Exercising a Resilient Java CAPS 6 HL7 v2 Repository Solution

Michael@Czapski.id.au
April 2010, Release 1.0.0.0

Table of Contents

Introduction

From time to time prospective clients ask for a proof that Java CAPS will not loose HL7 messages in the event of machine or network failure.

In this Note a heterogeneous, non-clustered collection of hosts will be used to implement and exercise Java CAPS 6/Repository HL7 v2 based solutions. The environment consists of two independent “machines”, which are not a part of an Operating System Cluster. Each “machine” hosts a GlassFish Application Server, which is the Java CAPS 6 runtime. Application Servers are independent of one another and are not clustered. This is to demonstrate that HL7 processing components, and solutions based on these components and other standard components in the Java CAPS infrastructure, can be designed and implemented in such a way that message loss in the event of typical failure and disruption scenarios is avoided.

This note discusses an exercise involving an example healthcare environment processing HL7 v2 messages. Discussion includes customization of a generic Java CAPS 6.2 VMware Virtual Appliance for a specific HL7 exercise and deploying ready-made Java CAPS 6/Repository-based solutions. The exercise for HL7 eWay and HL7 Inbound and Outbound projects, processing HL7 v2.3.1 messages, will be conducted and discussed.

The reader will be convinced that a resilient Java CAPS solution can be configured and that it will process messages in the face of typical failure and disruption scenarios without message loss or duplication.

Note that this article is not introductory in nature. It assumes that the reader has a fairly good working knowledge of the Java CAPS 5 or Java CAPS 6/Repository product and a good working knowledge of related areas, such as HL7 messaging, virtualisation and suchlike. These matters are not explained in this article.

Goals

HL7 v2.3.1 messages will be sent to a HL7 Receiver hosted on a machine which is experiencing scheduled shutdowns, unscheduled crashes and network outages. The HL7 Receiver will pass the messages it receives to a downstream host. It is expected that all messages sent by the original sender will be received by the final receiver at least once, in the order in which they were sent.

1. Demonstrate configuration of a pre-built HL7 Outbound project to support retries
2. Demonstrate configuration of pre-built HL7 Inbound project to support retries
3. Demonstrate HL7 Inbound project transactionality
4. Demonstrate JMS tranactionality
5. Demonstrate behaviour of the solution when the VM Appliance crashes and is restarted
6. Demonstrate behaviour of the solution when the VM Appliance is shut down from the operating system level and is restarted
7. Demonstrate behaviour of the solution when the Application Server in the VM Appliance is shut down and re-started
8. Demonstrate behaviour of the solution when the Network Interface Card is forcibly disconnected and re-connected
9. Demonstrate end-to-end lossless message processing

Assumptions

Familiarity with Java CAPS 6.2/Repository development and the use of pre-built HL7 projects is assumed.

It is assumed that pre-built logic in the HL7 Inbound and HL7 Outbound projects is appropriate for the exercise. No modification of this logic will be made.

It is assumed that a single-node Java MQ JMS implementation will suffice for in-flight message persistence for a single-node processing environment to demonstrate HL7 projects resilience capabilities.

JMS redelivery handling will be used to attempt to redeliver a message to outbound the HL7 eWay and the HL7 eWay redelivery handling will be used to redeliver to the remote receiver.

It is assumed that demonstration of HL7 message transformation is not required to prove that no message loss occurs.

Exercise Environment

The runtime environment for the exercise will consist of a VMware Virtual Machine with Java CAPS 6.2 runtime installation, and the VMware Host machine with a full

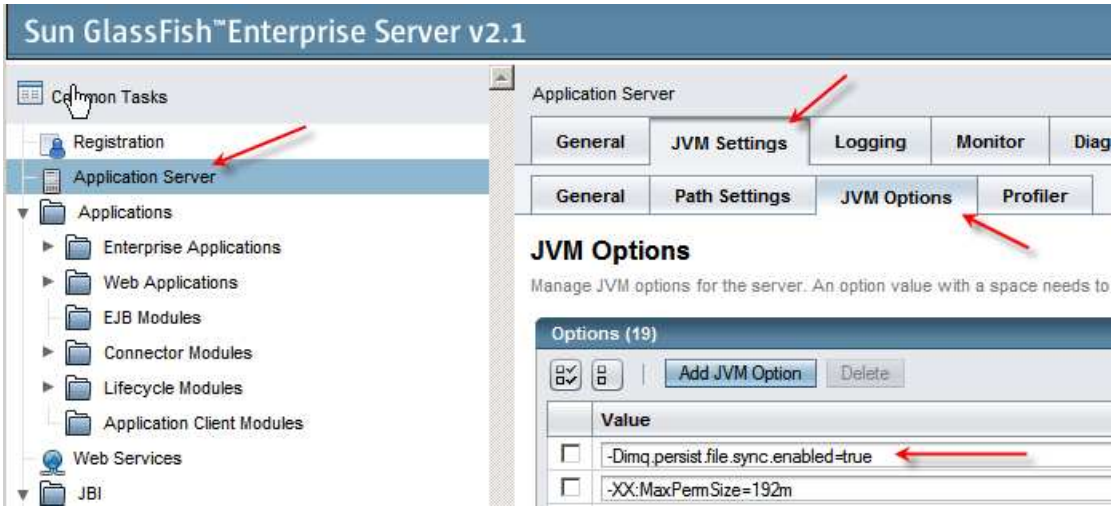
Java CAPS 6.2 development installation, in addition to its regular duties as the VMware host. Construction if this Virtual machine is discussed in the blog article "Installing Java CAPS 6.2 Runtime on the Basic JeOS Appliance for HL7 Resilience Testing" at http://blogs.czapski.id.au/?p=563 .

Let's customize the virtual machine.
The VM will run a part of the solution. It will be the solution, and the host, which are subjected to failures, network interruptions and orderly shutdowns. The other part of the solution will run on a different host, in this case the one which hosts the VM.

We need to allow the VM to resolve the name of the host partner host. Let's modify the /etc/hosts file on jc6202 and add the FQDN and aliases of the partner. For me this is mcz02.aus.sun.com, alias mcz02, alias mcz02.home, with the physical address of 192.168.47.1.

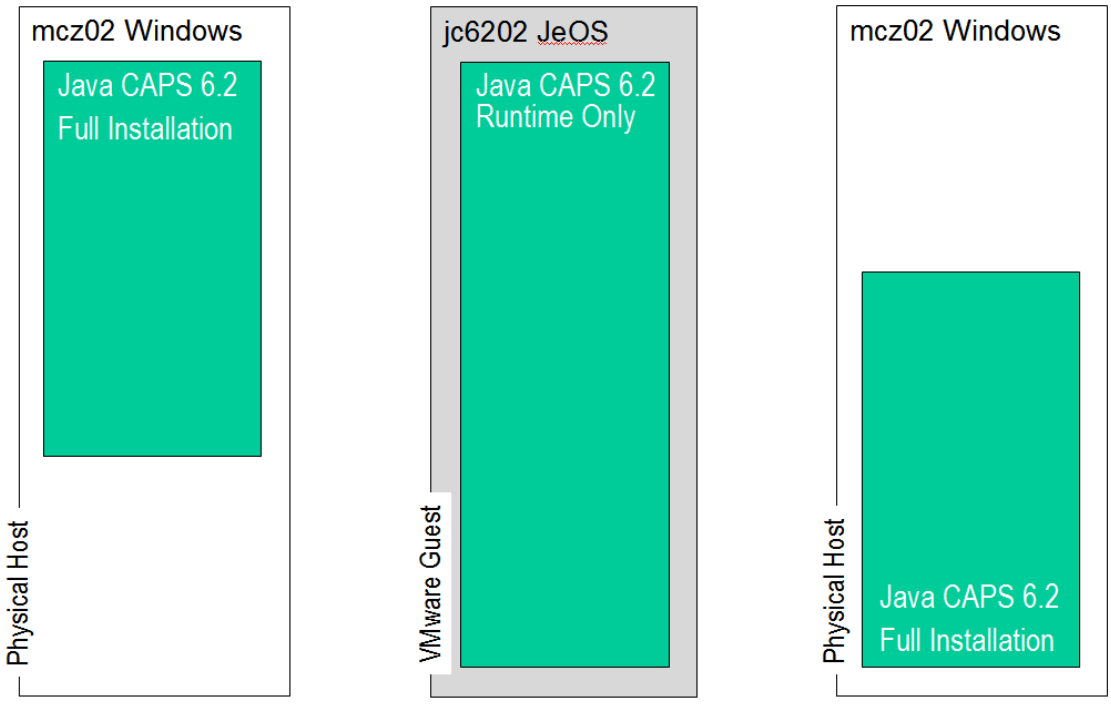
```
#
# ident "%Z%%M% %I%     %E% SMI"
#
# Internet host table
#
::1 osol-jeos osol-jeos.local localhost loghost
127.0.0.1 jc6202.local localhost loghost
192.168.47.129  jc6202 jc6202.home jc6202.aus.sun.com
192.147.47.1    mcz02.aus.sun.com mcz02 mcz02.home
```

In a default configuration the Sun Java System Message Queue (Java MQ), used in the exercise, does not synchronize its message store to disk. This means that the Java MQ keeps a certain number of message in memory as it processes them and in a crash scenario message loss can occur. To avoid potential message loss one must enable synchronization using the Java MQ **imq.persist.file.sync.enabled** property (see http://docs.sun.com/app/docs/doc/820-4916/gheap?a=view for an overview). Enabling synchronization has negative performance implications so this is a trade-off between reliability and performance. Add "-Dimq.persist.file.sync.enabled=true" to the jc6202's GlassFish Application Server JVM options using the GlassFisk Application Server Administration Console.



The schematic below depicts the "physical" environment.

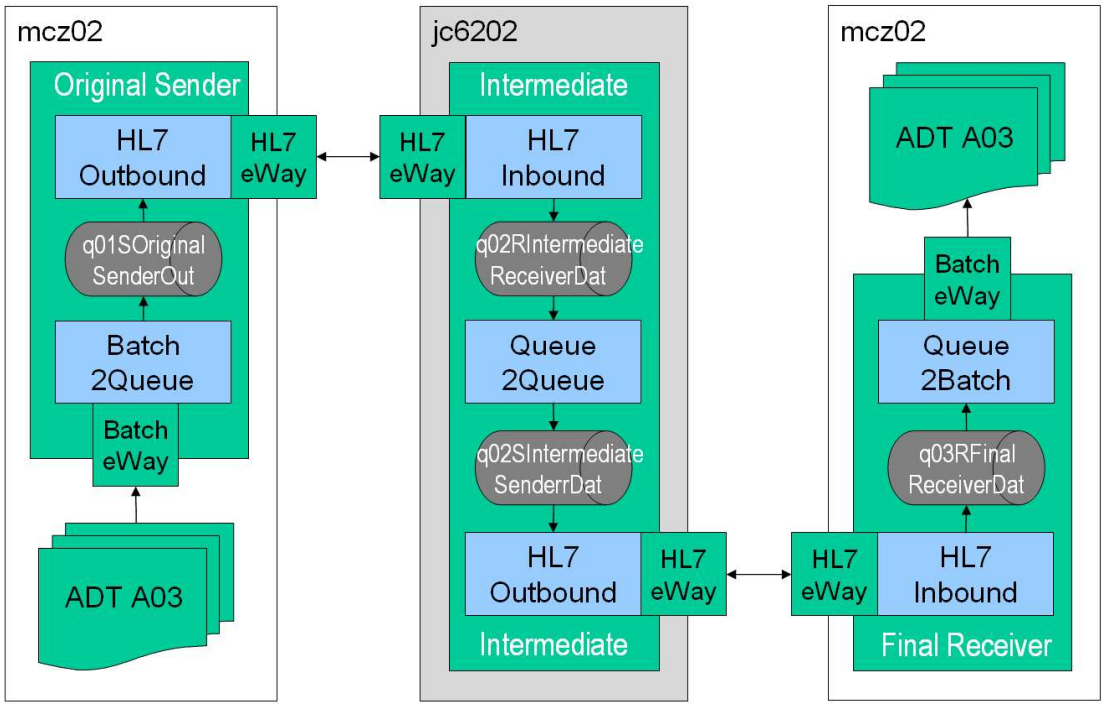
To facilitate discussion of the solution, the host environment will be shown as though mcz02 consisted of two separate hosts, which it is not.



This is to convey the separation of the original sender from the final receiver and make diagrams more readable.

Solution Implementation

The solution is designed to exercise a heterogeneous, non-clustered, failure-proof configuration for HL7-based messaging implementations.



There are three project sets: an Original Sender, an Intermediate and a Final Receiver.

The original sender reads multiple records from a file in a file system and sends each record as a separate message to the intermediary receiver, using the HL7 over TCP transport. It then waits for an accept acknowledgement – a HL7 standard response message, send back by the intermediate receiver.

The intermediate receiver receives a message, sends the message to a JMS server, generates an accept acknowledgement and sends it back to the original sender.

There is an intermediate forwarder which reads the message form the queue to which the intermediate receiver sent it, logs selected HL7 MSH fields to the console and sends the message to another JMS queue.

There is an intermediate sender, which receives a message from the JMS queue into which it was deposited by the intermediate forwarder, sends it using the HL7 over TCP transport to the final receiver and waits for the accept acknowledgment.

The final receiver receives a message from the HL7 intermediate sender and writes it to a file with the unique name containing the message ID and a date/time stamp.

All components of the solution process HL7 v2.3.1 ADT A03 messages, sent from senders to the receivers, and HL7 v2.3.1 ACK messages, retuned by receivers to senders as accept acknowledgments.

A typical A03 message might look like:

```
1 MSH|^~\&|SystemA|HosA|PI|MDM|20080910112956||ADT^A03|000000_CTLID_20080910112956|P|2.3.1|||AL|NE
2 EVN|A03|20080910112956|||JavaCAPS6^^^^^^^USERS
3 PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7 South 3rd Circle^^Downham Market^Eng
4 PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^MAIN|||EMR|||||||||V20080908801529^^^^VISIT|||||||||||||||||DISH
. loc|||||||20080908801529|20080910112956
5
```

Highlighted is the Message Control ID field, which embeds a message sequence number.

A typical acknowledgement message might look like:

```
1 MSH|^~\&|PI|MDM|SystemA|HosA|20080910112956||ADT^ACK|000000_CTLID_20080910112956|P|2.3.1|||AL|NE
2 MSA|CA|000000_CTLID_20080910112956_02A___-on-mcz02|Host mcz02 accepted and forwarded the message||D
3
```

Highlighted is the part of the Message Control ID of the message to which this is an acknowledgment.

Discussion of the HL7 version 2.3.1 messaging standard is well beyond the scope of this article. Please see www.hl7.org for material on the topic.

