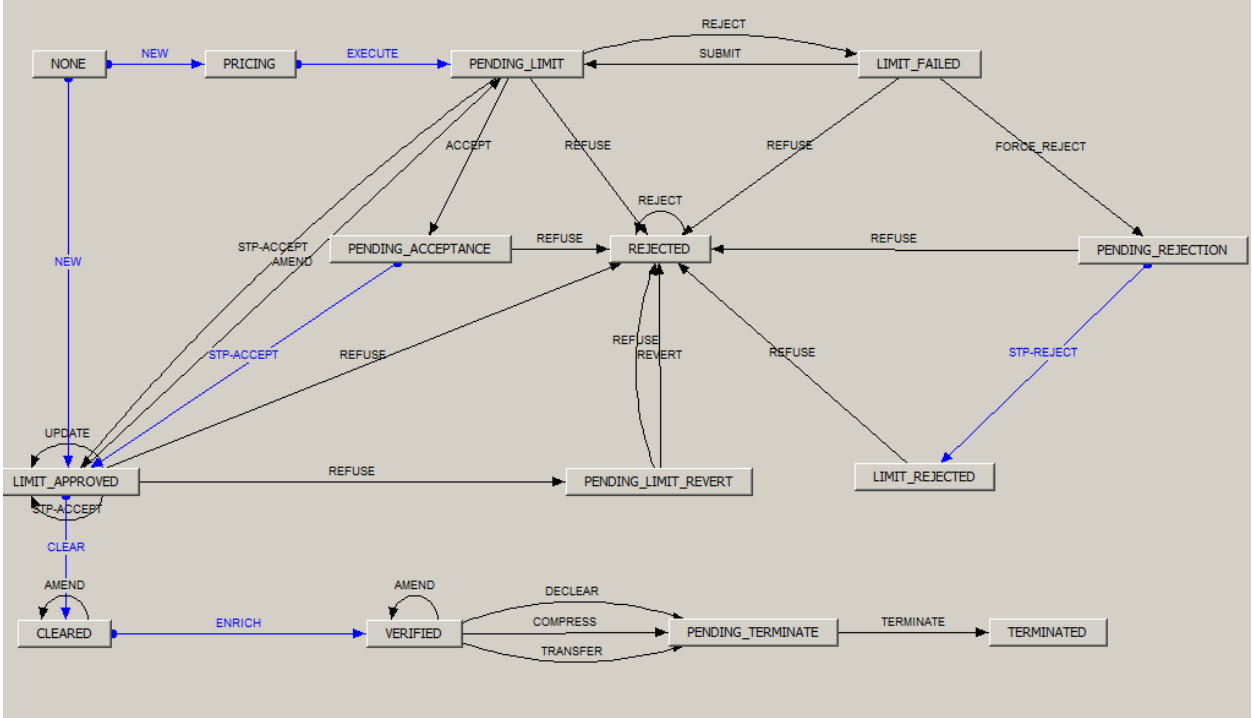


Though the workflow is configurable the following statuses are important and should not be changed. The Update Manager Engine listens to these status events and propagates the message to the Head Room Check component via the Message Bus.

- PENDING_HRC
- PENDING_HRC_REJECTION
- PENDING_HRC_REVERT
- PENDING_HRC_TERMINATE
- PENDING_HRC_AMEND

4.2 TRADE_WORKFLOW_HRC.wf

Used in the Clearing Member Solution without Head Room Check.



Though the workflow is configurable the following statuses are important and should not be changed. The Update Manager Engine listens to these status events and propagates the message to the OutgoingFeed component via the Message Bus.

- PENDING_ACCEPTANCE
- PENDING_REJECTION

Starting the Messaging Components

Messaging Framework requires many processes to be run depending on the FCM Solution being used, however the following processes are common across the solution.

- Messaging Server
- Incoming Feed (one for each exchange CME/LCH)
- Feed Translator
- Data Persistor
- Outgoing Feed

You can start all these integration components in a single JVM process. However please note that each process can be started as a separate JVM process as described in below sections.

5.1 Start All Components in a Single JVM

All the messaging components including the Messaging server can be started by running the following batch file: <Calypso_Home>/client/bin/allFcmMessagingComponents.bat for Windows OS.

<Calypso_Home>/client/bin/allFcmMessagingComponents.sh for UNIX like OS.

The Script has following options:

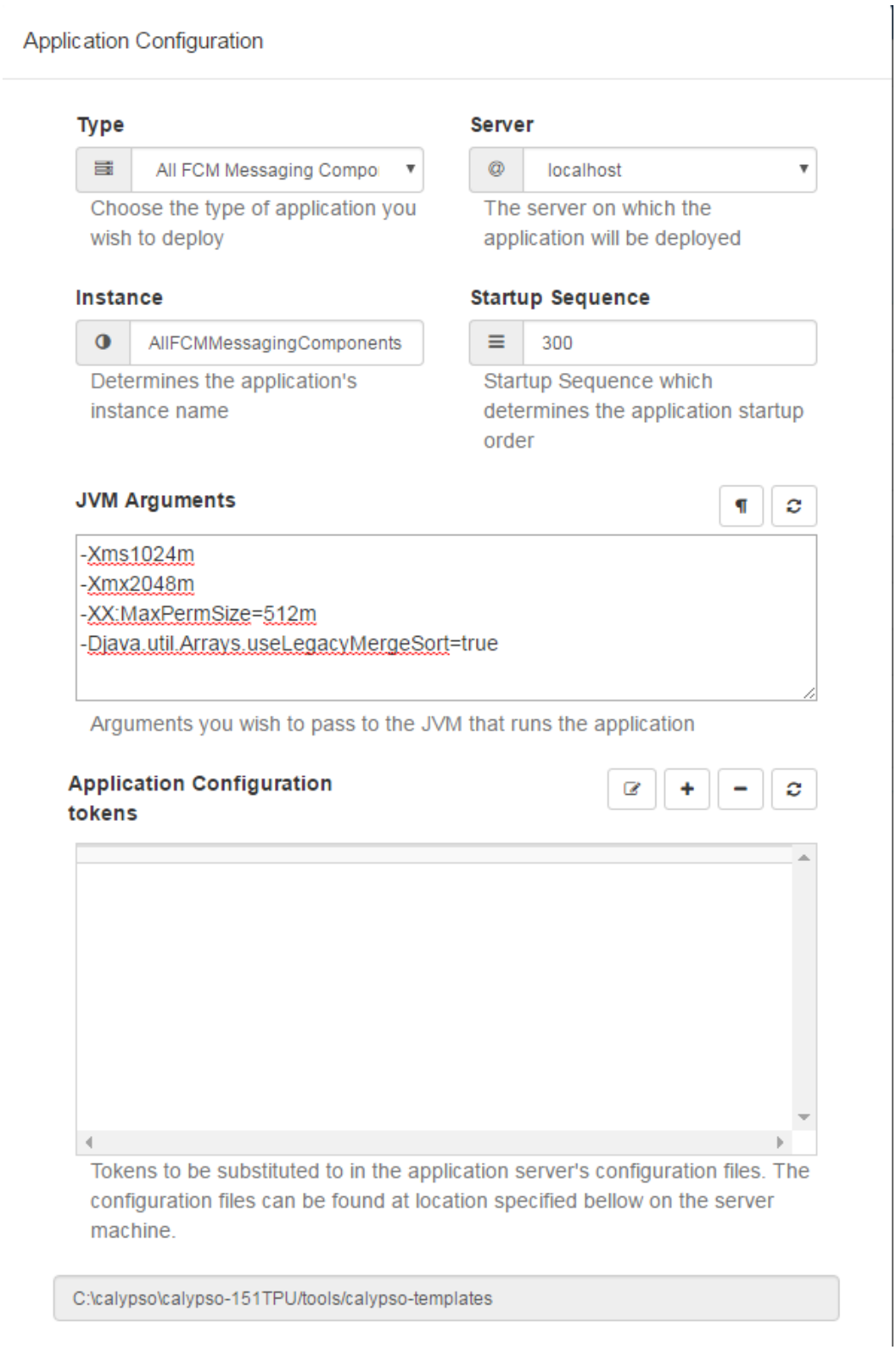
- -startMessageBus true/false: defaulted to false so that the Messaging Server can be started separately as described in next section, but by passing true, the messaging server is also started and there is no need to start it separately.
- -service LCH,CME,CCE: takes comma separated values of services, this parameter is optional and the Incoming Feeds for each exchange can be started separately as explained below, but when used the incoming feeds for the respective exchanges are also started in the single JVM. If this property is not used, the Incoming Feeds are not started and they have to be started separately as usual.

