GlassFish ESB v2.2 Field Notes

Exercising Load Balanced, Highly Available, Horizontally Scalable HL7 v2 Processing Solutions

Michael.Czapski@sun.com

January 2010, Release 1.0.0.1

Table of Contents

Introduction

It seems frequently assumed that architecting and deploying Highly Available (HA) solutions requires Application Server and/or Operating System clustering. When it comes to SOA and Integration solutions this is not necessarily a correct assumption. Load Balanced (LB) and Highly Available HA) SOA and Integration solutions may not require that degree of complexity and sophistication. Frequently, protocol, binding component, JBI and architectural application design properties can be exploited to design highly available solutions. Testing LB and HA solutions requires infrastructure consisting of multiple hosts and the ability to "crash" hosts at will. With virtualization technologies available now it is far easier to use multiple virtual machines then to use physical machines. It is also easier and potentially less destructive to "crash" virtual machines then it is to do so with physical machines.

In this Note a heterogeneous, non-clustered collection of hosts will be used to implement and exercise three load balanced, highly available GlassFish ESB-based solutions. The environment consists of a number of independent “machines”, which are not a part of an Operating System Cluster. Each “machine” hosts a GlassFish Application Server. Application Servers are independent of one another and are not clustered. This is to demonstrate that load balanced, highly available, horizontally scalable solutions, based on the GlassFish ESB software alone, can be designed and implemented.

The specific class of solutions to which this discussion applies is the class of solutions which:

1. are exposed as request/reply services
    a. HL7 messaging with explicit Application Acknowledgement
       or
    b. Request/Reply Web Services
       or
    c. JMS in Request/Reply mode
2. implement business logic as short lived processes
3. are
    a. atomic
       or
    b. are idempotent
       or
    c. tolerant of duplicate messages

Classes of solutions with characteristics different from these named above require different approaches to high availability and horizontal scalability, and are not discussed here.

In this Note only high availability and scalability of receiver solutions is addressed. This aspect is the focus because a failure to process a message by a receiver may result in message loss –generally a bad thing.

Paradoxical as it may sound; senders are special cases of receivers. Just as a receiver is triggered by arrival of a message so too is a sender. Making sure that the sender trigger message does not get lost is much the same as making sure the message a receiver receives does not get lost. This means that the same considerations apply to senders and to receivers.

This note discusses an exercise involving an example load balanced, highly available, horizontally scalable healthcare environment, processing HL7 v2 messages. Discussion includes customization of generic GlassFish ESB v2.2 VMware Virtual Appliances for a specific Load Balancing and High Availability exercise and deploying ready-made GlassFish ESB solutions. The exercise for HL7 BC-based, Web Service-based and JMS-based highly available, load balanced, and horizontally scalable receivers, processing HL7 v2.3.1 messages, will be conducted and discussed.

At the end of the Note we will have three GlassFish ESB VMware Appliances with GlassFish ESB v2.2 Runtime infrastructure, ready to use for further GlassFish ESB Load Balancing and High Availability exercises.

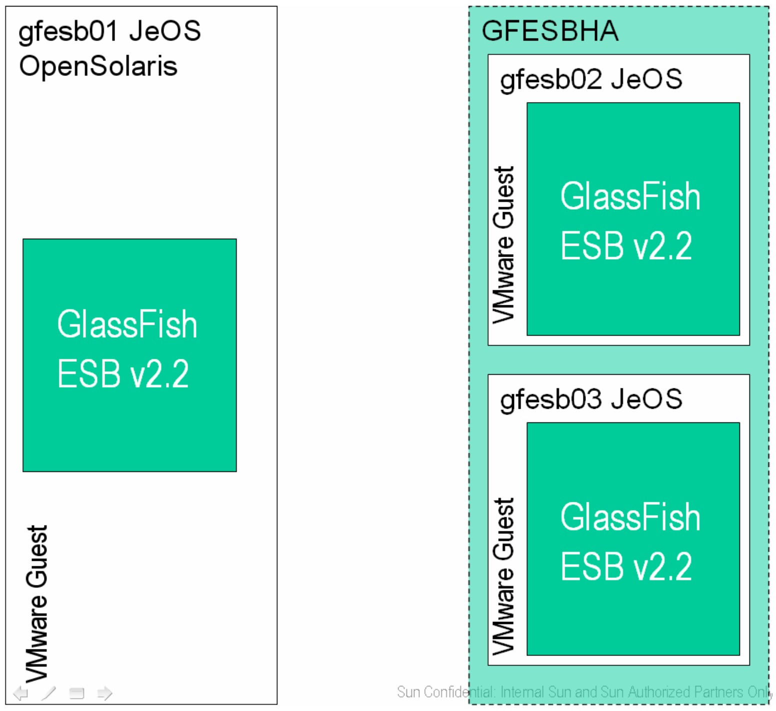
The reader will be convinced, one hopes, that for the applicable class of GlassFish ESB-based solutions, load balancing and dynamic failover without message loss work. For that class of solutions this provides for high availability and horizontal scalability without resorting to Application Server or Operating System clustering.