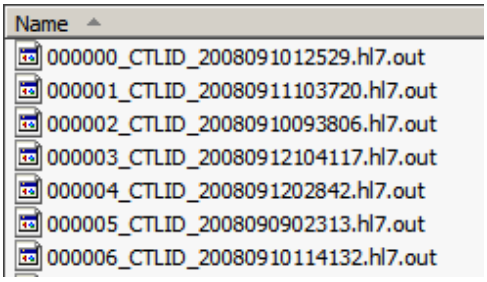
The Message ID, which is critical in this implementation to recognition of gaps in message sequence and out-of-order message delivery, is embedded in each message. Since each message is either a HL7 v2.3.1 ADT A03 message or a HL7 v2.3.1 ACK message, MSH-10, Message Control ID filed in the A03 and MSA-2 Message Control ID in the ACK are used to carry a unique Message ID. The message id looks like that shown below:

```
000000_CTLID_20080910112956
```

The first 6 digits of the message id are the serial number, which is unique, and contiguously increasing in each message. Message 1 will be 000000, message 2 will be 000001 and so on.

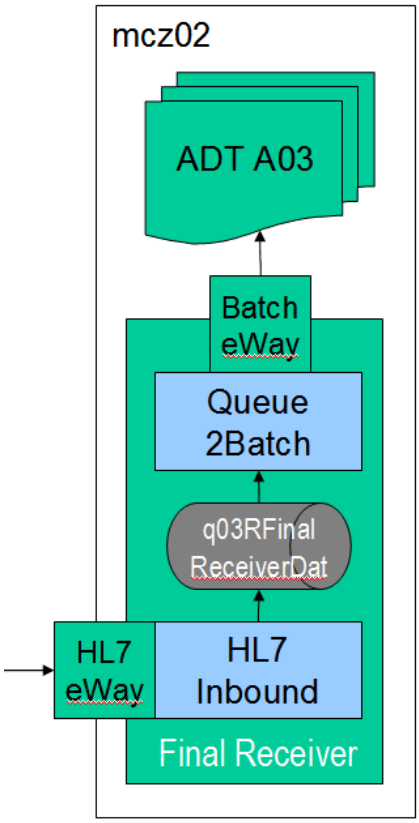
Names of files written by the final receiver, containing application messages, will start with the message id. Any breaks in sequence will be readily apparent to a human by inspection of file names in the destination directory.

File names will look like these shown below. Note the sequence numbers and timestamps embedded in file names.



The messaging stream is serialized at the sender by the HL7 eWay in such a way that the sender will not send a new message until it received an acknowledgment for the previous message.

It is critical to ensure that the sender recognizes that the receiver crashed and to retry.



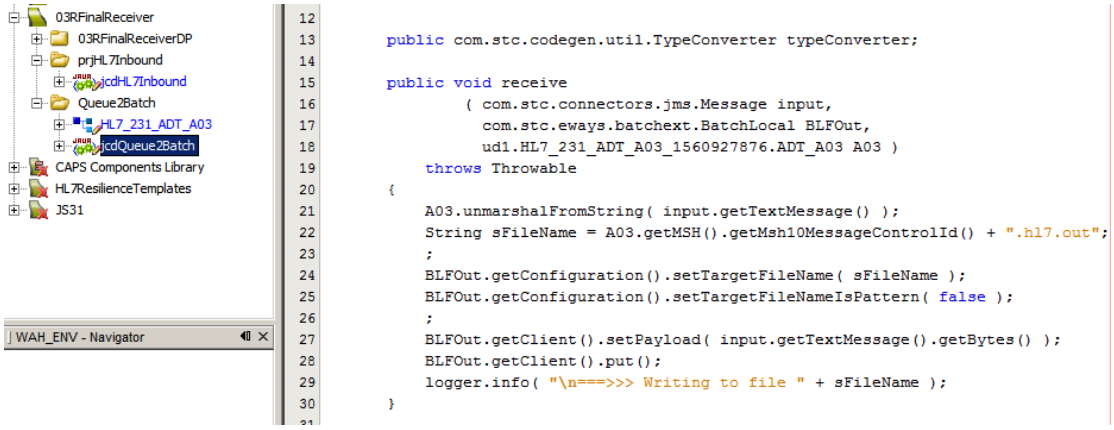
The retry must be timely. The time taken to wait for a response before concluding that it is not going to come must not be so long that it unduly slows down message processing, and must not be so short that a longer then normal time to produce a response causes the sender to re-send the request when no receiver failure actually occurred.

Overview of Final Receiver Projects

The Final Receiver project (an instance of the pre-built prjHL7Inbound) receives messages from a HL7 eWay and deposits them in a JMS Queue. A Queue2Batch project receives these messages from JMS and writes them into a specific file system directory, one file per message.

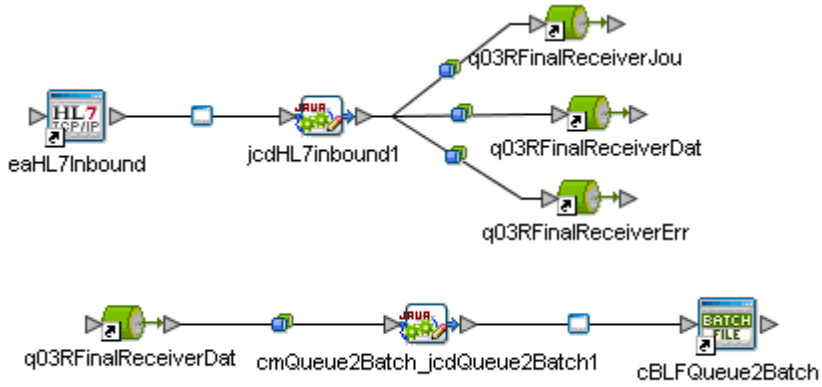
The HL7 Inbound is an instance of the standard, pre-built prjHL7Inbound.

The Queue2Batch is based on a simple Java Collaboration.



The collaboration constructs a file name, based on the message control id embedded in the HL7 ADT A03 message, and writes the entire JMS payload to a file with that file name.

The connectivity Maps are shown below.



As delivered, the HL7 Inbound pre-built component has a coding bug which prevents it from correctly recognising HL7 accept acknowledgements (type C). The jcdHL7Inbound collaboration in the prjHL7Inbound must be modified to correct this bug.
Open the collaboration, locate the line that reads:

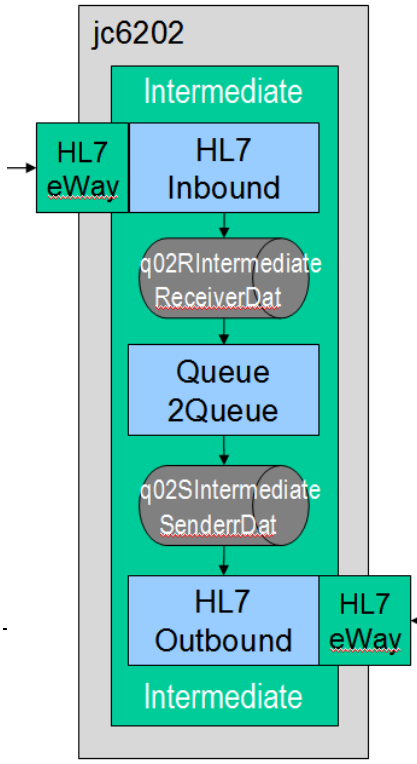
"`if (input.getHL7MessageInfo().getHL7AcknowledgmentInfo().`**`getAcknowledgmentLevel() == "C")`** {"

and change it to read:

"`if (input.getHL7MessageInfo().getHL7AcknowledgmentInfo().`**`getAcknowledgmentLevel().equalsIgnoreCase( "C" ))`** {"

The developer clearly confused languages and data types.

Save the collaboration and re-build any projects in which it is used. From now on the accept acknowledgement setting will be correctly recognised.
All project exports are available for download as JC62_HL7_Resilience_Project_Exports_with_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_with_Envs.zip and JC62_HL7_Resilience_Project_Exports_no_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_no_Envs.zip.
It is assumed that HL7 eWay and HL7 2.3.1 OTD Libraries are installed.



Overview of Intermediate Projects

The intermediate receiver, an instance of the pre-built prjHL7Inbound, receives messages from the HL7 eWay, sends the HL7 payload to the JMS

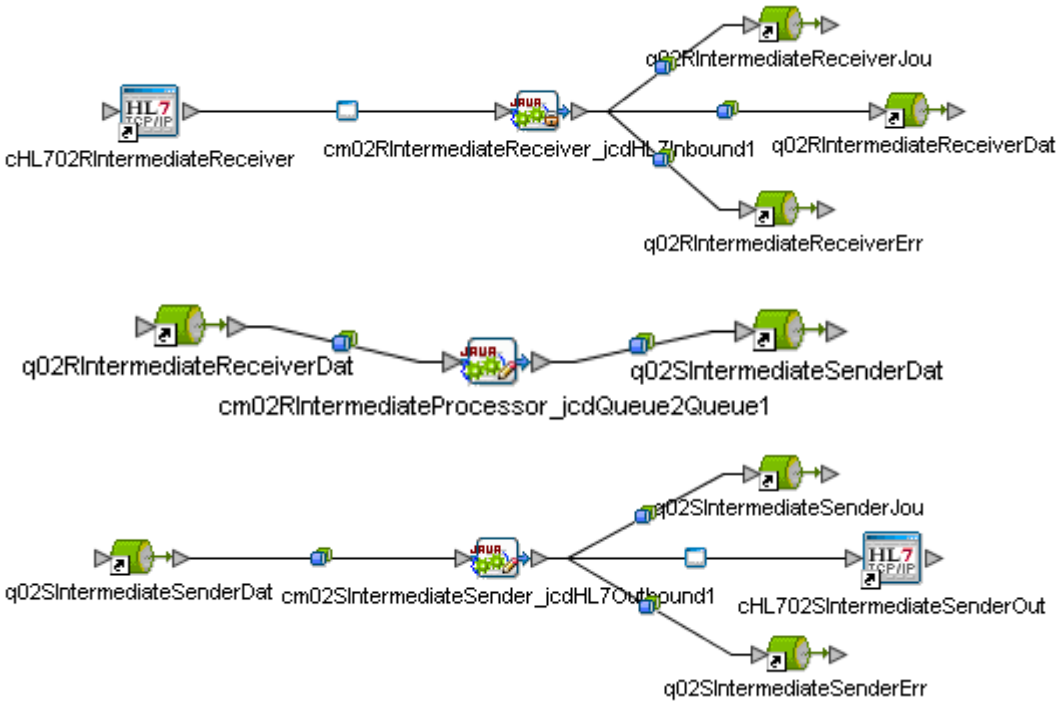
Server, constructs a HL7 ACK message and sends the ACK as a response to the sender.

The HL7 message is picked up by the Queue2Queue intermediary project which parse HL7 MSH segment, logs the MSH-10 Message Control ID to the server.log and sends the whole HL7 message to another JSM queue.

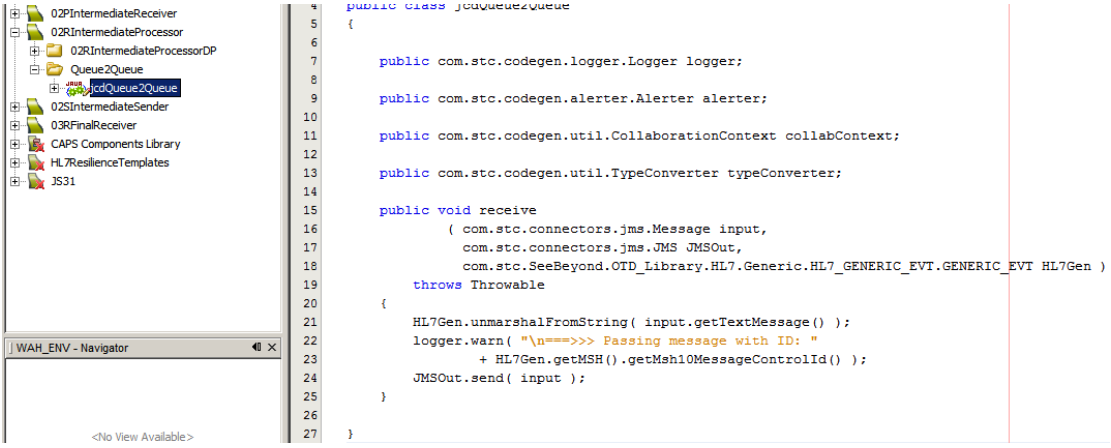
The HL7 message is picked up by the sender intermediary, which is an instance of a prjHL7Outbound pre-built project, sends it to a HL7 external and waits for the application ACK.

Remember to make sure that the jcdHL7Inbound collaboration is fixed to correctly recognise accept acknowledgment settings in the HL7 eWay configuration, as discussed in section “Overview of Final Receiver Projects”.

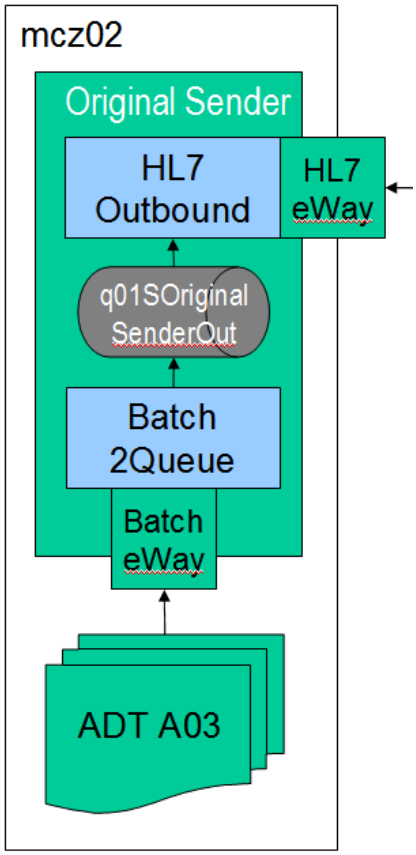
The connectivity maps are shown below.



The intermediary processor is a simple Java collaboration, there merely to log the MSH-10 Message Control ID. The intermediary receiver could directly send HL7 messages to the queue to which the intermediary sender listens instead of adding another component to copy messages between queues.



All project exports are available for download as JC62_HL7_Resilience_Project_Exports_with_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_with_Envs.zip and JC62_HL7_Resilience_Project_Exports_no_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_no_Envs.zip.