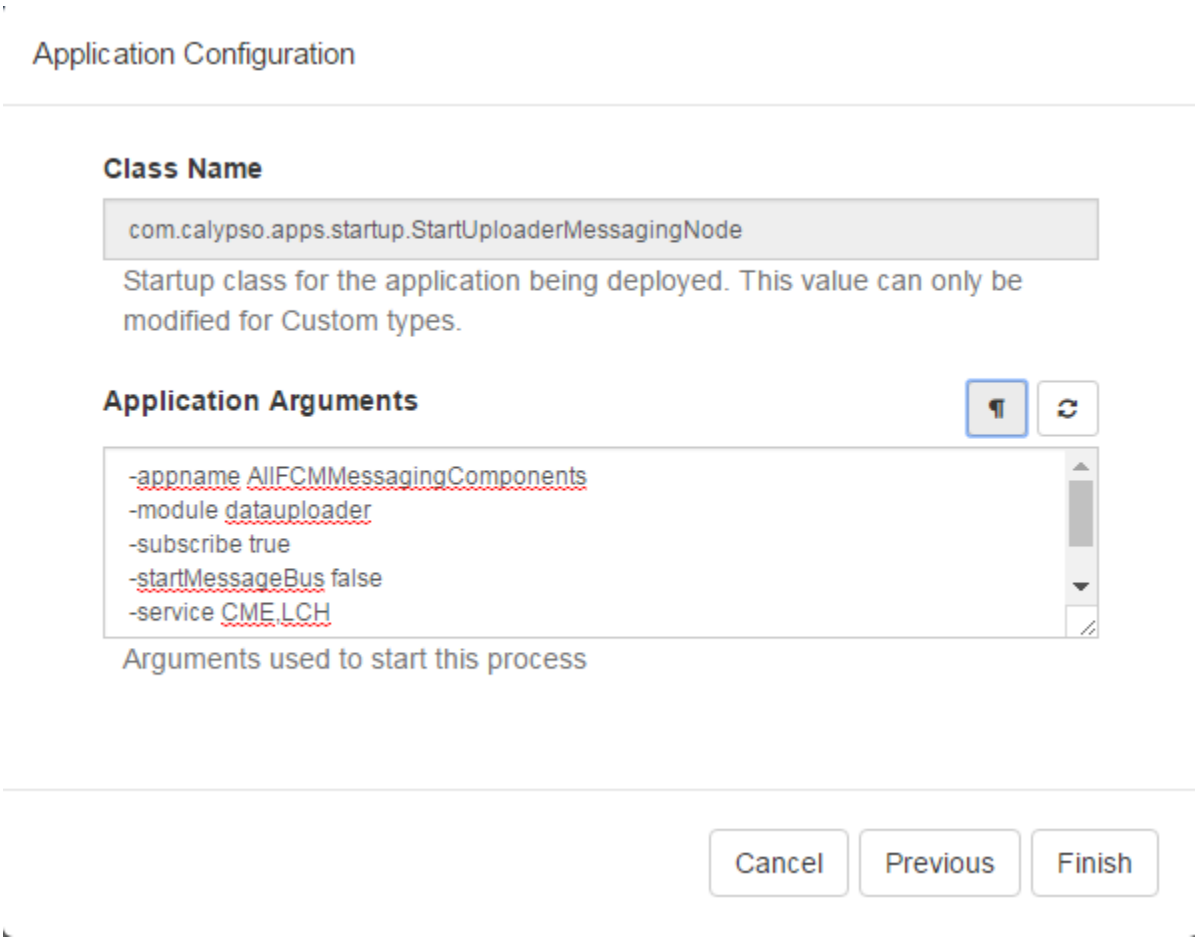
And the components Feed Translator, Data Persistor, Outgoing Feed are started in the same JVM.

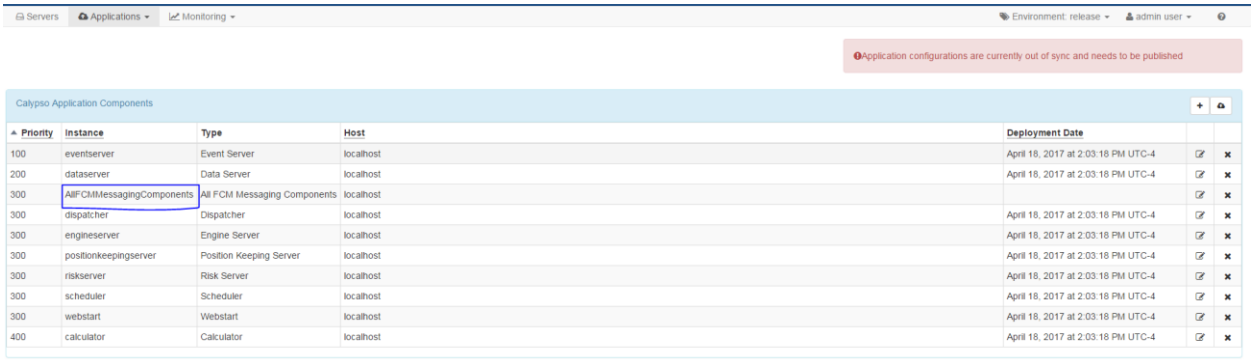
If the Clearing Member Solution is using Head Room Check for Limits then a different start-up script supplied by the Head Room Check module should be used, which starts all the FCM based components and the limit components like tradeVar & head room check.

This Process can also be controlled (start/stop) using the Technical Operations Dashboard, by adding as an Application.