Exercise Environment

The runtime environment for the non-clustered Load Balancing and High Availability exercise will consist of three VMware Virtual Machines, each with GlassFish ESB v2.2 runtime installation, and the VMware Host machine which will serve as the Load Balancer machine, in addition to its regular duties as the VMware host.

Note that the VMware Virtual Machines, which will be used for this exercise, will use 1280, 640 and 640 Megabytes of physical memory each. The VMware Host must have at least 512 Megabytes of memory for its own Operating System so the absolute minimum physical memory required in the Host is 3 gigabytes. It will be a tight fit on a machine with so little memory so it will be much better to have a machine with sufficient physical memory to allow at least 1536, 1024 and 1024 Megabytes of memory for the VMware virtual machines (gfesb01, gfesb02 and gfesb03 respectively) and at least 1 Gigabyte of memory extra for the Host OS. This adds up to over 4.5 Gigabytes of physical memory. This also eliminates Operating Systems like 32-bit Windows XP, which will only support 3.2 Gigabytes of memory. Naturally, there may be multiple VMware Hosts used. Each VMware Virtual Machine could be running on its own physical machine, eliminating the memory shortage issue and OS memory support restrictions.

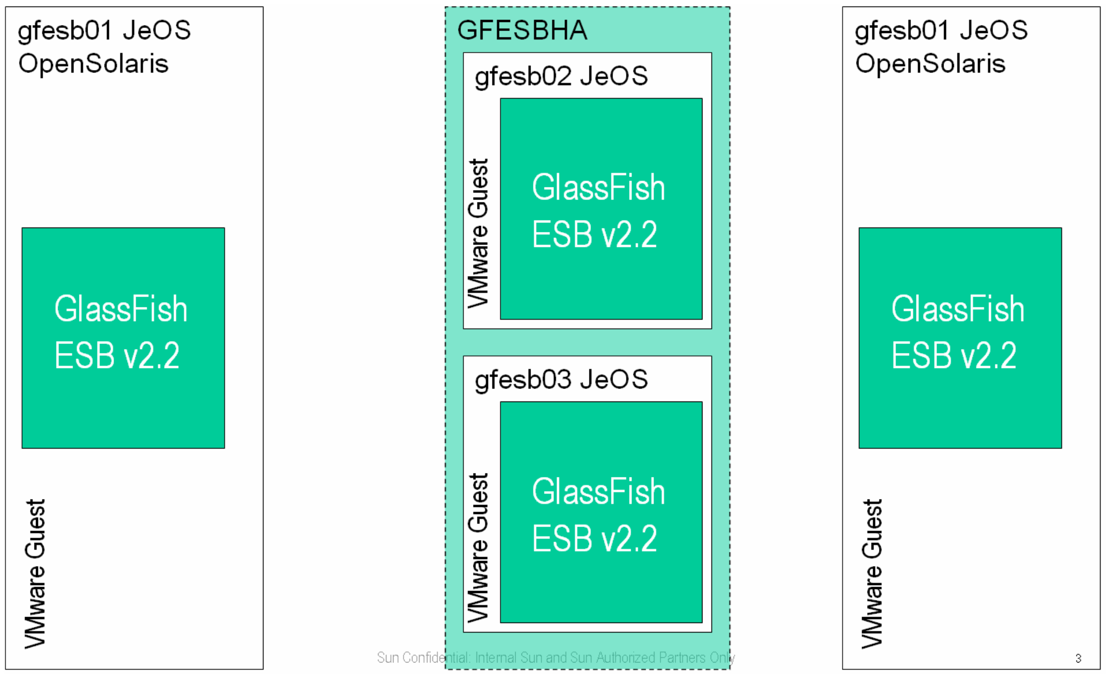
The schematic below depicts the “physical” environment.