Overview of Original Sender Projects

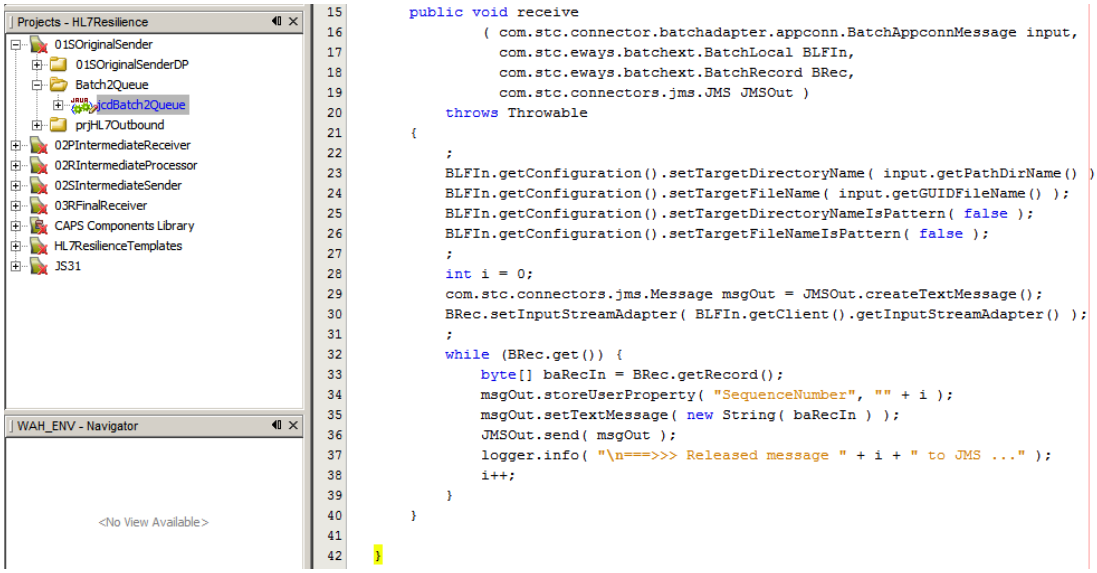
A Batch2Queue project in the Original Sender reads a file containing multiple HL7 ADT A03 records and sends each as a separate message to a JMS Queue. The HL7Outbound, which is an instance of the pre-built prjHL7Outbound, sends each message to the external using the HL7 eWay and waits for the HL7 Accept ACK.

Normally one would feed the solution we are discussing with messages from some hospital system, for example a lab system or a patient registration system. Since I don't have one of these to offer the Batch2Queue is the external system simulator. What it does to emit a series of HL7 v2 messages, and how it goes about doing it, is not really relevant to the discussion on HL7 messaging resilience. Since, however, it is a bit tricky, I will make a few comments on how it is constructed and why, to foster a better understanding of the product and its usages.

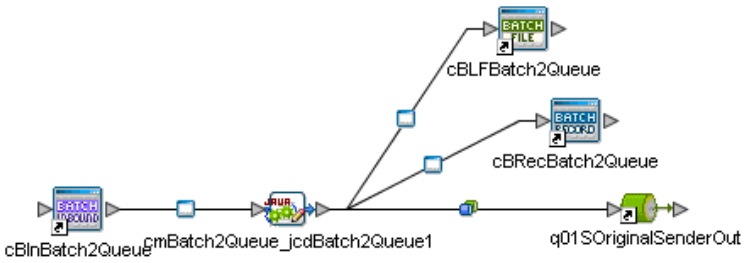
The Batch2Queue must read a file containing potentially a very large number of records (in the exercise we will use a file with 5099 records), break it up at "record boundary", and submit each record to JMS. The Java Collaboration uses the BatchInbound eWay, the BatchLocalFile eWay and the BatchRecord eWay to accomplish the task of reading and splitting the file. This is a fairly typical way to

accomplish this task in Java CAPS and has been discussed in detail elsewhere, for example in the book “Java CAPS Basics – Implementing Common EAI Patterns” (http://www.google.com.au/search?q=%22Java+CAPS+Basics%22).

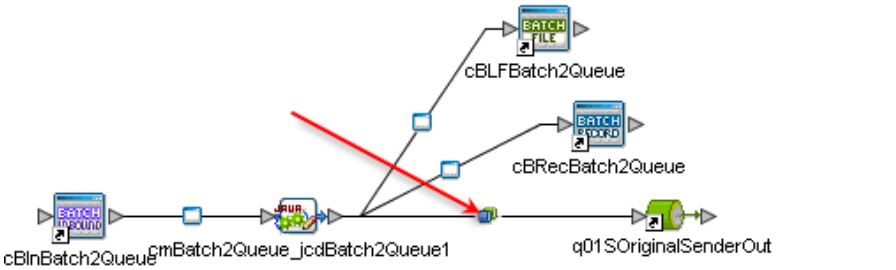
The key elements of the source of this collaboration are reproduced below.

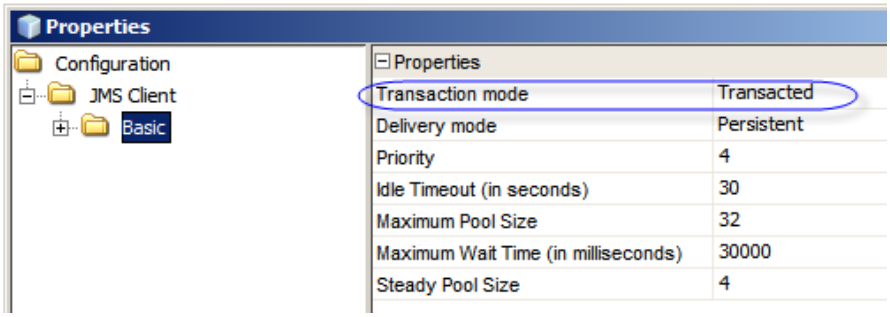


The connectivity map for this part of the original sender solution is shown below.

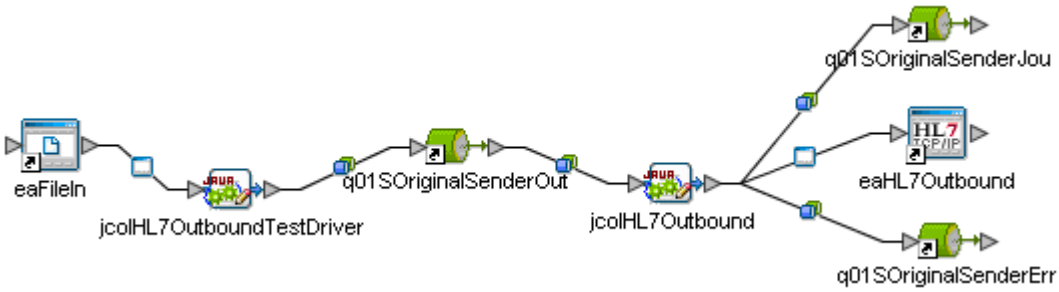


By default the JMS connector in the connectivity map is configured to use XA. This means that until the collaboration completes all records sent to the JMS queue will not be committed. With Sun Java System Message Queue (JMQ), for instance, there is a default limit on the number of buffered sends (1000 if memory serves). An exception will be thrown if one attempts to send more then that number in a single transaction. While this limit can be explicitly raised the general issue is not really addressed since there can be a number of records in a file greater that the new limit and the issue will recur. The solution is to change the JMS connector properties to Transactional, which will cause each JMS send to be committed individually. This will allow unlimited number of records to be processed by the single invocation of the collaboration at the potential cost of record duplication if the collaboration is aborted mid-processing and re-started with the same file.

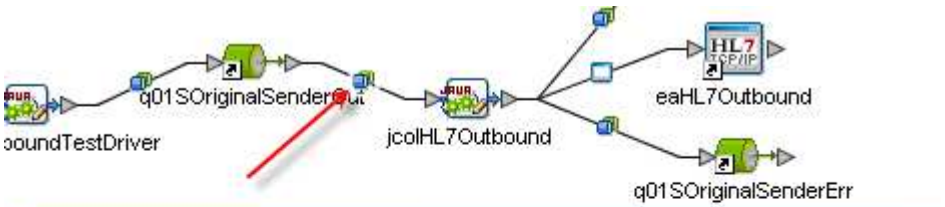


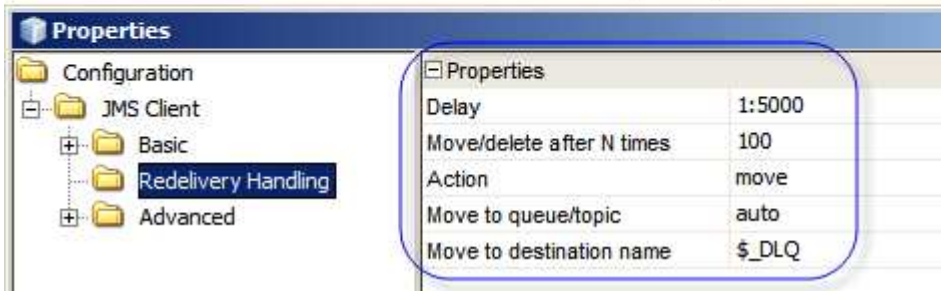


The HL7 Outbound project is an instance of the pre-built prjHL7Outbound. The connectivity map for this project is shown below.



To ensure correct behaviour in face of failures few critical configuration changes must be made to the standard connectivity map. Most notable is the requirement to configure redelivery handling for the JMS connector between the queue from which the jcdHL7Outbound collaboration receives messages and the collaboration itself.



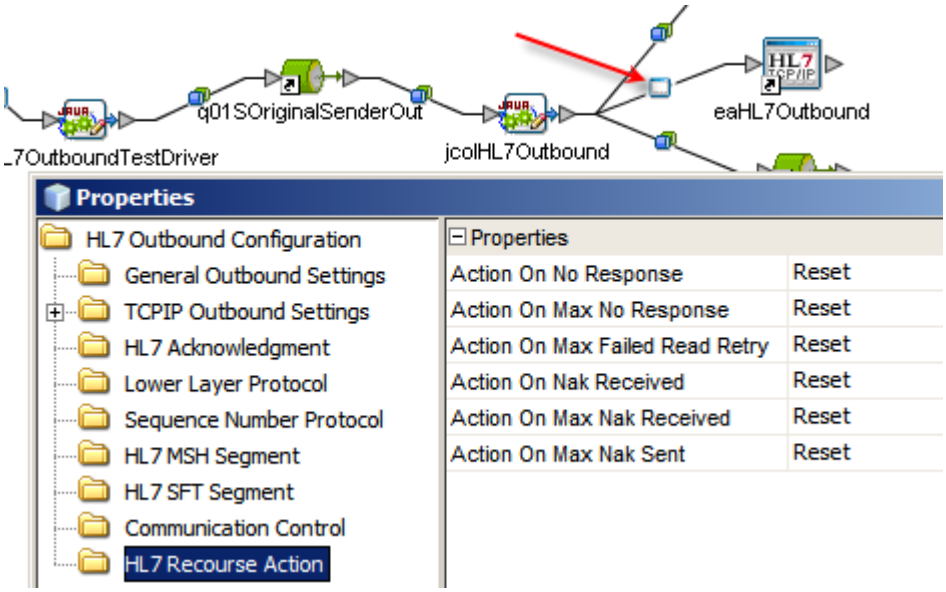


By default redelivery handling is not configured. If the downstream Java collaboration throws an exception and redelivery handling is not configured explicitly at the connector, and redelivery handling is not configured globally for the JMS instance, the message will be rolled back and re-delivered immediately. This will continue until the message is delivered or until the infrastructure is brought down. This will cause processing loop continually retrying, most likely failing and using up all machine resources while doing so, and certainly preventing further messages from being processed. Whether this is a good thing or a bad thing, therefore whether something needs to be done about it, you will have to decide for your solution.

The Java collaboration will throw an exception in any number of circumstances. When connectivity between the sending HL7 eWay and the external system to which it sends is disrupted, the HL7 eWay will enter recourse processing mode, attempting to re-connect/re-deliver as configured. When the configured number of re-connection/re-delivery attempts is exhausted without the eWay being able to re-connect/re-deliver, it will throw an exception. Default redelivery handling will cause the message to be immediately re-delivered to the collaboration, which will try to get it sent again. When the collaboration receives an invalid message it will throw an exception and never attempt to deliver it. Retrying in such a circumstance is not useful since the message is unlikely to magically become valid and processable.

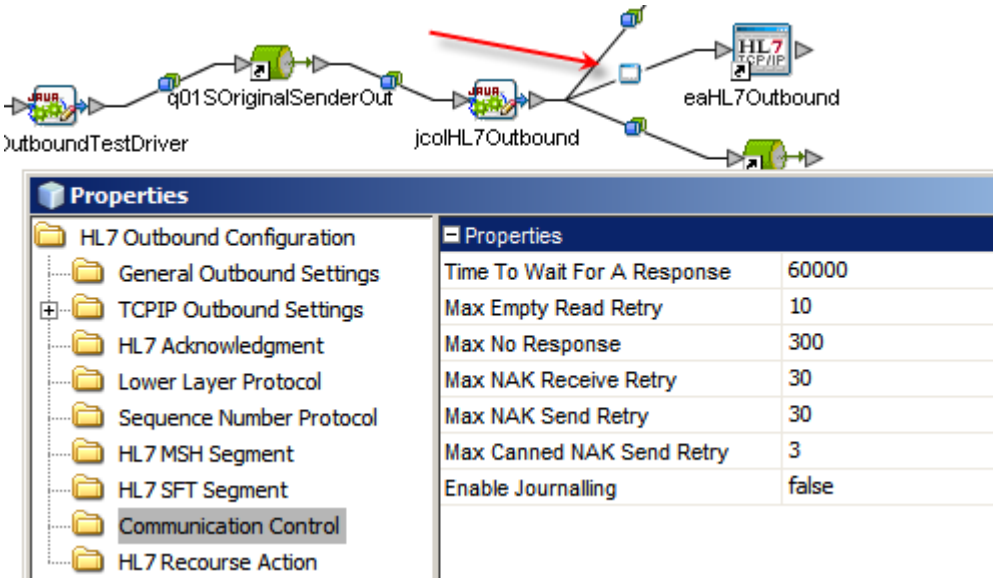
We need to introduce a **delay** of **5000** milliseconds between each retry, to slow down the process. For this solution we expect to retry up to 100 times (which adds up to about 83 minutes – more then enough for this exercise). The production solution may require a different approach, for example fewer or more redelivery attempts, or shorter wait periods. We are also setting a limit on the number of retries and having a message which can not be processed/delivered **move**d **after 100 times** to a destination named **$_DLQ**, which is the same kind of JMS destination with the original name suffixed with _DLQ.