The following screenshots describe on how to add this as an application in the Operations Dashboard.


Application Configuration
Type
All FCM Messaging Compo
Choose the type of application you wish to deploy
Server
localhost
The server on which the application will be deployed
Instance
AllFCMMessagingComponents
Determines the application's instance name
Startup Sequence
300
Startup Sequence which determines the application startup order
JVM Arguments
-Xms1024m
-Xmx2048m
-XX:MaxPermSize=512m
-Djava.util.Arrays.useLegacyMergeSort=true
Arguments you wish to pass to the JVM that runs the application
Application Configuration tokens
Tokens to be substituted to in the application server's configuration files. The configuration files can be found at location specified bellow on the server machine.
C:\calypso\calypso-151TPU/tools/calypso-templates





Once the Application is added in the Dashboard, it can be started, stopped and the logs can be viewed from the Operations Dashboard screen in the browser.

The Next Sections explain on starting these processes each in a separate JVM.

5.2 Start Messaging Server

The Messaging server is used to store the incoming messages in queues so that the messaging framework components can process them asynchronously. Active MQ is used as the Messaging Bus.

By default, the messaging server will run with the following host and port - tcp://localhost:61917

5.2.1 Messaging Server Configuration

The following parameters can be passed while running the UploadMessagingServer as VM arguments as shown below:

```
com.calypso.apps.startup.StartUploadMessagingServer -D<parameterName=value>
```

Parameter	Description
DU_MESSAGING_SERVER_JMX_HOST	Indicates the JMX host Default = localhost
DU_MESSAGING_SERVER_JMX_PORT	Indicates the JMX port Default = 2001
DU_MESSAGING_SERVER_PERSISTENT	True to persist incoming messages or false otherwise **[NOTE: If set to 'false', the incoming messages are not persisted at the messaging server level, and if the application is shutdown due to technical failures, these messages may be lost]** Default = true
DU_MESSAGING_SERVER_PERSISTER_DIRECTORY	Only applies if DU_MESSAGING_SERVER_PERSISTENT = true The directory where the messages and server persistence logs are created This folder should be monitored closely as it may fill the disk space Default = ${user.home}/Calypso/DU_MESSAGING_SERVER
DU_MESSAGING_SERVER_QUEUE_STORAGE	The memory usage of storage queue Default = 3GB
DU_MESSAGING_SERVER_SYSTEM_STORAGE	The memory usage of system queue Default = 20GB
DU_MESSAGING_SERVER_TEMP_STORAGE	The memory usage of temporary queue

Parameter	Description
	Default = 300MB
DU_MESSAGING_SERVER_URL	Indicates the Messaging Server URL Default = tcp://localhost:61917

5.2.2 Clearing Member Solution without Head Room Check

Run the following batch file: <Calypso_Home>/client/bin/StartUploaderMessagingServer.bat for Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/StartUploaderMessagingServer.sh for UNIX like OS.

The System property DU_MESSAGING_SERVER_URL should be used to customize the messaging server host and port. If the components of the messaging solution (like Incoming Feed) are run in different machines, then this System property should be set in all the machines.

5.2.3 Clearing Member Solution with Head Room Check

Run the following batch file: <Calypso_Home>/client/bin/StartHeadroomCheckEventServer.bat for Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/StartHeadroomCheckEventServer.sh for UNIX like OS.

To customize the host and port of the Head room check event server, the System property HRCEVENTSERVERURL should be used. Please make sure that the System property DU_MESSAGING_SERVER_URL is set to the same value. If the components of the messaging solution (like Incoming Feed) are run in different machines, then these System properties should be set in all the machines.

5.3 Start Incoming Feed

Run the following batch file: <Calypso_Home>/client/bin/Start<Service>IncomingFeed.bat on Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/Start<Service>IncomingFeed.sh on UNIX like OS.

5.4 Start Feed Translator

Run the following batch file: <Calypso_Home>/client/bin/StartFeedTranslator.bat on Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/StartFeedTranslator.sh on UNIX like OS.

5.5 Start Outgoing Feed

OutgoingFeed is used to send the acknowledgement to the CCP (CME/ LCH).

Run the following batch file: <Calypso_Home>/client/bin/StartOutgoingFeed.bat on Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/StartOutgoingFeed.sh on UNIX like OS.

5.6 Start Data Persistor

DataPersistor persists objects into calypso database.

Run the following batch file: <Calypso_Home>/client/bin/StartDataPersistor.bat on Windows OS.

Run the following batch file: <Calypso_Home>/client/bin/StartDataPersistor.sh on UNIX like OS.

5.7 Additional Components

Please note that if Clearing Member solution with Head Room Check is being used, few extra processes like TradeVaR and HeadRoomCheck need to be started. Please refer to the Head Room Check Solution Document for more details.

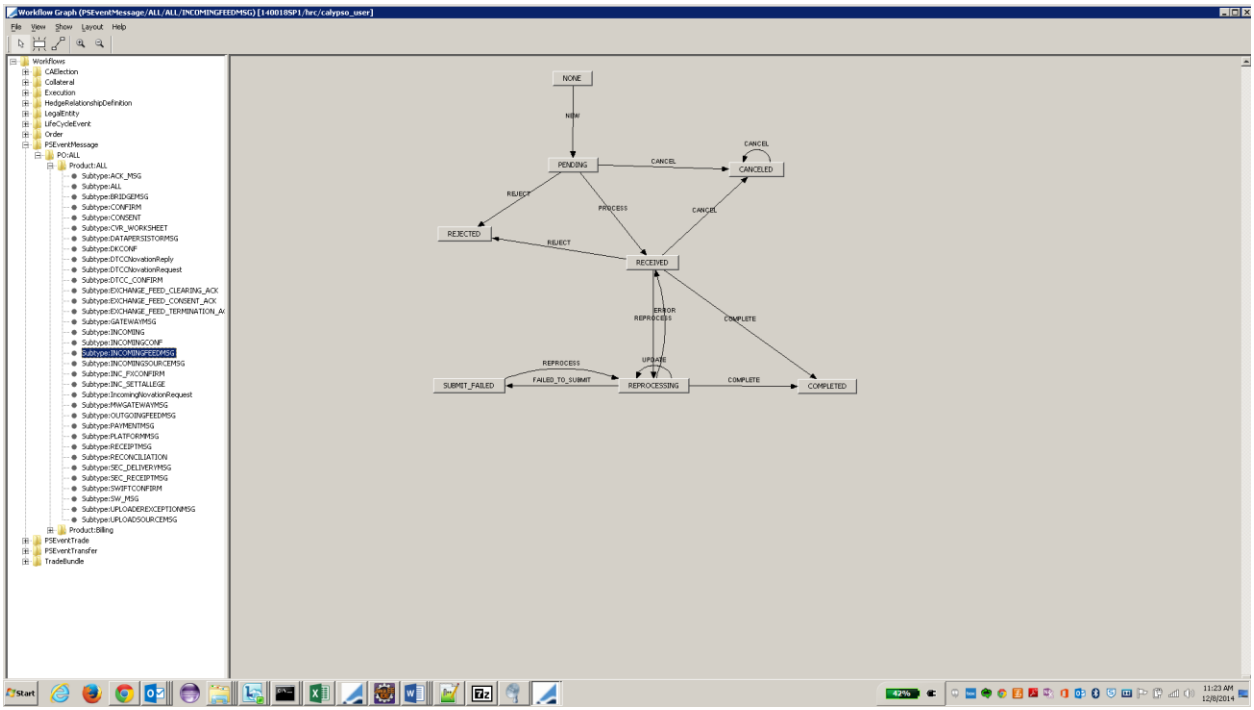
Incoming Feed

The purpose of this component is to receive messages from the external world into calypso (basically in the Clearing Member Solution, receives messages from the CCP).