gfesb02 and gfesb03 will be configured identically, from the GlassFish ESB solution perspective. They will host the same projects and will share the common JMS infrastructure with shared JMS Queues to which each will forward messages. Each message received by either of the two hosts will be processed identically.

Further hosts, with the GlassFish ESB environment configured like that on gfesb02 and gfesb03, can be added to horizontally scale the receiver solution.

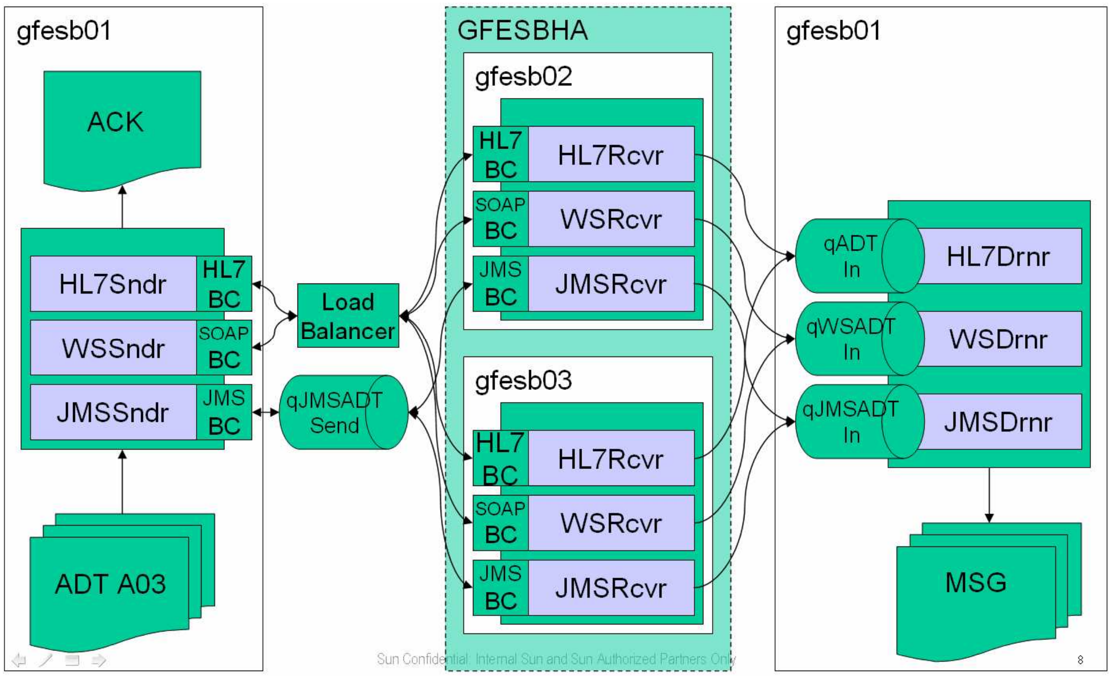
To facilitate discussion of the solution, the host environment will be shown as though gfesb01 consisted of two separate hosts, which it is not.



This is to convey the separation of the sender from the drainer and make diagrams more readable.

Solution Implementation

The solution is designed to exercise a heterogeneous, non-clustered, highly available, failure-proof configuration for HL7-based, Web Service-based and JMS-based messaging implementations.



For each transport protocol in use there are three components: a Sender, a Receiver and a Drainer.

The sender component reads multiple records from a file in a file system and sends each record as a separate message to the receiver, using the transport with which it is configured. It then waits for an application acknowledgement – a response message, send back by the receiver. The acknowledgement is written to a file with a unique name containing the message ID and a date/time stamp.