There are other configuration properties that need to be changed. For the HL7 eWay we need to change the recourse action values to:

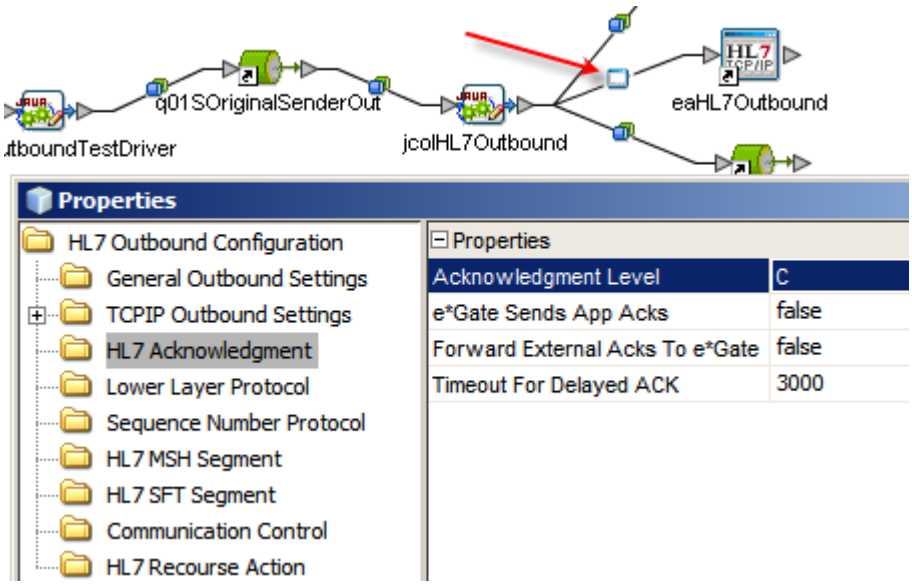


By specifying “reset” in all cases we are causing an exception to be thrown for each of the possible outcomes which in turn triggers JMS redelivery and further attempts to deliver without message loss.

One can also tweak communications parameters to increase or decrease the various parameters and affect delays and timing.



Because we are only interested in getting messages form one endpoint to another we need to set the HL7 Acknowledgment to “C” (Commit), from the default of A (Application).



This must be done consistently in all HL7 externals.

All project exports are available for download as JC62_HL7_Resilience_Project_Exports_with_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_with_Envs.zip and

JC62_HL7_Resilience_Project_Exports_no_Envs.zip at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_no_Envs.zip.

Appliance Preparation

Following instructions in Blog articles "GlassFish ESB v2.x Field Notes - Preparing Basic JeOS Appliance for GlassFish ESB LB and HA Testing", at http://blogs.czapski.id.au/?p=15 and "Installing Java CAPS 6.2 Runtime on the Basic JeOS Appliance for HL7 Resilience Testing" at http://blogs.czapski.id.au/?p=563, create an appliance whose host name is jc6202.

Add the alias jc6202, pointing to the virtual host to the hosts file on the physical host. Add the alias for the physical host to the /etc/hosts on the virtual host.

Appliance Customization

To prevent the GlassFish Application Server Admin Console form going off to the Internet to try to get a Sun Commercial displayed in the bottom of the initial window, add the following JVM argument:

```
-Dcom.sun.enterprise.tools.admingui.NO_NETWORK=true
```

On the physical host (for me mcz02) create the following directory hierarchy:

```
C:\JCAPS62Projects\HL7Resilience\data
```

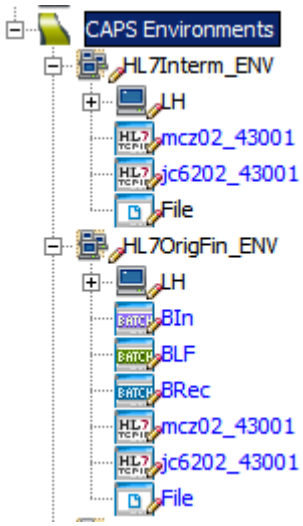
The data directory will contain HL7 files to be processed and these produced as the result of processing. Obtain and unzip into the `sources` sub-directory of the `data` directory, content of the archive HL7_A03_sources_sources.zip. This archive can be obtained from http://blogs.czapski.id.au/wp-content/uploads/2010/04/HL7_A03_sources_sources.zip.

Project Deployment

Archive JC62_HL7_Resilience_Project_Exports_no_Envs.zip, at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_no_Envs.zip, which contains no Java CAPS Environments, or JC62_HL7_Resilience_Project_Exports_with_Envs.zip, at http://blogs.czapski.id.au/wp-content/uploads/2010/04/JC62_HL7_Resilience_Project_Exports_with_Envs.zip, which contains Java CAPS Environments, are the project exports of all projects used in the exercise. Download whichever archive is appropriate for you and import it into you Java CAPS IDE.

Creation of a deployment profile, build and deployment assume that there exists Java CAPS environments created and configured correctly. The project export JC62_HL7_Resilience_Project_Exports_with_Envs.zip contains the environment and the project export JC62_HL7_Resilience_Project_Exports_no_Envs.zip does not.

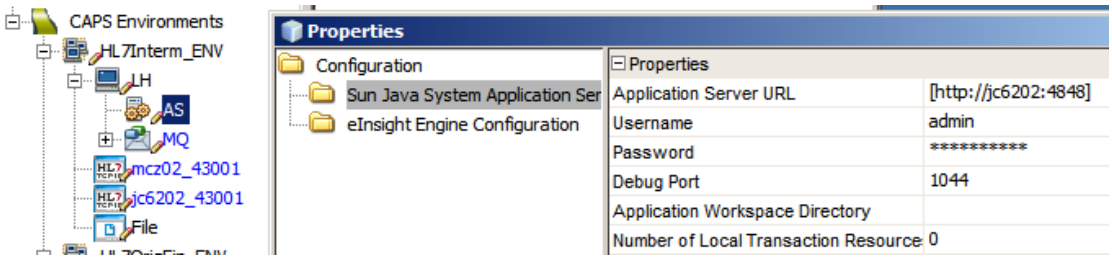
Projects 01SOriginalSender and 03RFinalReceiver will be deployed to the physical host, mcz02 in my case. Projects 02PIntermediateReceiver, 02RIntermediateProcessor and 02SIntermediateSender will be deployed to the virtual host, jc6202 in my case.



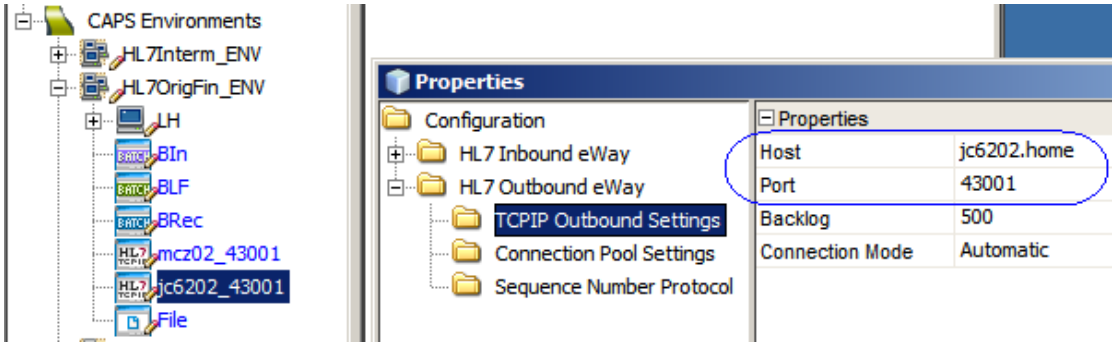
It follows that there must be two Java CAPS Environments, the one that corresponds to the physical host (mcz02) and the one that corresponds to the virtual host (jc6202).

The environment named in the picture HL7OriFin_ENV contains property settings for the physical host (mcz02). It includes the Batch Local File container (BLF), the Batch Inbound container (Bin), the Batch Record container (BRec), a File container (which is configured in the standard HL7 Outbound connectivity map but not used in this exercise) and one HL7 container with correctly configured outbound section. It also needs to be configured to allow deployment to the mcz02 runtime infrastructure.

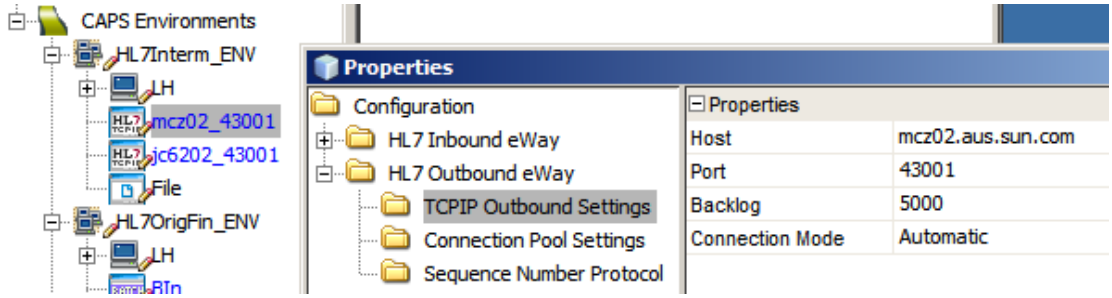
The environment named in the picture HL7Interm_ENV only needs a File outbound (not used in the exercise but used in the HL7 Outbound connectivity map) and the HL7 container with the outbound section configured to connect to the physical host (mcz02). It also needs to be configured to allow deployment to the jc6202 runtime infrastructure.



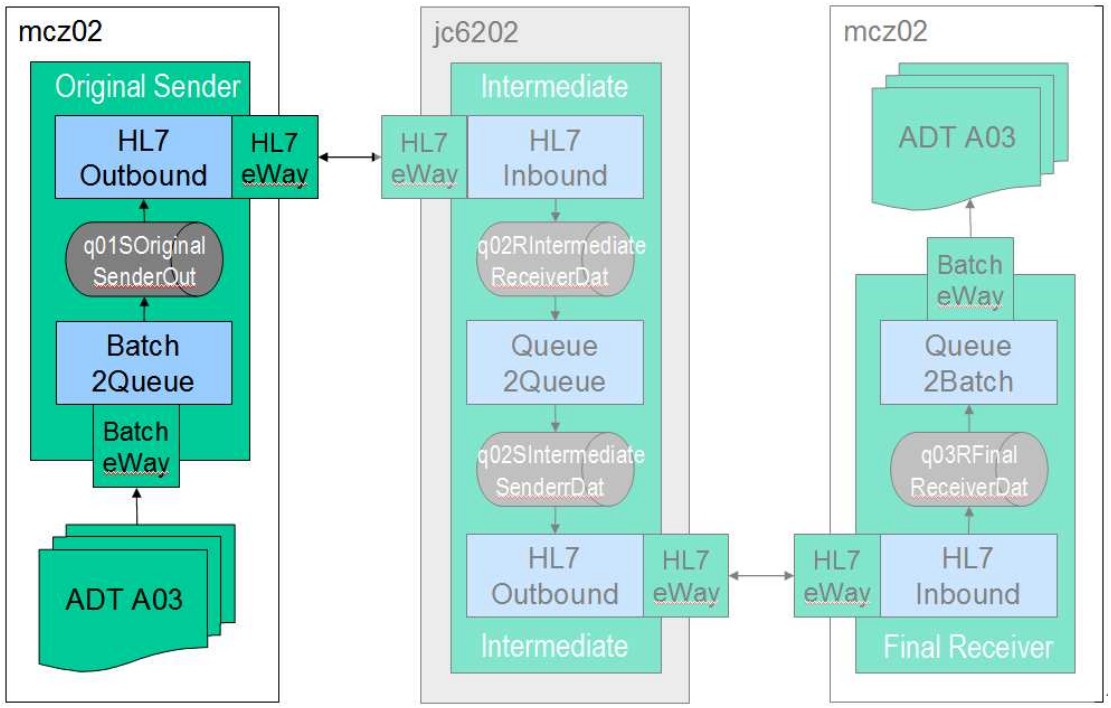
The outbound HL7 External System Container properties in the Java CAPS Environment HL7OriFin_ENV are configured such that the outbound HL7 eWay connects to the virtual host jc6202.



The outbound HL7 External System Container properties in the Java CAPS Environment HL7Interm_ENV are configured such that the outbound HL7 eWay connects to the physical host mcz02.

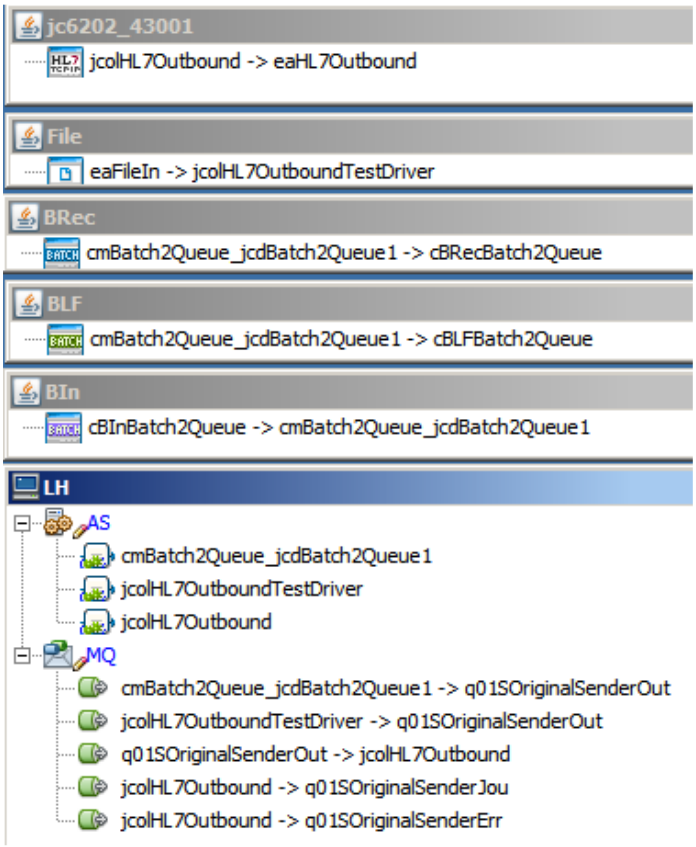


Project 01SOriginalSender will be deployed to the physical host mcz02. This project contains subprojects emulating the sender external and exercising Java CAPS HL7 recourse configuration and JMS redelivery handling, critical to ensuring that the sender does not lose messages.