Once the message is successfully received, IncomingFeed parses the message to get some useful information like the business key that uniquely identifies the message, so that all the incoming messages from the external world are processed sequentially by the rest of the messaging framework.

Exception Handling in Incoming Feed

When an exception occurs in processing the incoming message the Incoming Feed, would send a message to the Data Persistor component to create a BO Message of type INCOMINGFEEDMSG so that the message could be reprocessed once the exception is addressed.



Feed Translator

The purpose of this component is to translate the external message to a format understood by Calypso which is the data uploader xml format. If the translation is successful, an in memory trade object is created, and now the trade is ready to be processed by the Risk system to check for limits and collateral. Feed Translator sends the message to the Head Room Check component or to the Data Persistor based on whether risk is part of the solution.

Exception Handling in Feed Translator

When an exception occurs in translating the incoming message the Feed Translator, would send a message to the Data Persistor component to create a BO Message of type INCOMINGFEEDMSG so that the message could be reprocessed once the exception is addressed.

Outgoing Feed

Once the incoming message is handled successfully by all the components of the messaging framework and the Trade is accepted / rejected, the Acknowledgement is now ready to be sent out using the Outgoing Feed Component. The trade acceptance/rejection will come from either the Head Room Check component or the Trade Workflow, based on whether risk is part of the solution.

Exception Handling in Outgoing Feed

When an exception occurs in sending the acknowledgement message to the external world (CCP), this component would send a message to the Data Persistor component to create a BO Message of type OUTGOINGFEEDMSG so that the message could be reprocessed once the exception is addressed.

Data Persistor

The Data Persistor component's main purpose is to persist objects into calypso data base. There is a designated queue in the Message Bus, to which rest of the components can send messages (all different types of messages like trades, BO messages, collateral, and limits updates and so on). Data Persistor identifies the right persistor class based on the object passed and invokes the persistor to persist the objects. At times there can be a collection of objects passed as a single message to the persistor in which case the persistor persists all the objects as a single transaction so that either all the objects are persisted or none are persisted.

9.1 Exception Handling in Data Persistor

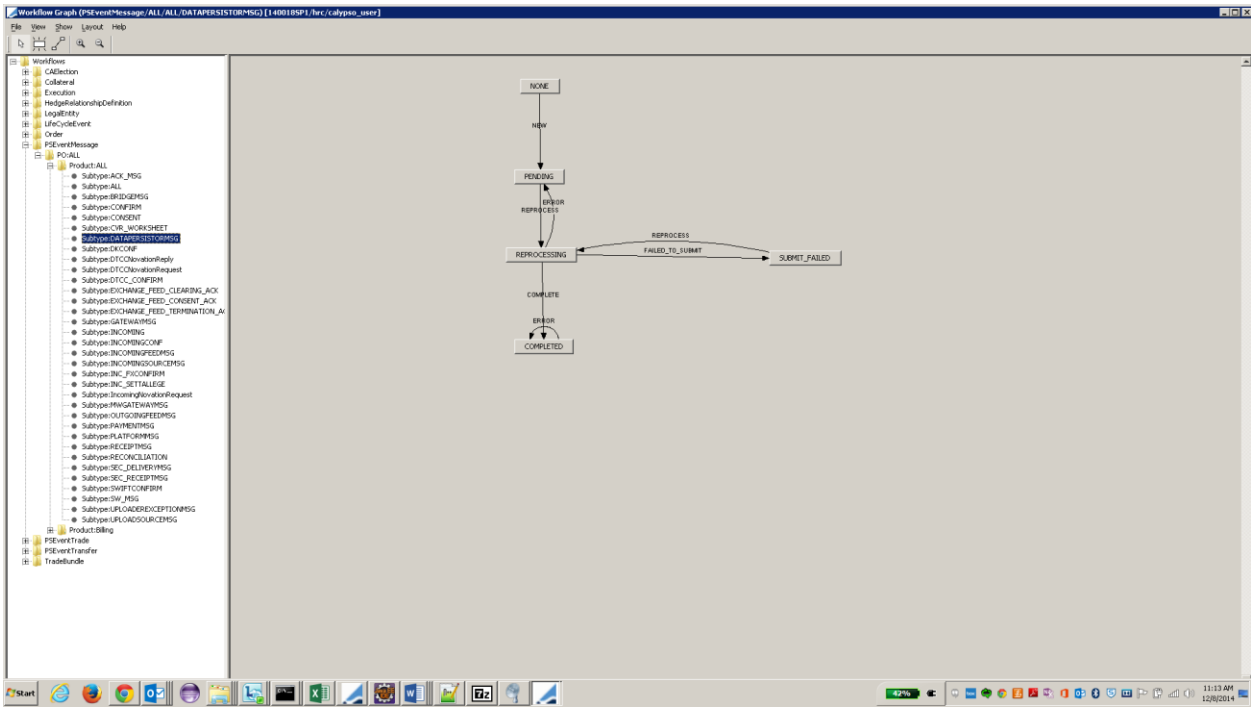
As the persistor tries to persist an object, failures or exceptions might occur because of various reasons, like the workflow is not configured or some other technical reason. Data Persistor is able to handle all these exception scenarios and provide robust functionality.

Data persistor handles exceptions similar to how the Update Manager handles the exceptions, which is by creating a BO Message and persist to data base and create task station entries so that the users are notified and would be able to reprocess the message using the BO Message workflow.

9.2 Exception Recovery

Exceptions are recovered by reprocessing the BO Messages of type DATAPERSISTORMSG by applying REPROCESS action from within the Task Station. This will generate an event, which will be picked up by the Update Manager, and in turn will be sent to the Data Persistor component to retry the save action.

9.3 DATAPERSISTORMSG Workflow



When a DATAPERSISTORMSG with PENDING status is reprocessed using REPROCESS action, Update Manager Engine listens to this event and sends the message to the Data Persistor via the Message Bus to be processed as explained above.

Update Manager Engine

Update Manager is a component that listens to real time events from Calypso and sends messaging to

- Head room check when needed to check limits and collateral
- Outgoing Feed to send Acknowledgement to CCP
- Reprocessing of Failed Messages

Update Manager is only interested in the BO Messages of the following message types.

- INCOMINGFEEDMSG
- OUTGOINGFEEDMSG
- UPLOADEREXCEPTIONMSG
- DATAPERSISTORMSG

Whenever a REPROCESS/RESEND message event for the above mentioned message types are generated, Update Manager will listen to that event and send to the appropriate component based on the message type.