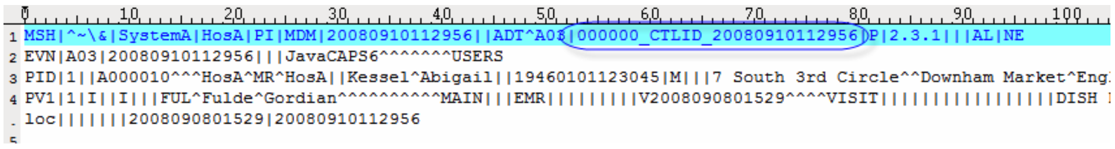
The receiver receives a message over the transport configured for it, sends the message to an external, shared JMS server, generates an application acknowledgement and sends it back to the sender.

The drainer receives a message from the external, shared JMS server, where the receiver put it, and writes it to a file with the unique name containing the message ID and a date/time stamp.

Receiver components use a common, shared JMS Server. This JMS server is assumed to be highly-available and hosted separately from the receiver components, such that a failure of a receiver host does not cause the failure of the JMS Server.

Both the HL7 BC-based, the Web Service-based and the JMS-based solutions process HL7 v2.3.1 ADT A03 messages, sent from the sender to the receiver, and HL7 v2.3.1 ACK messages, retuned by the receiver to the sender as application acknowledgments.

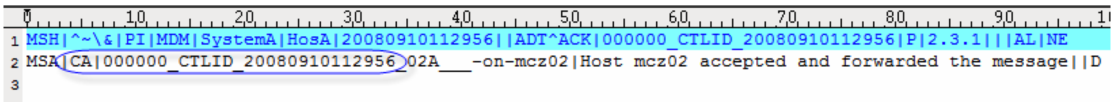
A typical A03 message might look like:



```
MSH|^~\&|SystemA|HosA|PI|MDM|20080910112956||ADT^A03|000000_CTLID_20080910112956|P|2.3.1|||AL|NE
EVN|A03|20080910112956|||JavaCAPS6^^^^^^^USERS
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7 South 3rd Circle^^Downham Market^Eng
PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^MAIN||EMR||||||||V2008090801529^^^^VISIT|||||||||||||||DISH
loc||||||2008090801529|20080910112956
```

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

A typical acknowledgement message might look like:



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

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 discussion. 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
```

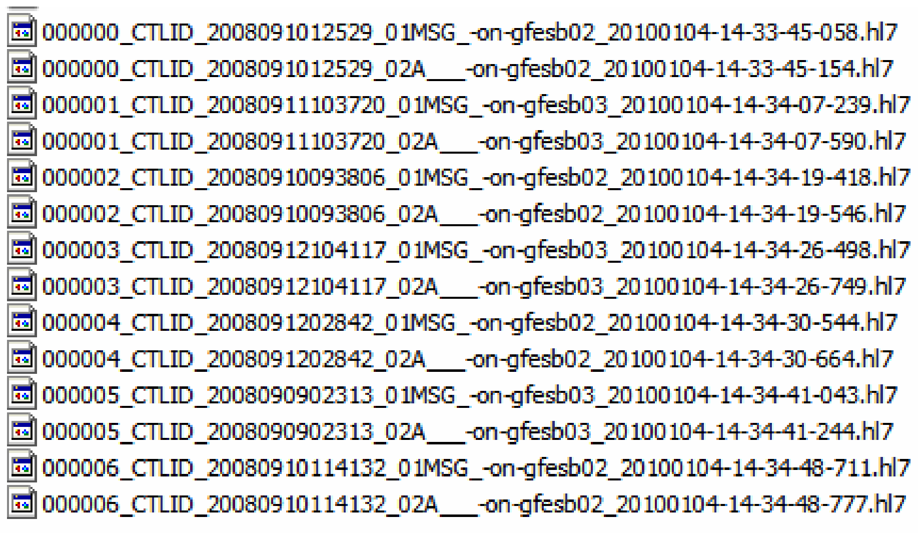
The first 6 digits of the message id are the serial number, which is unique, and contiguously increasing in each message. Message 1 will be 000000, message 2 will be 000001 and so on.

Names of files written by the sender, containing application acknowledgements, 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.

Names of files written by the drainer, 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.

Each receiver will determine the name of the host on which it is running and will embed that name in the message id prior to passing the message on. The names of the files written by the drainer and the sender will contain that host name. This will allow a human to determine which receiver (on which host) processed the message. The round-robin load balancing and fail-over processing will be readily apparent on inspection of file names.

File names will look like these shown below. Note the sequence numbers, message types (01MSG and 02A___), processing hosts (gfesb02, gfesb03) and timestamps embedded in file names.