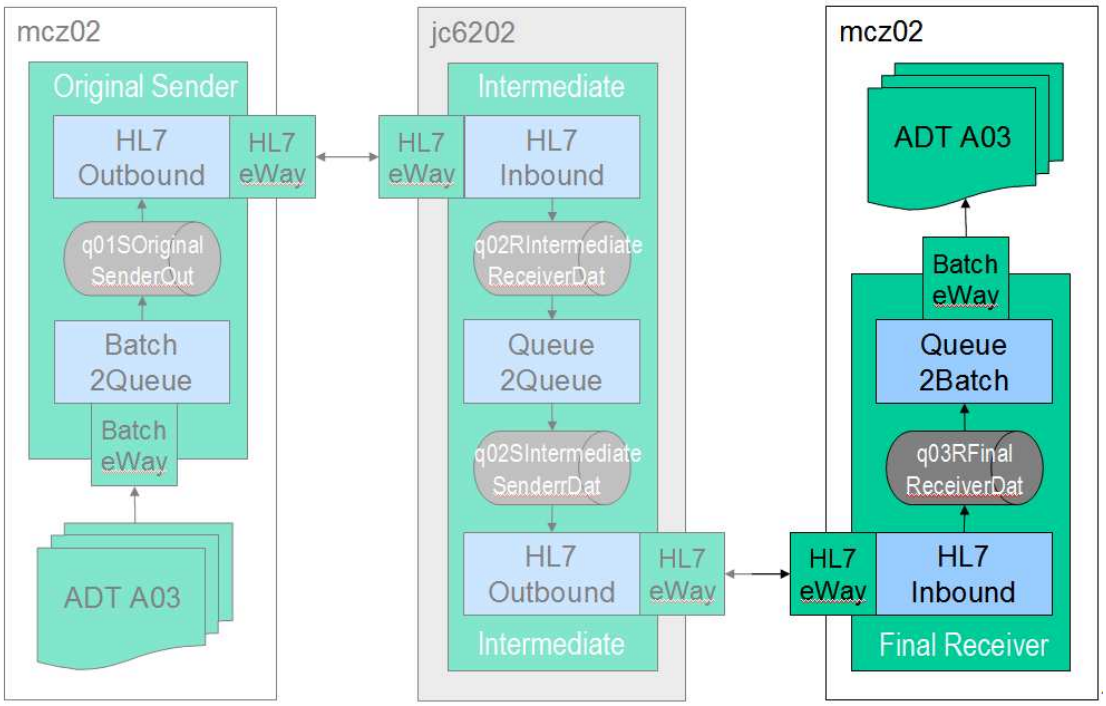
Expand the project structure through the 01SOriginalSender \01SOriginalSenderDP and create, in that subproject, a new deployment profile, for example named

dp01SOriginalSender_jc6202. Include both connectivity maps found in the 01SOriginalSenderDP project and make sure to map the HL7 Outbound eWay to the HL7 Client external system configured to connect to the jc6202 virtual host.

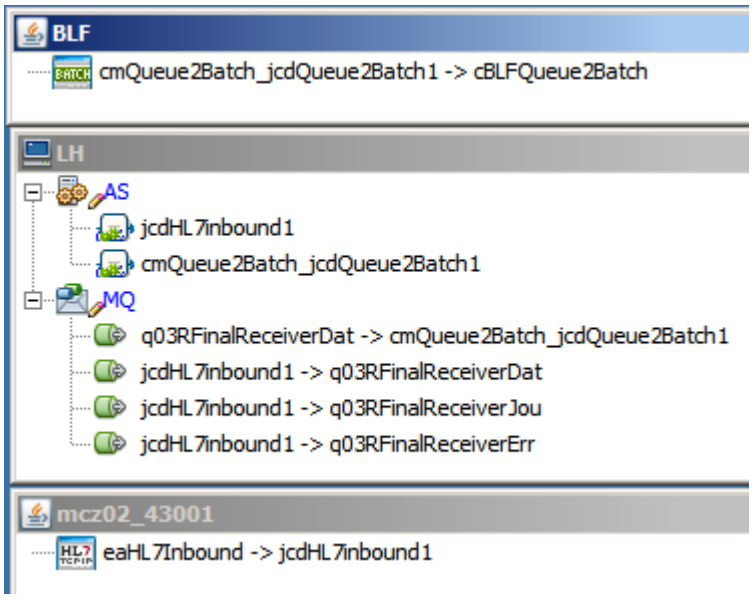


Build and deploy the project.

Project 03RFinalReceiver will also be deployed to the physical host mcz02. This project contains subprojects emulating the receiving external. It also writes files to a directory where they can be inspected to detect sequence breaks, if any, and duplication, if any.

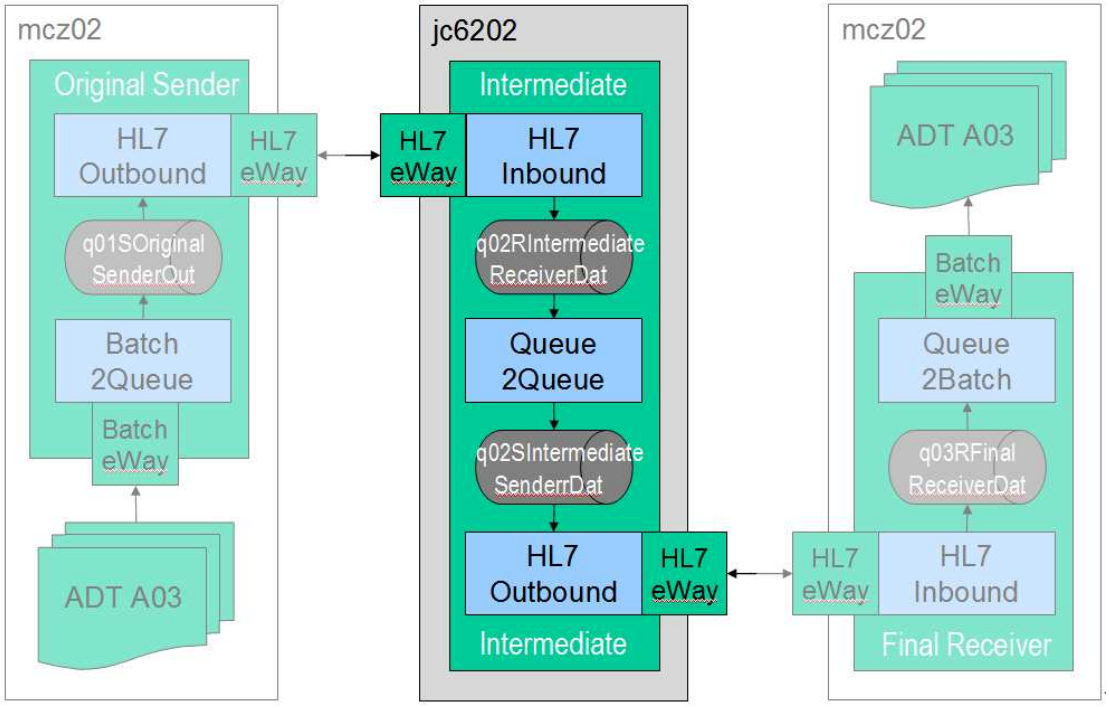


Expand the project structure through the 03RFinalReceiver\ 03RFinalReceiverDP and create, in that subproject, a new deployment profile, for example named dp03RFinalReceiver. Include both connectivity maps found in the 03RFinalReceiverDP.

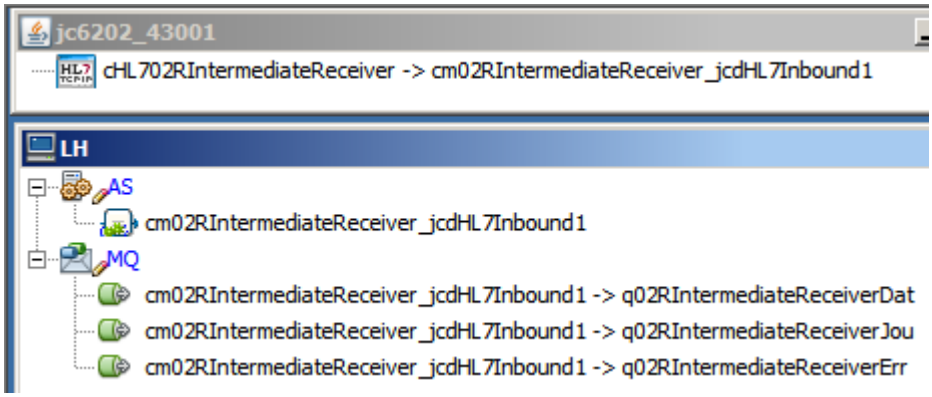


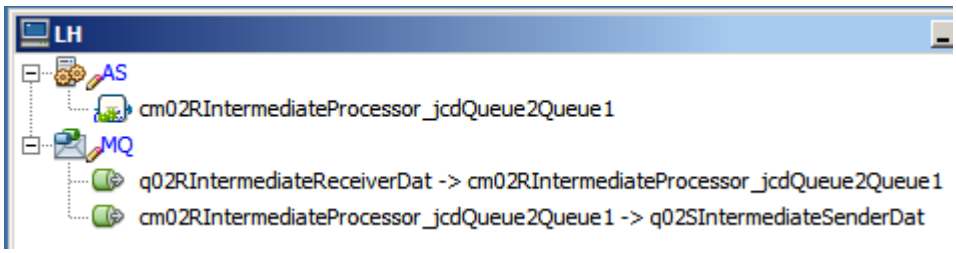
Build and deploy the project.

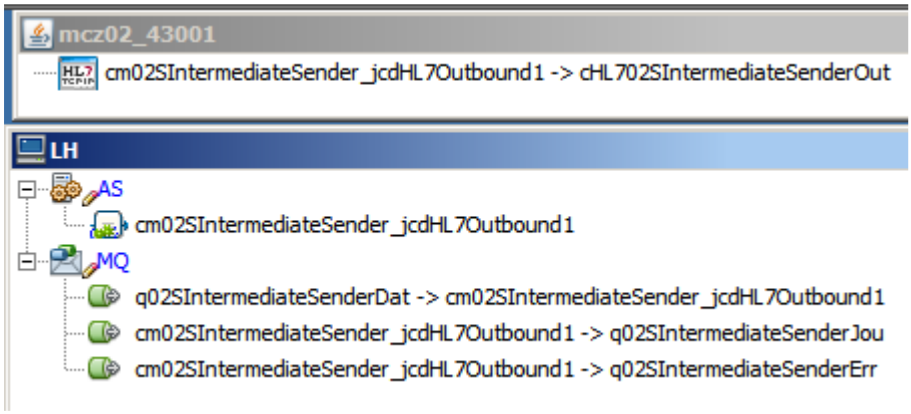
Projects 02PIntermediateReceiver, 02RIntermediateProcessor and 02SIntermediateSender will be deployed to the virtual host jc6202. These projects contain subprojects to receive HL7 messages, log MSH-10 to server.log and forward HL7 messages to the external system. There was no particularly good reason why they should have been developed as separate, discrete projects. The functionality could have been combined into a single project with a single deployment profile.



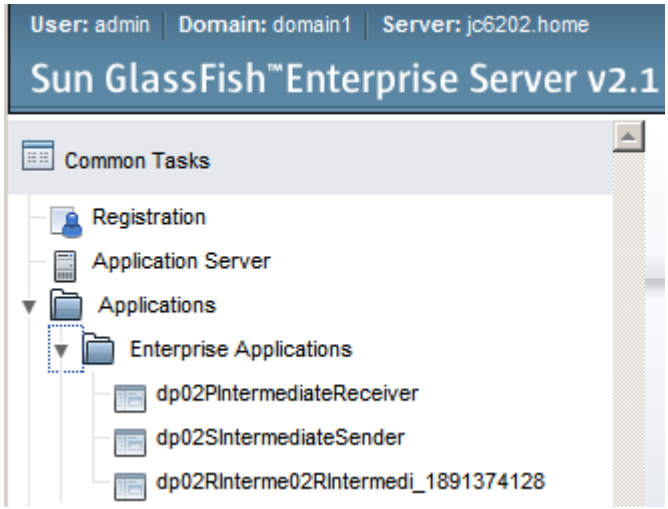
Expand project structures and create new deployment profiles, for example named dp02PIntermediateReceiver, dp02RIntermediateProcessor and dp02SIntermediateSender. Each should have a single connectivity map. Make sure to map the HL7 Outbound eWay in the dp02SIntermediateSender to the HL7 Client external system configured to connect to the mcz02 physical host.







Build and deploy the three projects, making sure they are deployed to the virtual host, jc6202.

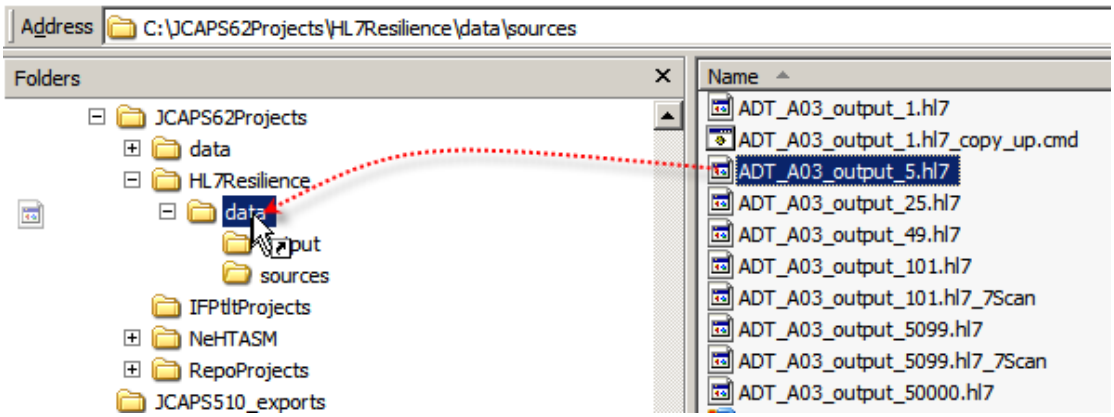


Test Preparation

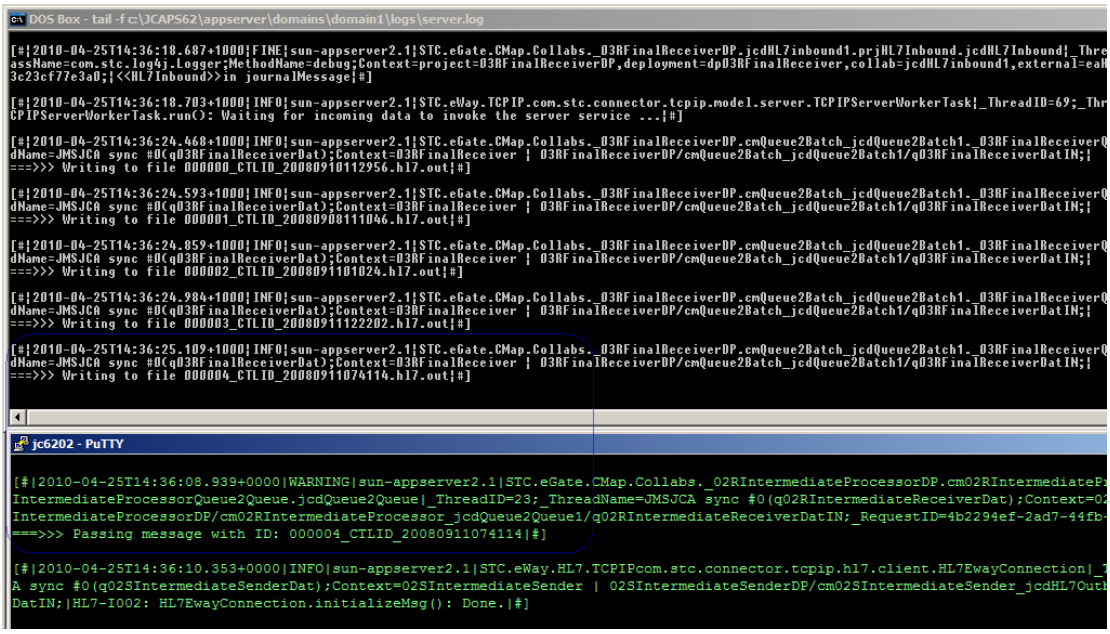
The environment must be brought to the state ready for testing. To prepare the environment perform the following steps:

1. Start Java CAPS Runtime environment (GlassFish Application Server) on the physical host.
2. tail server.log in a console window on the physical host to see activity as it occurs
3. Start jc6202 VM until it shows the IP address
4. Start Putty/SSH Client on jc6202 and tail server.log
5. Move the Putty/SSH Client console windows around in such a way that the bottom 1/3rd of it shows the output of a “tail” command continuously showing the server.log on each of the hosts.