10.1 Clearing Member Solution with Head Room Check

Update Manager Engine listens to the following events:

10.1.1 Trade

Though the trade workflow is configurable with different states, in order for the Clearing Member Solution to work properly, the Update Manager listens to the trades with following statuses.

- PENDING_HRC (meaning that the trade is PENDING from Head Room Check to be either accepted or rejected)
- PENDING_HRC_REJECTION (meaning that the trade is Force Rejected by user and the OutgoingFeed component should send a rejection message to the CCP)
- PENDING_HRC_REVERT (meaning that the trade is Refused by the other party and the trade is waiting on the Head Room Check to make sure that if there is limit already allocated, that should be reverted)
- PENDING_HRC_TERMINATE (meaning that a TERMINATE is to be applied on the trade, and the trade is pending with Head Room Check for the allocated limit to be reverted)
- PENDING_HRC_AMEND (meaning that an AMEND is received and the trade is pending with Head Room Check to update the reserved limit for this trade)

Whenever a trade event with the above-mentioned status is generated, Update Manager listens to that event and sends the event to the Head Room Check component via the Message Bus, to check for Limits & Collateral. The exception to this rule is if the action applied is UPDATE. In which case Update Manager will ignore that event.

10.1.2 Collateral / Limit Update

Collateral and Limit Update events are created when the user update the collateral and limit objects in the system, in which case the Update Manager sends these events to the necessary components via the Message Bus.

10.2 Clearing Member Solution without Head Room Check

In case where the Head Room Check is not being used and the user is using some other system to apply limit and collateral checks, the following 2 statuses are very important.

- PENDING_ACCEPTANCE (to send consent granted message to CCP)
- PENDING_REJECTION (to send consent refused message to CCP)

Whenever a trade event with the above-mentioned status is generated, Update Manager will listen that event and send it to outgoing for sending acknowledgement. The exception to this rule is if the action applied is UPDATE. In which case Update Manager will ignore that event

10.3 Engine Configuration

The UpdateManagerEngine is configured in the Engine Manager of Web Admin.

For Clearing Member solution with Head Room Check, please add the following Events.

PSEventHeadroomCheckCollateralUpdate

PSEventHeadroomCheckLimitUpdate

The following configuration files should be deployed to the engine server after being customized as specified in previous sections:

- datauploaderQueueNames.properties
- <Service>Service.properties
- gatewayservice.properties

The Update Manager Engine is run as part of the Engine server.

10.4 Exception Handling in Update Manager Engine

As explained above the job of the Update Manager Engine is to listen to real time events and send those messages to the designated Queue on the Message Bus, so that the respective consumer component processes the message.

If for whatever reason the engine cannot send the message to the bus (e.g. technical failure or the Bus is down), then depending upon the type of event the Engine would handle it as described in the sections below.

10.4.1 Trades

In the case of a submit failure for a trade, the trade will be moved to a status SUBMIT_FAILED. This scenario can arise for all those statuses for which the Update Manager listens.

In Clearing Member Solution with Head Room Check these statuses are:

- PENDING_HRC
- PENDING_HRC_REJECTION
- PENDING_HRC_REVERT
- PENDING_HRC_TERMINATE
- PENDING_HRC_AMEND

In Clearing Member Solution without Head Room Check these statuses are:

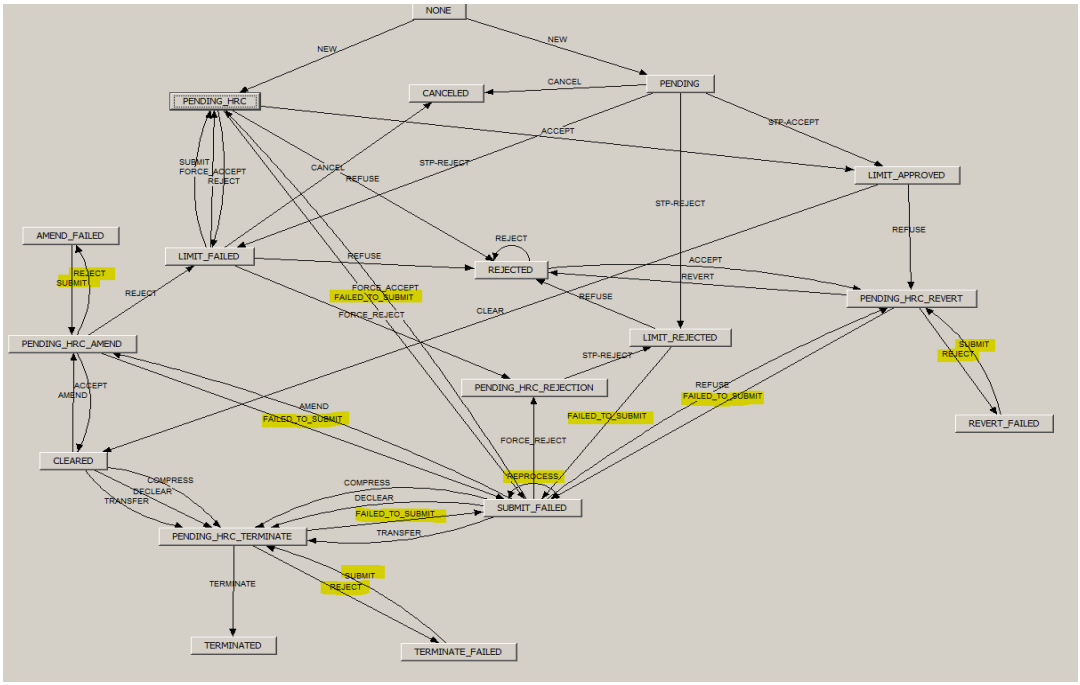
- PENDING_ACCEPTANCE
- PENDING_REJECTION

There is a transition from each of the above statuses to a SUBMIT_FAILED status, and Update Manager moves the trade to SUBMIT_FAILED status by applying an action FAILED_TO_SUBMIT. When doing that, Update Manager will also add a keyword to the trade to indicate the Submit action applied in the keyword with name 'SubmitAction', so that the user does not have to know from which state the trade has moved to SUBMIT_FAILED; regardless of how the trade came to SUBMIT_FAILED, the user just has to apply REPROCESS action on the trade for the trade to be submitted and the Update Manager will know exactly which status to move the trade back to, by applying the action in the keyword 'SubmitAction' resulting in the trade moving to one of the statuses mentioned above. This will trigger Update Manger to send the message again to the appropriate component, and this cycle repeats until it is successful. By moving the trade to a separate SUBMIT_FAILED status, it will avoid a scenario where a user tries to submit the same trade multiple times.

Another scenario is where the trade is submitted successfully to the Head Room Check component, and it rejects the trade due to some technical reason, but from a business perspective it cannot reject the trade. For example, the Head Room Check component rejects a trade received in a Clearing Confirmed auto-consent message, which by definition must be accepted since it has been auto-consented.