```
000000_CTLID_2008091012529_01MSG_-on-gfesb02_20100104-14-33-45-058.hl7
000000_CTLID_2008091012529_02A___-on-gfesb02_20100104-14-33-45-154.hl7
000001_CTLID_20080911103720_01MSG_-on-gfesb03_20100104-14-34-07-239.hl7
000001_CTLID_20080911103720_02A___-on-gfesb03_20100104-14-34-07-590.hl7
000002_CTLID_20080910093806_01MSG_-on-gfesb02_20100104-14-34-19-418.hl7
000002_CTLID_20080910093806_02A___-on-gfesb02_20100104-14-34-19-546.hl7
000003_CTLID_20080912104117_01MSG_-on-gfesb03_20100104-14-34-26-498.hl7
000003_CTLID_20080912104117_02A___-on-gfesb03_20100104-14-34-26-749.hl7
000004_CTLID_2008091202842_01MSG_-on-gfesb02_20100104-14-34-30-544.hl7
000004_CTLID_2008091202842_02A___-on-gfesb02_20100104-14-34-30-664.hl7
000005_CTLID_2008090902313_01MSG_-on-gfesb03_20100104-14-34-41-043.hl7
000005_CTLID_2008090902313_02A___-on-gfesb03_20100104-14-34-41-244.hl7
000006_CTLID_20080910114132_01MSG_-on-gfesb02_20100104-14-34-48-711.hl7
000006_CTLID_20080910114132_02A___-on-gfesb02_20100104-14-34-48-777.hl7
```

If the sender fails, nothing much can be done about that fact, except to try to get a sender up and running as quickly as possible. If the sender runs in a regular clustered environment, that is configured to do so, it can be started on another node in the cluster and resume sending. Since this discussion is about horizontal scaling, load balancing (LB) and high availability (HA) of receivers, there will be no further discussion of senders, except as far as their configuration impacts load balancing and high availability of receivers.