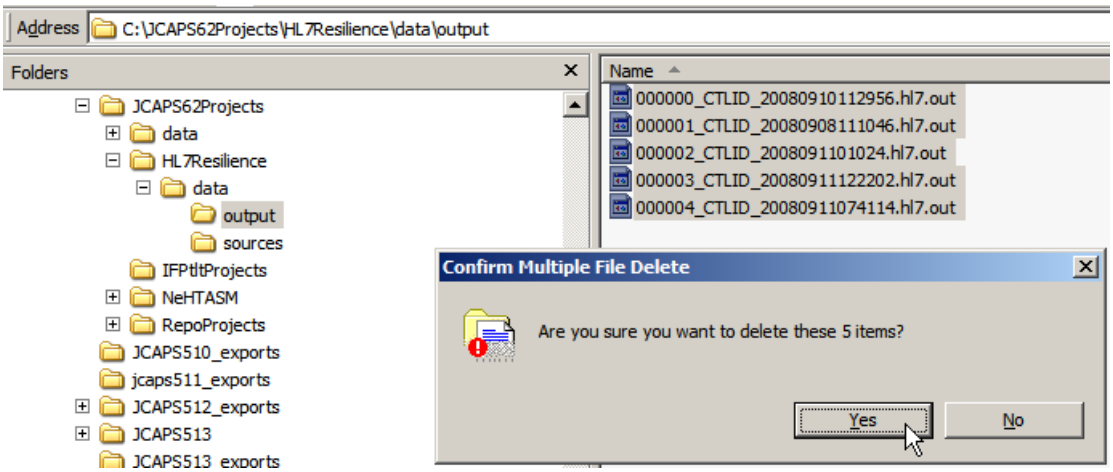
6. Submit 5 message set by copying the ADT_A03_output_5.hl7 file to data directory, to prime the infrastructure and make sure all components work



7. Observe messages being processed by the jc6202 host



8. once all messages are processed, clear output directory of messages



Testing resilience of the HL7 Solution

We intend to start with both “machines” and all components started and processing messages. This is what was happening in the preparation stage discussed in the previous section. We expect messages to be processed sequentially. Once we begin to see HL7 messages appear in the output directory, we will “crash” jc6202 by closing the VMware Player window in which it runs. We will then boot the machine again and when the message flow resumes we will shut down the machine in an orderly manner using the operating system shutdown command. When the machine is shut down we will boot it again. As soon as the message flow resumes we will shut down the application server using the application server console. When the application server is shut down we will start it again. As soon as the message flow resumes we will “disrupt the network” by disabling the VM network interface. When jc6202 exhausts the pool of messages queued by the HL7 inbound, and stops logging new messages, we will enable the VM network interface again. When the message flow resumes we will use the Enterprise Manager on jc6202 to shut down the inbound HL7 eWay. When the message flow stop and jc6202 stops logging new messages, we will start the inbound HL7 eWay again. Once the message flow resumes we will stop the HL7 Inbound on mcz02 to simulate external system down event. When file writing stops we will re-start the HL7 Inbound on mcz02 and allow message processing to run to completion.

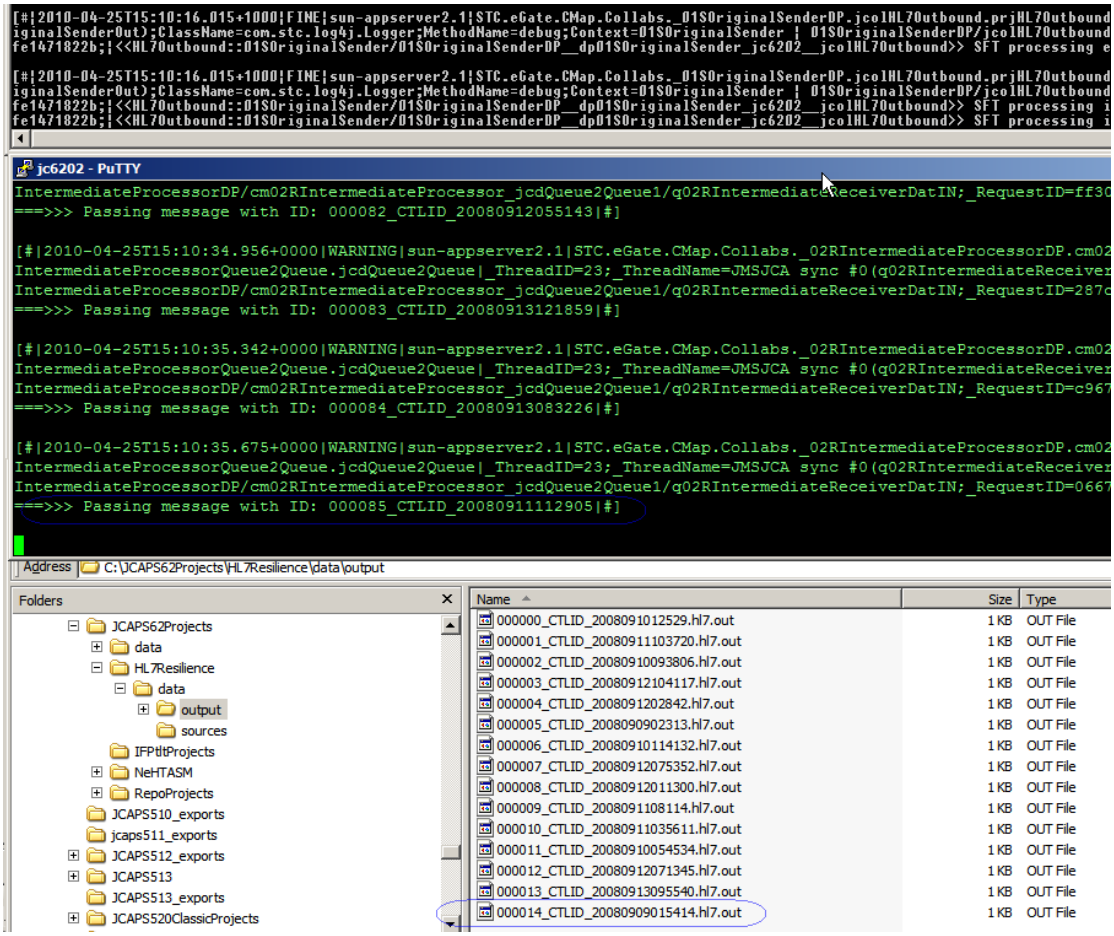
The VM crash and orderly shutdowns will cause retry functionality in the mcz02-based HL7 sender, after timeout period, to be invoked. We will see HL7 eWay retry attempts in the server.log, resets when the recourse actions are invoked and JMS redelivery attempts when HL7 eWay recourse actions cause exceptions and transaction rollbacks. The serve.log will have a great deal of messages logged - way too many to show in the article. I encourage you, however, to take a closer look to see how connection aborts, retries, resets and JMS rollbacks are manifested in the server.log so you can tell one when you see one.

We expect message flow to stop each time we interfere with the jc6202 or the components which it runs and others which run on mcz02. No new file will be written to the output directory once buffers are emptied.

We are not configuring this solution to maintain message flow in the face of component failures/interruptions – for a discussion of that topic see “GlassFish ESB v2.2 Field Notes – Exercising Load Balanced, Highly Available, Horizontally Scalable HL7 v2 Processing Solutions” at http://blogs.czapski.id.au/?p=13, bearing in mind that while it discusses components of the GlassFish ESB product the concepts and methods are equally applicable to the Java CAPS 6/Repository-based solutions.

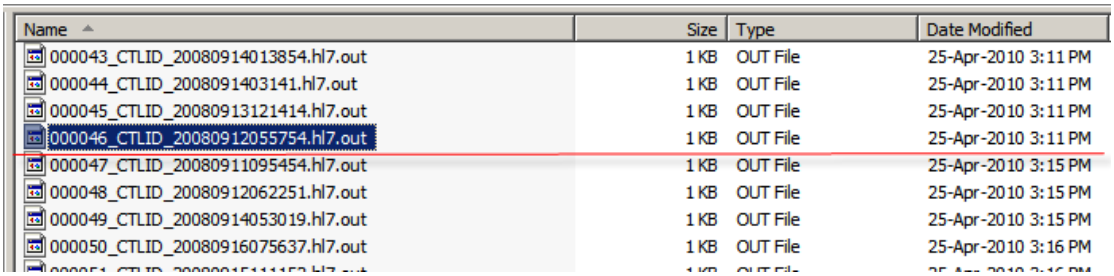
We are configuring this solution to make sure that we will not lose messages despite machine and component failures.

Let’s begin by copying the file ADT_A03_output_5099.hl7, containing 5099 ADT A03 messages, to the data directory, and observing the console windows and the output directory until around 15 messages are written to the output directory.



Note that the jc6202 log shows message 55 being processed and there are only 14 messages in the output directory. This is because messages are buffered in the JMS infrastructure in jc6202 and JMS infrastructure on mcz02, and writing to a file system directory is much slower then sending messages over the wire.

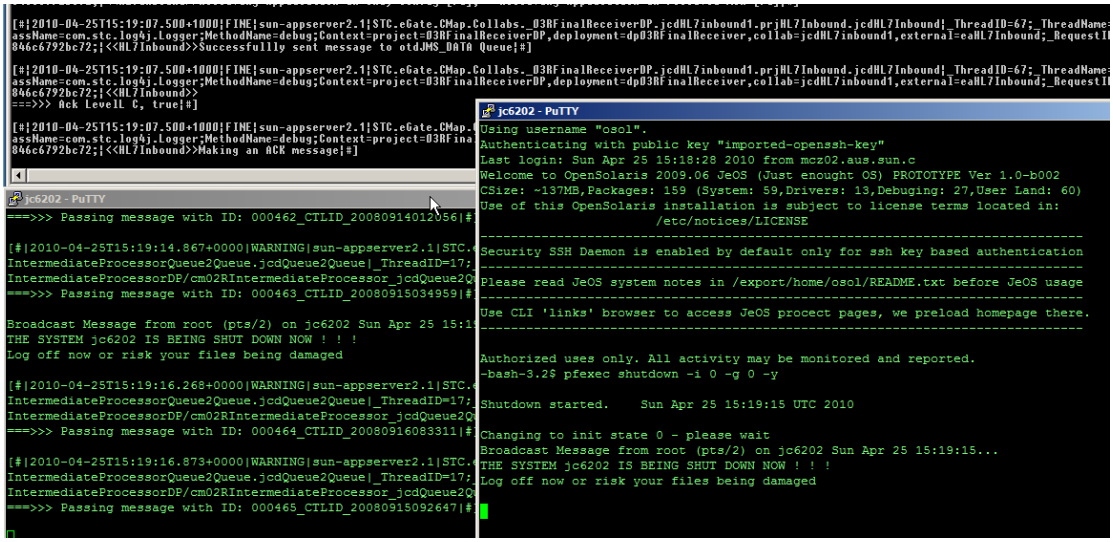
At the time all outstanding files were written, as distinct from the time the screenshot above was taken, there were 47 messages in the output directory, the last being 000046_CTLID_20080912055754.hl7.out.



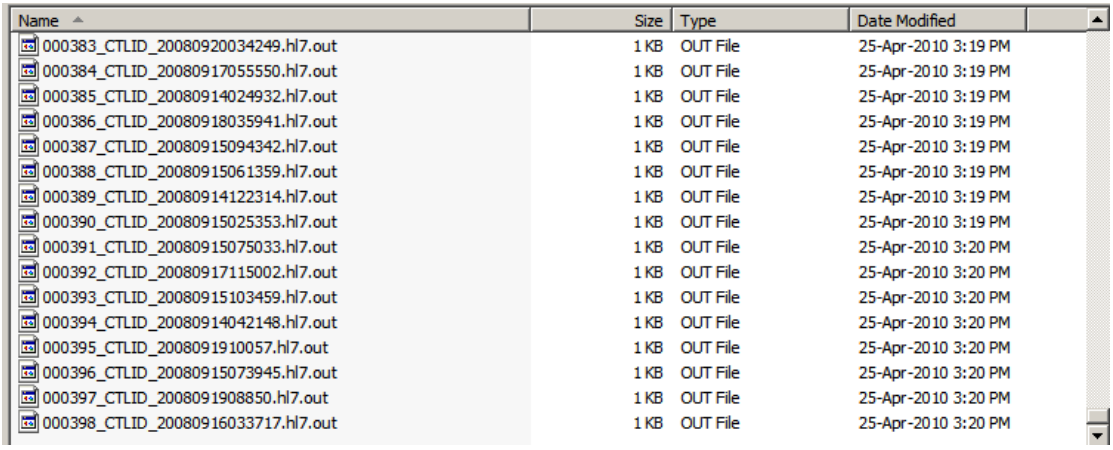
The machine took about 4 minutes to come back up and for the messages to start flowing again. Note that no message was lost.

Boot the virtual machine, jc6202, connect to the console, tail server.log and watch for message processing to resume.

Now that the machine is back up let's shut it down using the operating system shutdown command.