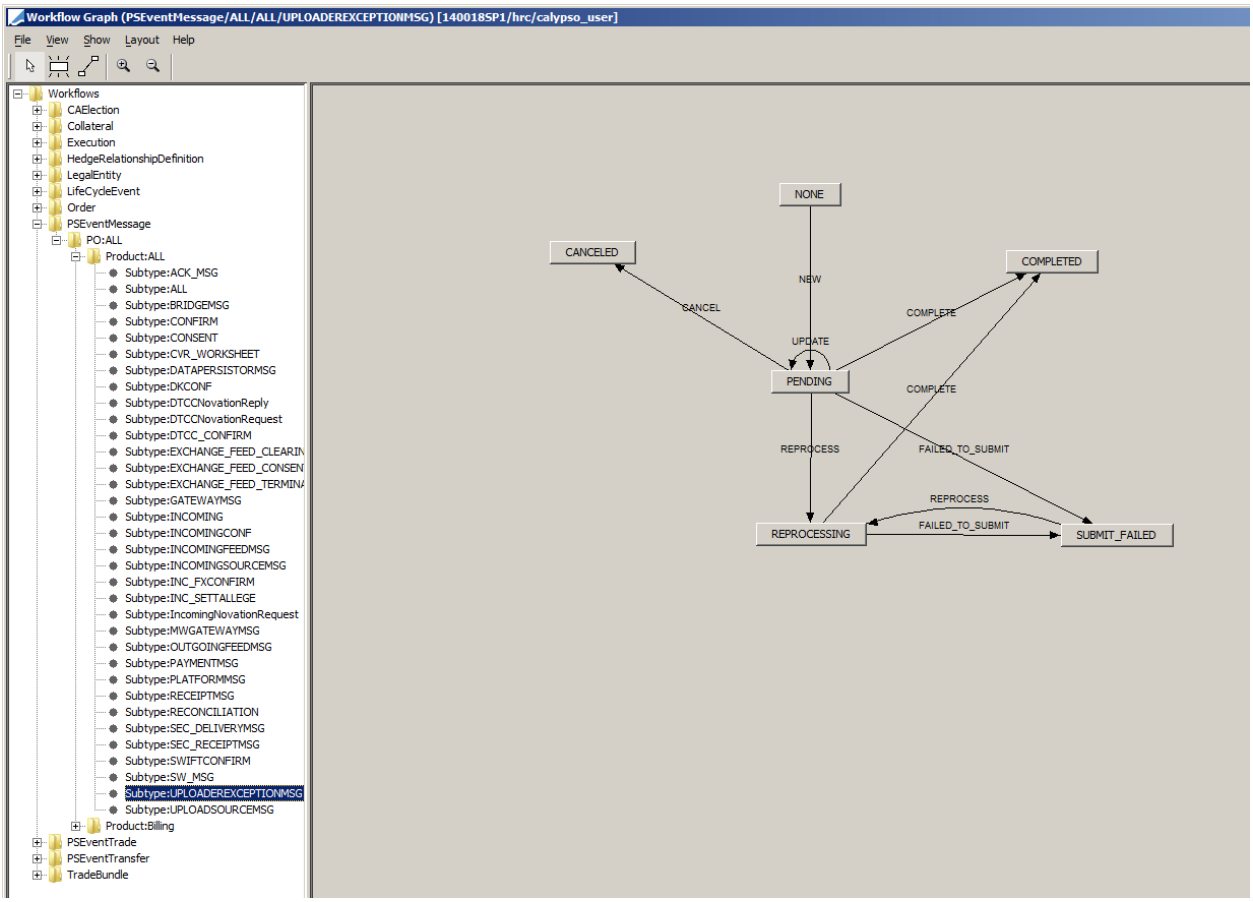
In these cases, the Data Persistor would move the trade to another failed status such as REVERT_FAILED, TERMINATE_FALIED, and AMEND_FAILED by applying the REJECT action, if the result is successful a respective business accept action is applied to move the trade to the appropriate status. By doing so the user will be able to see clearly from the trade status that a REVERT/AMEND/TERMINATE has been submitted to HRC but HRC rejected with some reason which can be found in the trade keyword 'HRCRejectReason', and then resubmit the trade to HRC by applying SUBMIT action.

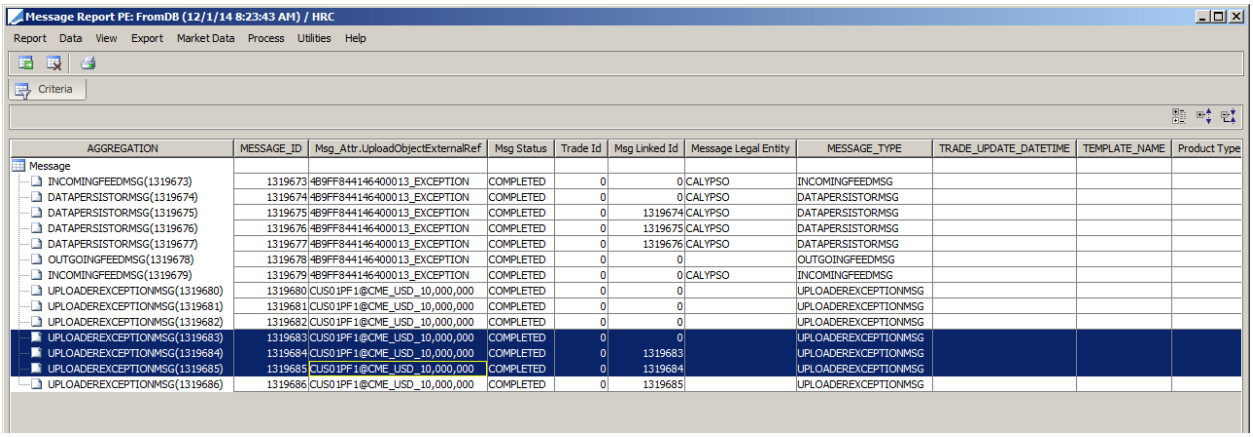
The following workflow diagram depicts these scenarios, and the transitions are highlighted for reference.


NONE
NEW
NEW
PENDING_HRC
CANCELED
CANCEL
PENDING
ACCEPT
STP-ACCEPT
LIMIT_APPROVED
SUBMIT
FORCE_ACCEPT
REJECT
CANCEL
REFUSE
STP-REJECT
STP-REJECT
REFUSE
AMEND_FAILED
REJECT
SUBMIT
LIMIT_FAILED
REFUSE
REJECTED
REJECT
ACCEPT
REVERT
PENDING_HRC_REVERT
REJECT
FORCE_ACCEPT
FAILED_TO_SUBMIT
FORCE_REJECT
CLEAR
REFUSE
LIMIT_REJECTED
PENDING_HRC_AMEND
STP-REJECT
SUBMIT
REJECT
ACCEPT
AMEND
AMEND
FAILED_TO_SUBMIT
PENDING_HRC_REJECTION
REFUSE
FAILED_TO_SUBMIT
FAILED_TO_SUBMIT
REVERT_FAILED
FORCE_REJECT
CLEARED
COMPRESS
DECLEAR
TRANSFER
REPROCESS
COMPRESS
DECLEAR
SUBMIT_FAILED
FAILED_TO_SUBMIT
TRANSFER
PENDING_HRC_TERMINATE
TERMINATE
SUBMIT
REJECT
TERMINATED
TERMINATE_FAILED

10.4.2 Collateral Update & Limit Update

Collateral and Limit Update objects do not have any workflow like trades, and hence when the Update Manager fails to send these objects to Head Room Check component, Update Manager would create a BO Message of type UPLOADEREXCEPTIONMSG with its own workflow and persist the event as the payload of the advice document attached to the message. In this case, the user should REPROCESS the message, so that it can be resubmitted to the Update Manager, to resend it to the Head Room Check component.