Three types of solutions are used in the exercise – a HL7 BC-based, a Web Service-based (SOAP over HTTP) and a JMS-based. Each uses a different protocol and each behaves differently in the face of a failure of the receiver.

Each of the messaging streams is serialized at the sender in such a way that the sender will not send a new message until it received an acknowledgment for the previous message. This restriction is implemented to allow a human to observe messaging, load balancing, retries and fail-over in human time. This also allows a degree of control of when to crash a receiver host to simulate host failure. In a regular implementation, which does not care about message sequencing, this restriction would not be imposed.

To simulate time used for processing by a receiver, each receiver executes a BPEL Wait for a random duration of between 0 and 10 seconds. Only when the wait is over does a receiver generate and send an application acknowledgement. This allows the host on which the receiver runs to be crashed, and to interrupt processing so the acknowledgment to a message being processed at the time never gets delivered. Crashing a currently executing host explicitly induces retry and redelivery.

The HL7 BC-based and the Web Service-based solutions will use a load balancer to distribute workload between participating hosts and to ensure continuity of message delivery in the face of failing receivers. The JMS-based solution will not use the load balancer because the receivers are configured as “competing consumers”, an Enterprise Integration Patterns (EIP) pattern which describes multiple independent receivers which can process JMS messages from a single JMS Queue. A “natural” load balancing takes place anyway.