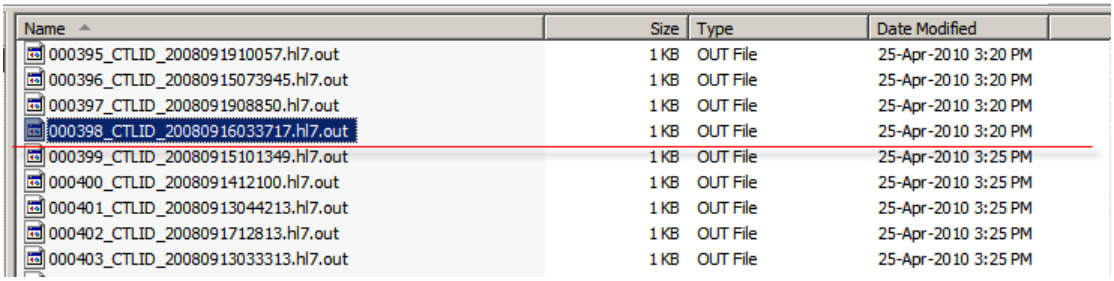
Last message written was message with the sequence number of 000398.



Name	Size	Type	Date Modified
000383_CTLID_20080920034249.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000384_CTLID_20080917055550.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000385_CTLID_20080914024932.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000386_CTLID_20080918035941.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000387_CTLID_20080915094342.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000388_CTLID_20080915061359.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000389_CTLID_20080914122314.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000390_CTLID_20080915025353.hl7.out	1 KB	OUT File	25-Apr-2010 3:19 PM
000391_CTLID_20080915075033.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000392_CTLID_20080917115002.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000393_CTLID_20080915103459.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000394_CTLID_20080914042148.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000395_CTLID_2008091910057.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000396_CTLID_20080915073945.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000397_CTLID_2008091908850.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000398_CTLID_20080916033717.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM

Boot the virtual machine, jc6202, connect to the console, tail server.log and watch for message processing to resume.

At the time all outstanding files were written, as distinct from the time the screenshot above was taken, there were 399 messages in the output directory, the last being the message with the name 000398_CTLID_20080916033717.hl7.out.



Name	Size	Type	Date Modified
000395_CTLID_2008091910057.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000396_CTLID_20080915073945.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000397_CTLID_2008091908850.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000398_CTLID_20080916033717.hl7.out	1 KB	OUT File	25-Apr-2010 3:20 PM
000399_CTLID_20080915101349.hl7.out	1 KB	OUT File	25-Apr-2010 3:25 PM
000400_CTLID_2008091412100.hl7.out	1 KB	OUT File	25-Apr-2010 3:25 PM
000401_CTLID_20080913044213.hl7.out	1 KB	OUT File	25-Apr-2010 3:25 PM
000402_CTLID_2008091712813.hl7.out	1 KB	OUT File	25-Apr-2010 3:25 PM
000403_CTLID_20080913033313.hl7.out	1 KB	OUT File	25-Apr-2010 3:25 PM

The machine took about 4 minutes to come back up and for the messages to start flowing again. Note that no message was lost.

Let's use the OpenSolars service management facilities (SMF) to shut down the GlassFish Application Server.

```
-bash-3.2$
-bash-3.2$ pfexec svcadm disable domain1
-bash-3.2$
```

After a while the application server stops and files cease to be written to the output directory.

```
[#|2010-04-25T15:29:12.363+0000|INFO|sun-appserver2.1|com.sun.jbi.framework|_ThreadID=21;_ThreadName=RMI TCP Connection(8)-127.0.0.1;|JBIFW0042: JBI framework termination complete.|#]
```

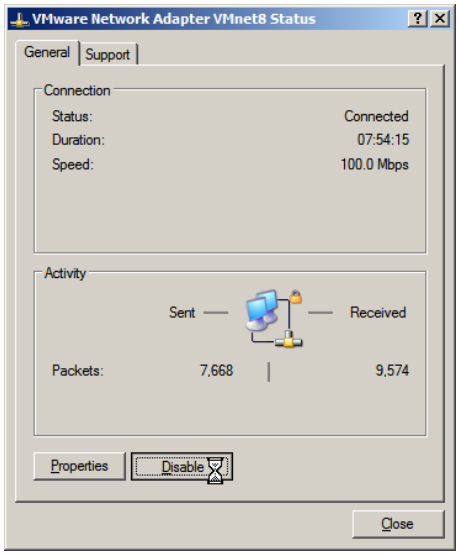
Let's use the SMF to start the application server again. Once it starts processing messages again let's inspect the output directory.

```
-bash-3.2$
-bash-3.2$ pfexec svcadm enable domain1
-bash-3.2$
```

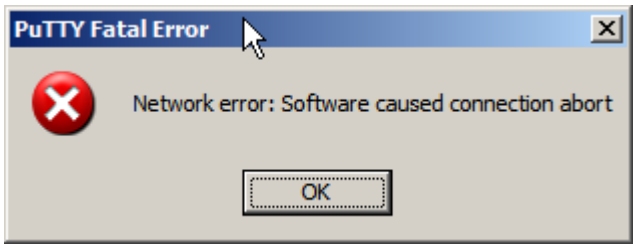
As before, message processing resumed without losing any messages.

000737_CTLID_20080919014434.hl7.out	1 KB	OUT File	25-Apr-2010 3:28 PM
000738_CTLID_20080918072547.hl7.out	1 KB	OUT File	25-Apr-2010 3:29 PM
000739_CTLID_20080919074627.hl7.out	1 KB	OUT File	25-Apr-2010 3:29 PM
000740_CTLID_20080919035813.hl7.out	1 KB	OUT File	25-Apr-2010 3:29 PM
000741_CTLID_20080918042806.hl7.out	1 KB	OUT File	25-Apr-2010 3:29 PM
000742_CTLID_20080920115545.hl7.out	1 KB	OUT File	25-Apr-2010 3:29 PM
000743_CTLID_20080920021018.hl7.out	1 KB	OUT File	25-Apr-2010 3:32 PM
000744_CTLID_20080917054633.hl7.out	1 KB	OUT File	25-Apr-2010 3:32 PM
000745_CTLID_20080918081457.hl7.out	1 KB	OUT File	25-Apr-2010 3:32 PM
000746_CTLID_20080920043209.hl7.out	1 KB	OUT File	25-Apr-2010 3:32 PM

Let's now "disrupt the network" by disabling the VM network interface.



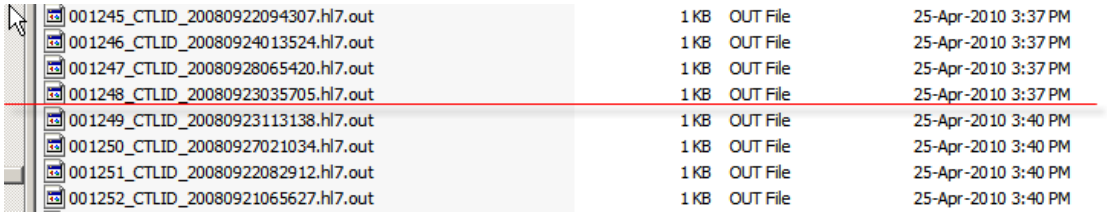
PuTTY dropped out, as did the interface between mcz02 and jc6202.



jc6202 continues processing and queuing messages it already received.

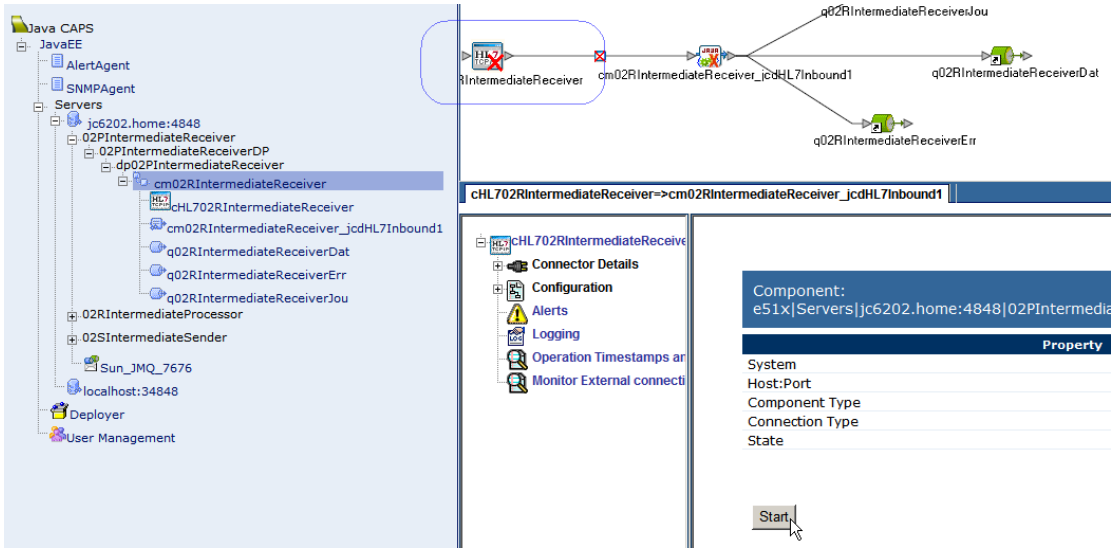
Let's enable the VM network interface and inspect the output directory.

After a while, during which the HL7 Outbound on the jc6202 re-establishes a connection to the HL7 receiver on mcz02, writing of messages to the output directory resumes again.



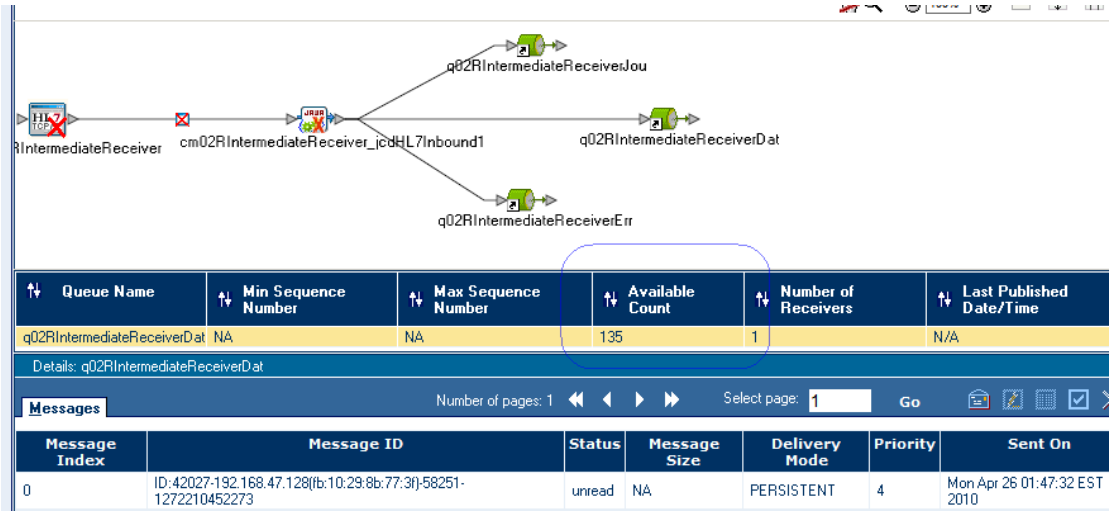
As before, no message was lost.

Let's now stop the inbound HL7 interface on jc6202 using the Java CAPS Enterprise Manager.

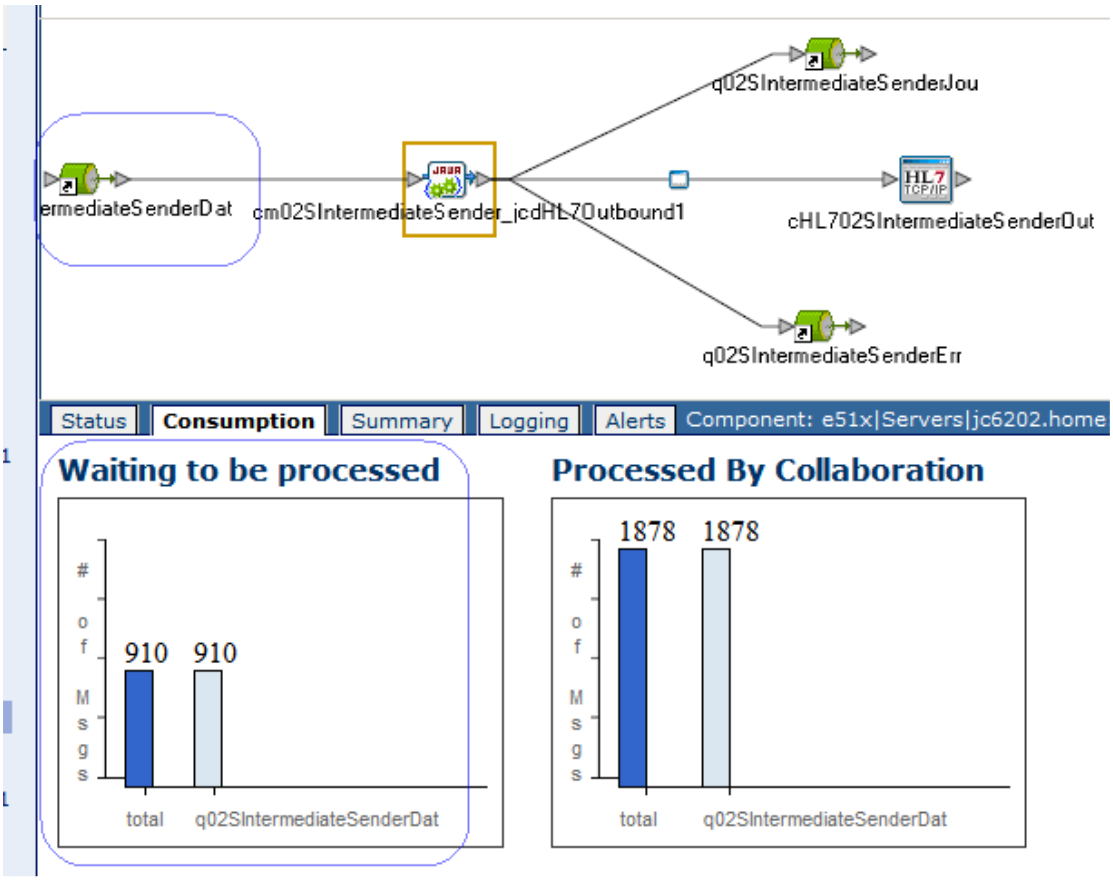


This has the effect of disrupting message inflow. Eventually jc6202 will process all messages and will deliver them to the outbound. How soon this will happen will depend on how many messages are queued in the infrastructure. The Enterprise Manager will reveal that when asked.

On the queue from the now-stopped HL7 Inbound.