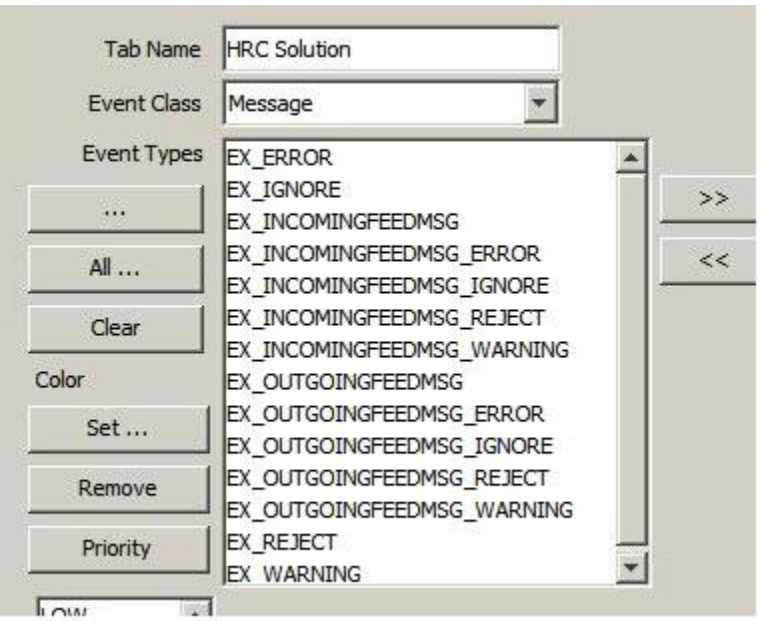


AGGREGATION	MESSAGE_ID	Msg_Attr.UploadObjectExternalRef	Msg Status	Trade Id	Msg Linked Id	Message Legal Entity	MESSAGE_TYPE	TRADE_UPDATE_DATETIME	TEMPLATE_NAME	Product Type
Message										
INCOMINGFEEDMSG(1319673)	1319673	4B9FF844146400013_EXCEPTION	COMPLETED	0	0	CALYPSO	INCOMINGFEEDMSG			
DATAPERSISTORMSG(1319674)	1319674	4B9FF844146400013_EXCEPTION	COMPLETED	0	0	CALYPSO	DATAPERSISTORMSG			
DATAPERSISTORMSG(1319675)	1319675	4B9FF844146400013_EXCEPTION	COMPLETED	0	1319674	CALYPSO	DATAPERSISTORMSG			
DATAPERSISTORMSG(1319676)	1319676	4B9FF844146400013_EXCEPTION	COMPLETED	0	1319675	CALYPSO	DATAPERSISTORMSG			
DATAPERSISTORMSG(1319677)	1319677	4B9FF844146400013_EXCEPTION	COMPLETED	0	1319676	CALYPSO	DATAPERSISTORMSG			
OUTGOINGFEEDMSG(1319678)	1319678	4B9FF844146400013_EXCEPTION	COMPLETED	0	0		OUTGOINGFEEDMSG			
INCOMINGFEEDMSG(1319679)	1319679	4B9FF844146400013_EXCEPTION	COMPLETED	0	0	CALYPSO	INCOMINGFEEDMSG			
UPLOADEREXCEPTIONMSG(1319680)	1319680	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	0		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319681)	1319681	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	0		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319682)	1319682	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	0		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319683)	1319683	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	0		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319684)	1319684	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	1319683		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319685)	1319685	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	1319684		UPLOADEREXCEPTIONMSG			
UPLOADEREXCEPTIONMSG(1319686)	1319686	CUS01PF1@CME_USD_10,000,000	COMPLETED	0	1319685		UPLOADEREXCEPTIONMSG			

Task Station

The following tab in Task Station should be added to monitor the Trade (Update Manager), INCOMINGFEEDMSG, OUTGOINGFEEDMSG objects.

Please add the same for the message types UPLOADEREXCEPTIONMSG and DATAPERSISTORMSG.



For Trades that are FAILED_TO_SUBMIT: EX_ERROR, SUBMIT_FAILED_TRADE.

Troubleshooting

This section contains details on how to troubleshoot any issues you may encounter with the messaging solution components.

Debug Logging

The following Log categories should be used across all the components of the messaging solution

UPLOADER (for general uploader framework logging)

UPLOADER_STATS (logs the time taken for things like translation and persistence)

MESSAGE_TRACE (Used to trace a particular message across the messaging components to trouble shoot which components a particular message has reached and if a message failed to process this log will tell you which component it failed)

Example:

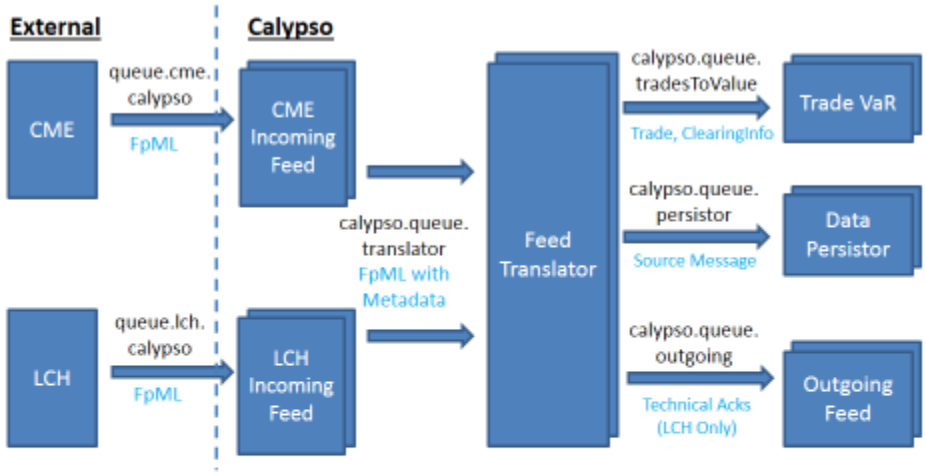
```
Start processing... | TraceId : 141807959863102 | MessageType : RequestConsent | JMSXGroupID : 85433 | From
Endpoint :upload://calypso.queue.persistor | From Route: rDataPersistor | current route : rDataPersistor |
current endPoint: upload://calypso.queue.persistor | exchangeId : ID-vhunglt-54469-1418078340854-0-8 |
exchange.getIn().getMessageId() : ID:vhunglt-54477-1418078342308-1:1:2:1:2


Finished processing... | TraceId : 141807959863102 | MessageType : RequestConsent | JMSXGroupID : 85433 | From
Endpoint :upload://calypso.queue.persistor | From Route: rDataPersistor | current route : rDataPersistor |
current endPoint: upload://calypso.queue.persistor | exchangeId : ID-vhunglt-54498-1418078346242-0-4 |
exchange.getIn().getMessageId() : ID:vhunglt-54477-1418078342308-1:1:2:1:2
```

The TraceId is unique for that message across all the components.