When message flow proceeds unimpeded, and the load balancer is configured to distribute requests in a round-robin manner, messages are received and processed by alternating receivers. Event in the case of JMS receivers, which don’t used the load balancer, messages are processed by alternating receivers.

What makes the solution highly available are the multiple receivers capable of processing messages on multiple independent hosts, and senders’ ability to recognize that a message failed to be received and their ability to retry delivery.

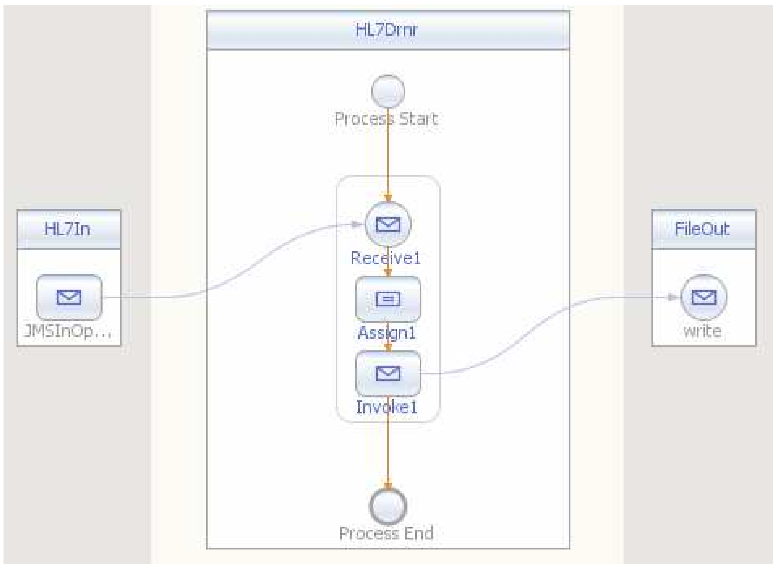
In all cases it is critical to ensure that the sender recognizes that the receiver crashed and to retry. The retry must be timely. The time taken to wait for a response before concluding that it is not going to come must not be so long that it unduly slows down message processing, and must not be so 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.

Discussion of specific solutions includes discussion of the timing parameters and other aspects of sender, receiver and load balancer configurations, which affect high availability and load balancing.

Overview of Drainer Projects

Each Drainer project receives messages from a JMS Queue and writes them into a specific file system directory, one file per message.

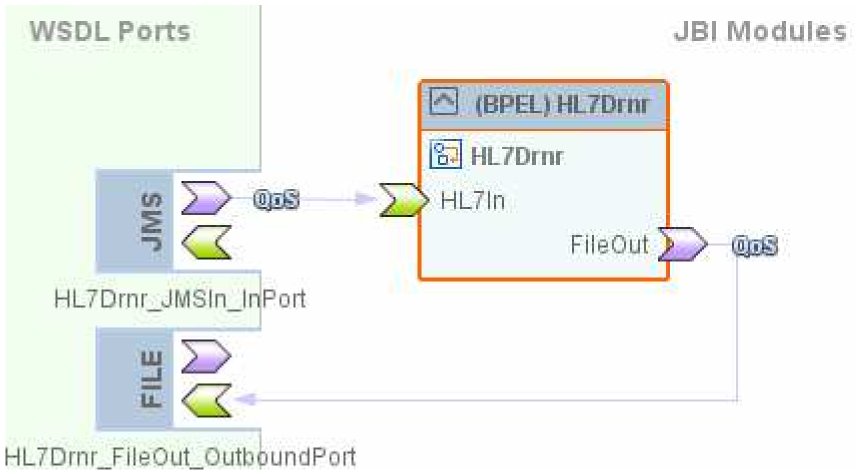
The BPEL process model, which implements the HL7 Drainer (HL7Drnr), is depicted below.



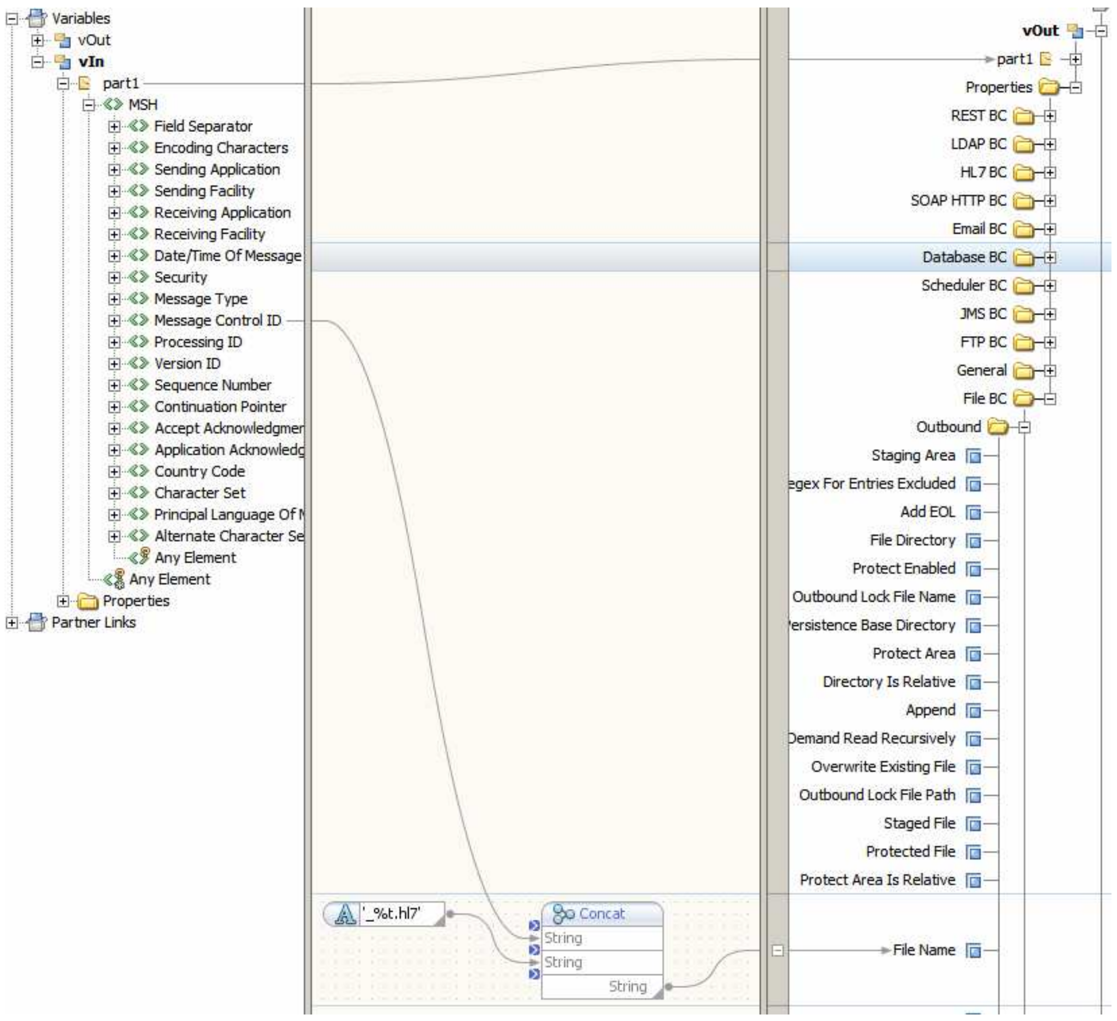
The JMS BC is configured as a one-way service receiving messages from a JMS Queue hosted on a shared JMS Server.

The File BC is configured to write each message to a separate file whose name is derived form the message id in the message.