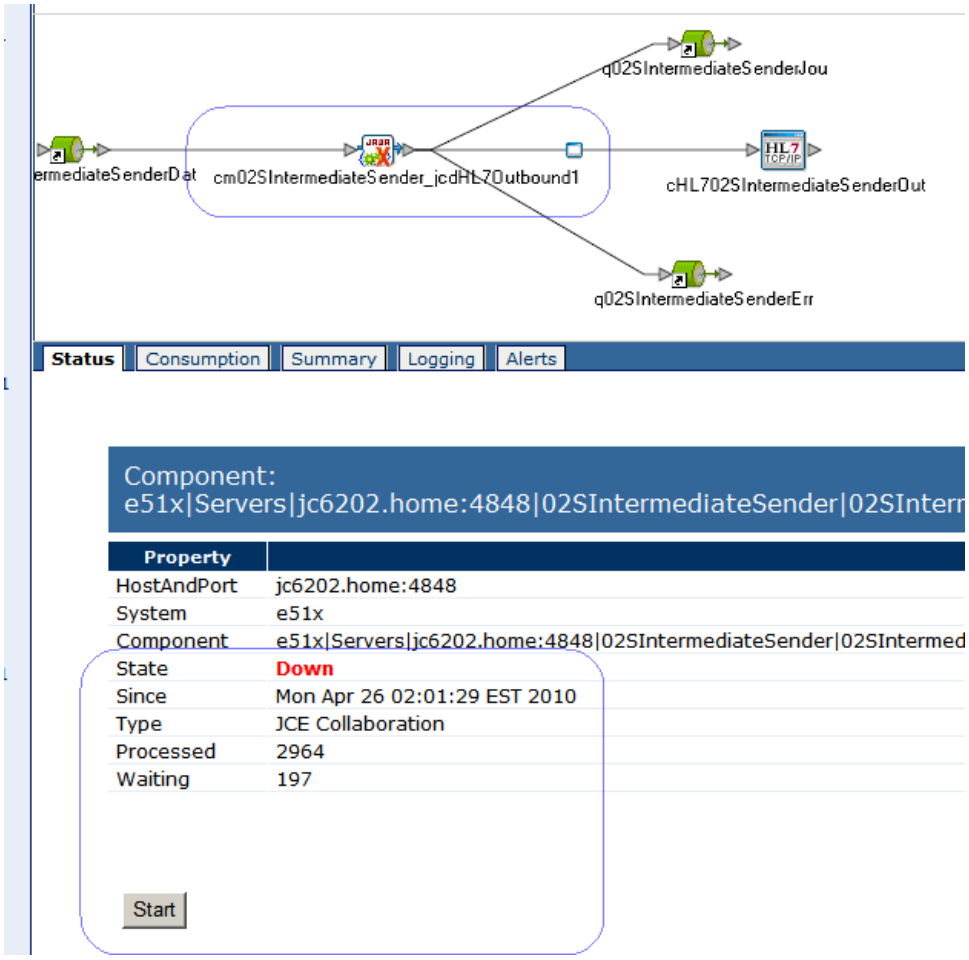


On the queue into the still-running HL7 Outbound.



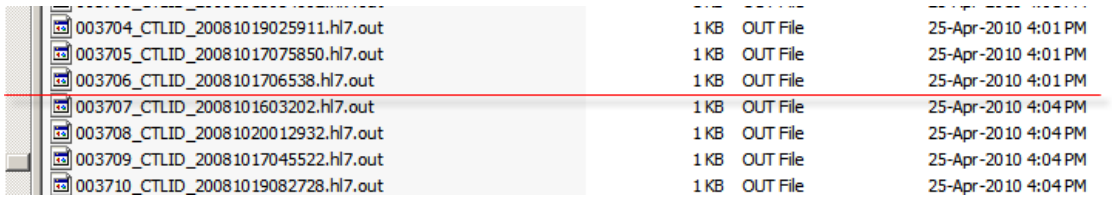
Let's leave the infrastructure running until all queues are empty. The last message written to a file was 003530.

Let's start the HL7 inbound and, when a few files get written to the output directory, stop the collaboration that delivers to the HL7 outbound to demonstrate that messages are queued in jc6202 when the outbound interface to the external system is shut down.

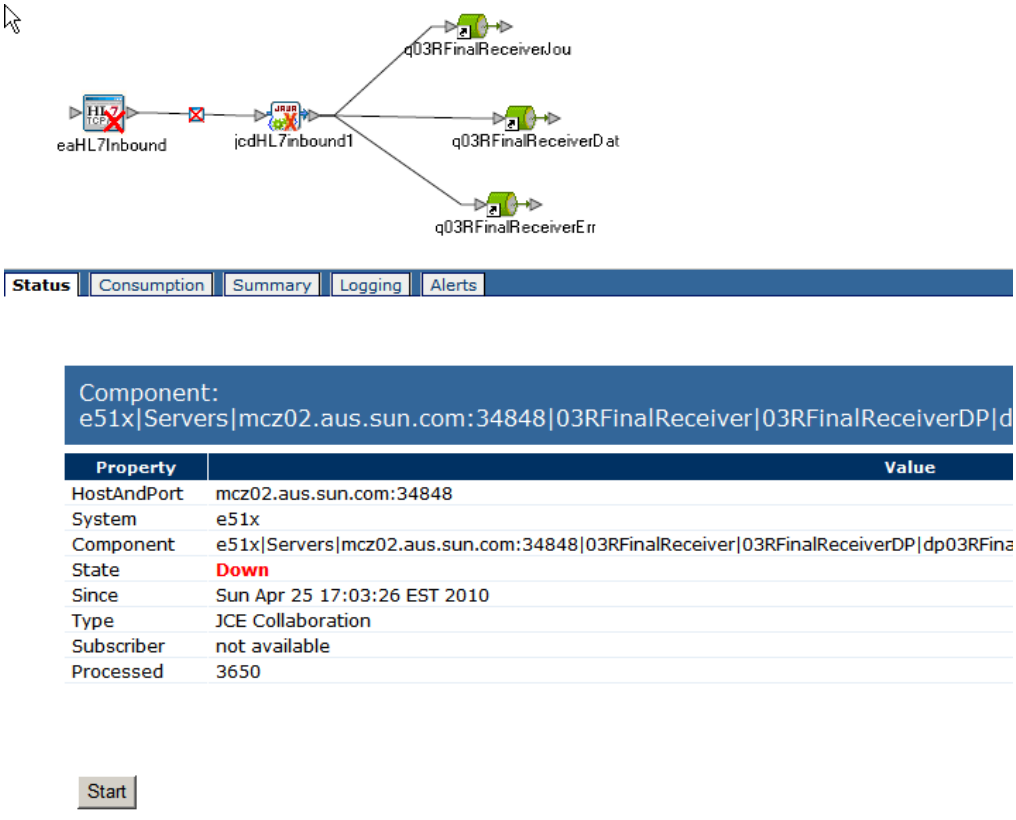


Last message file written was 003706.

Let's start the interface again to allow message flow to resume.

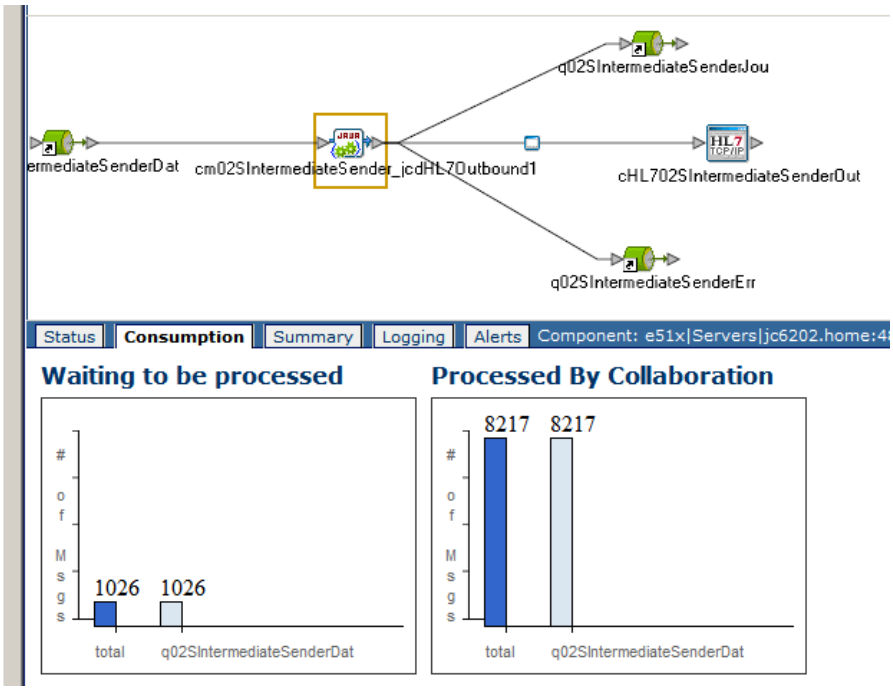


Once we start getting new files written to the directory let's stop the HL7 Inbound on mcz02 to simulate external system down scenario.



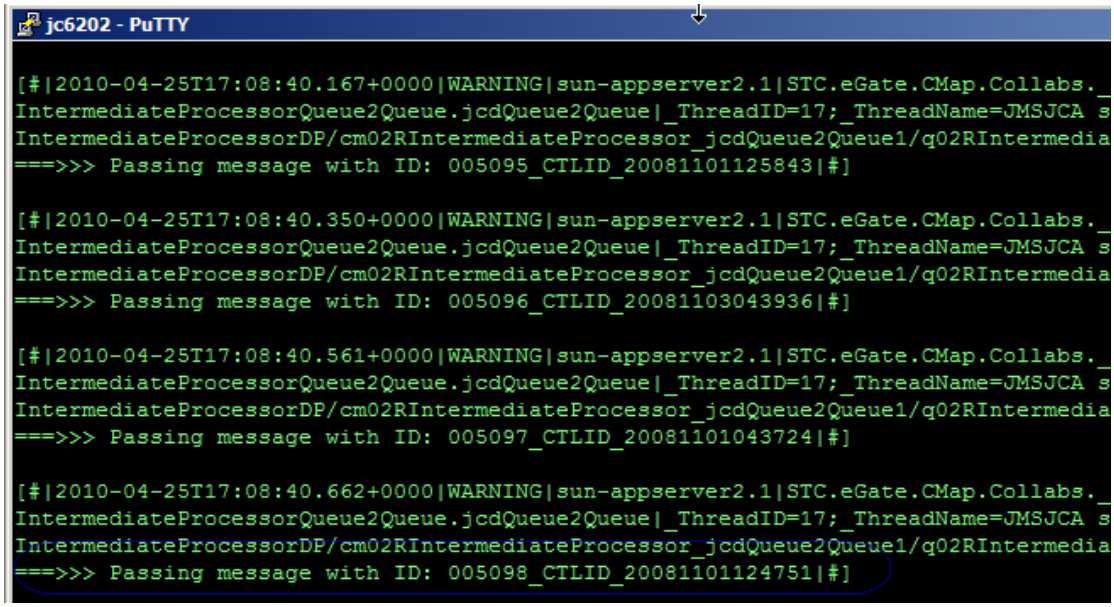
Let's re-start the HL7 Inbound after a few minutes to allow message flow to resume.

Since the HL7 Outbound on jc6202 could not deliver to the listener on mcz02 messages were queued. Over 1000 messages were queued in the time the outbound was unable to connect. It will now process the backlog.

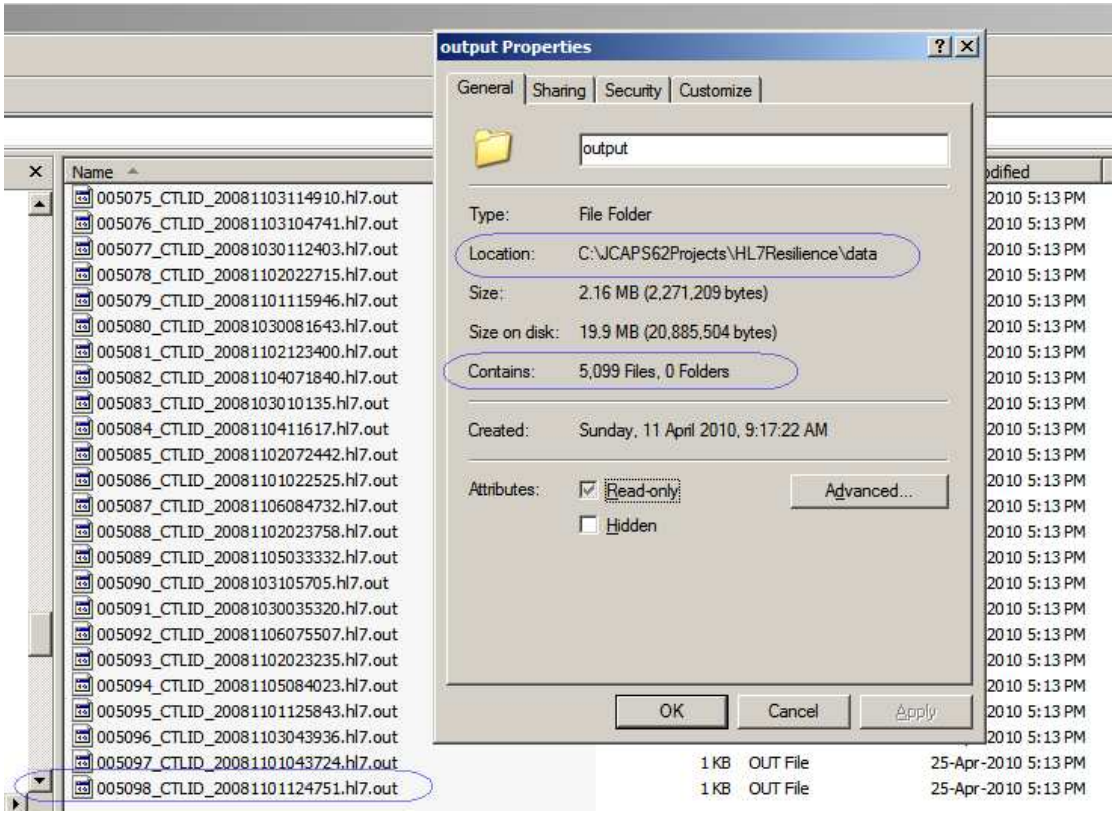


We will allow the processing stream to continue until all 5099 messages are processed and written to the file system directory.

First the intermediate processor processes all 5099 messages, as indicated in the log by a message with the sequence number of 005098.



Then, some time later, the HL7 Outbound on jc6202 will send them to the HL7 Inbound on mcz02 where they will be written to disk. The delay is occasioned by the message queuing between components both on jc6202 and on mcz02.
Indeed, all 5099 files were successfully processed and written to disk.



We know that in a correctly configured Java CAPS environment messages will not be lost and will not be duplicated.

Summary

This note walked through the preparation of the Java CAPS 6.2 VMware Virtual Appliances for a HL7 messaging resilience exercise and deploying ready-made Java CAPS 6.2 HL7 solutions. The exercise for HL7-based resilient solution, processing HL7 v2.3.1 messages, was conducted and discussed.

We are convinced that a resilient Java CAPS solution can be configured and that it will process messages in the face of typical failure and disruption scenarios without message loss or duplication.