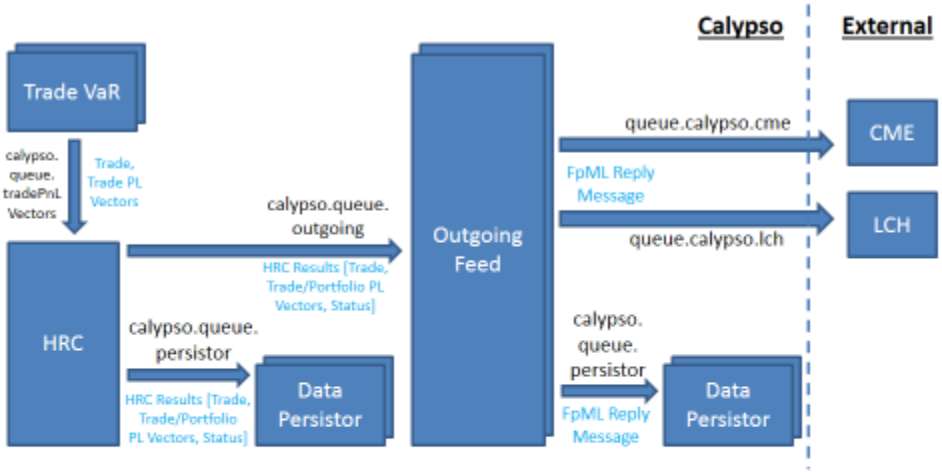
Appendix

13.1 FCMHRC Flow

The following 2 slides show the end-to-end flow for the FCMHRC solution.



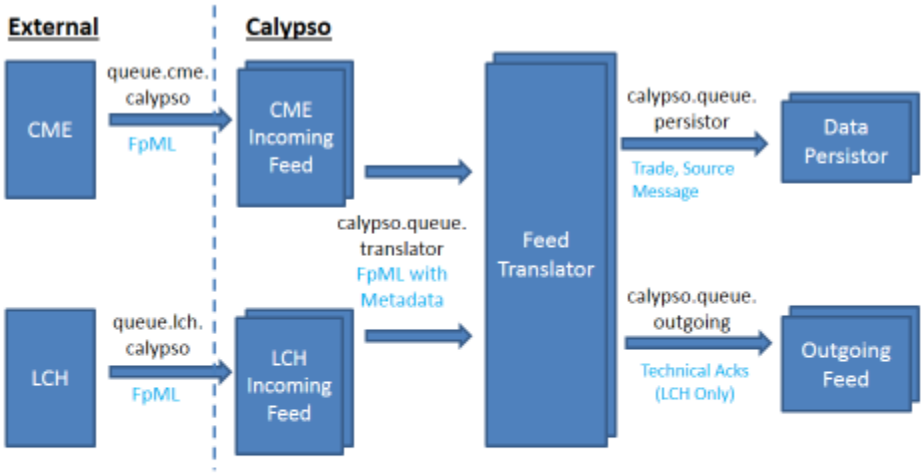
1. Trades arrive from CME/LCH into Websphere MQ instance.
2. The initial Incoming Feed entry point, per CCP, ensures proper message sequencing. Multiple instances can be run.
3. Trades are then sent to a shared Feed Translator; multiple instance of this can be run as well. In addition, instances can be configured to be dedicated to a single source type, e.g. CME only.
4. Feed Translator translates the FpML to a Calypso Trade, and hands off to Trade VaR. In addition, it will send a copy of the incoming message to the Data Persistor for audit purposes, as well as send technical acks to Outgoing Feed if required, e.g. LCH.



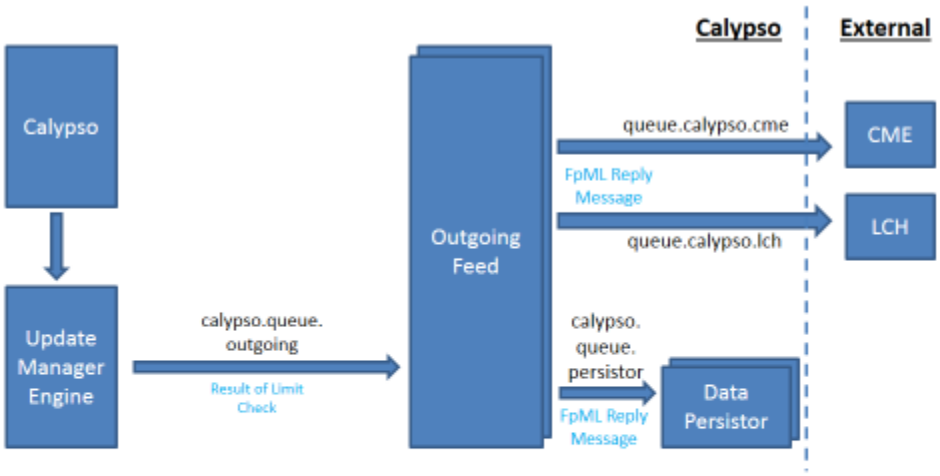
1. The Trade VaR process will calculate the Trade PL Vectors and hand off to the HRC process to perform the limit check. Once complete, HRC will hand off the result to Outgoing Feed. In addition, HRC will send the Trade and Trade/Portfolio PL Vectors to the Data Persistor for saving in the DB.
2. Outgoing Feed will generate an appropriate reply message to the CCP and send it to the correct outgoing queue for that CCP. It will also send a copy of the outgoing message to the Data Persistor for audit purposes.

13.2 FCM Flow

The following 2 slides show the end-to-end flow for the FCM solution



1. Trades arrive from CME/LCH into Websphere MQ instance.
2. The initial Incoming Feed entry point, per CCP, ensures proper message sequencing. Multiple instances can be run.
3. Trades are then sent to a shared Feed Translator; multiple instance of this can be run as well. In addition, instances can be configured to be dedicated to a single source type, e.g. CME only.
4. Feed Translator translates the FpML to a Calypso Trade, and hands it off to Data Persistor for saving into Calypso. In addition, it will send a copy of the incoming message to the Data Persistor for audit purposes, as well as send technical acks to Outgoing Feed if required, e.g. LCH.



1. Once the trade is in Calypso, standard trade workflow can be used to implement Limit Checking functionality. Once complete, specific Trade Statuses can be used to trigger the Update Manager Engine, which will send a message to Outgoing Feed.
2. Outgoing Feed will generate an appropriate reply message to the CCP and send it to the correct outgoing queue for that CCP. It will also send a copy of the outgoing message to the Data Persistor for audit purposes.