The Composite Application Service Assembly below illustrates connectivity aspects of the solution.



The process 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.



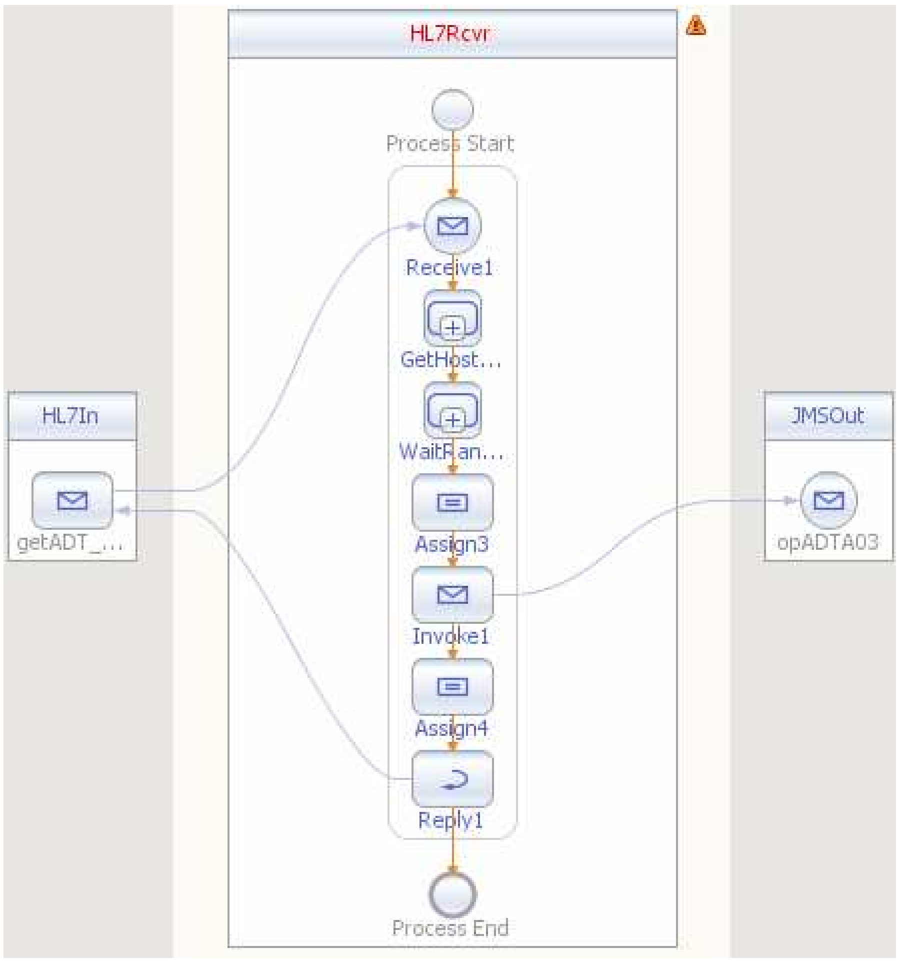
WSDrnr and JMSDrnr have the same structure and connectivity, though each writes files to a different file system directory.

All project sources are available for download at http://mediacast.sun.com/users/Michael.Czapski-Sun/media/GFESB_HA_LB_NetBeans_projects.zip/details.

Overview of Receiver Projects

Each Receiver receives messages from the binding component appropriate to the protocol it supports, obtains the name of the host on which the process instance runs, waits for a random period of up to 10 seconds, sends the message payload to the JMS Server, constructs a HL7 ACK message using data from the HL7 ADT A03 message it received, and sends the ACK as a response to the sender.

The BPEL process model which implements the HL7 Receiver (HL7Rcvr) is depicted below.

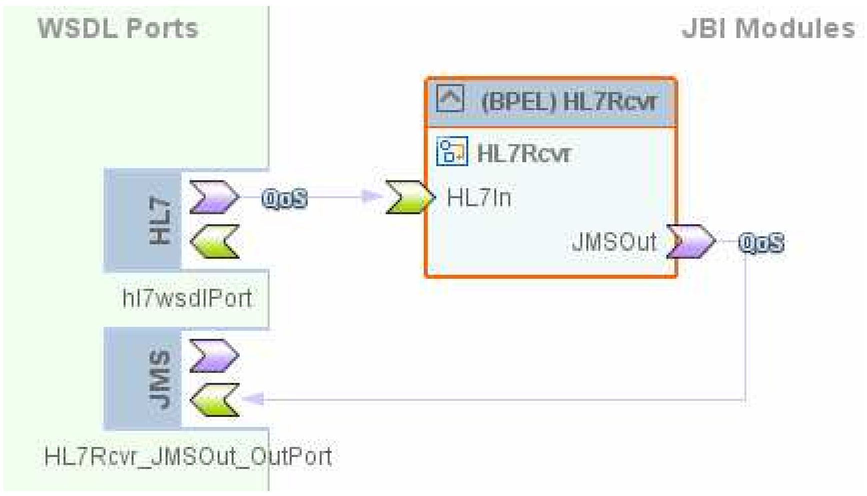


The “GetHostName” and “WaitRandTime” scopes are collapsed to de-clutter the diagram. Blog entries “GlassFish ESB v2.1 - Using JavaScript Codelets to Extend BPEL 2.0 Functionality” at http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_1_using and “GlassFish ESB v2.1 Field Notes - JavaScript Codelets to Make BPEL Process Wait for a Random Duration Up to a Maximum number of Milliseconds” at http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_1_field discuss how JavaScript and Java can be embedded in BPEL to provide required functionality.

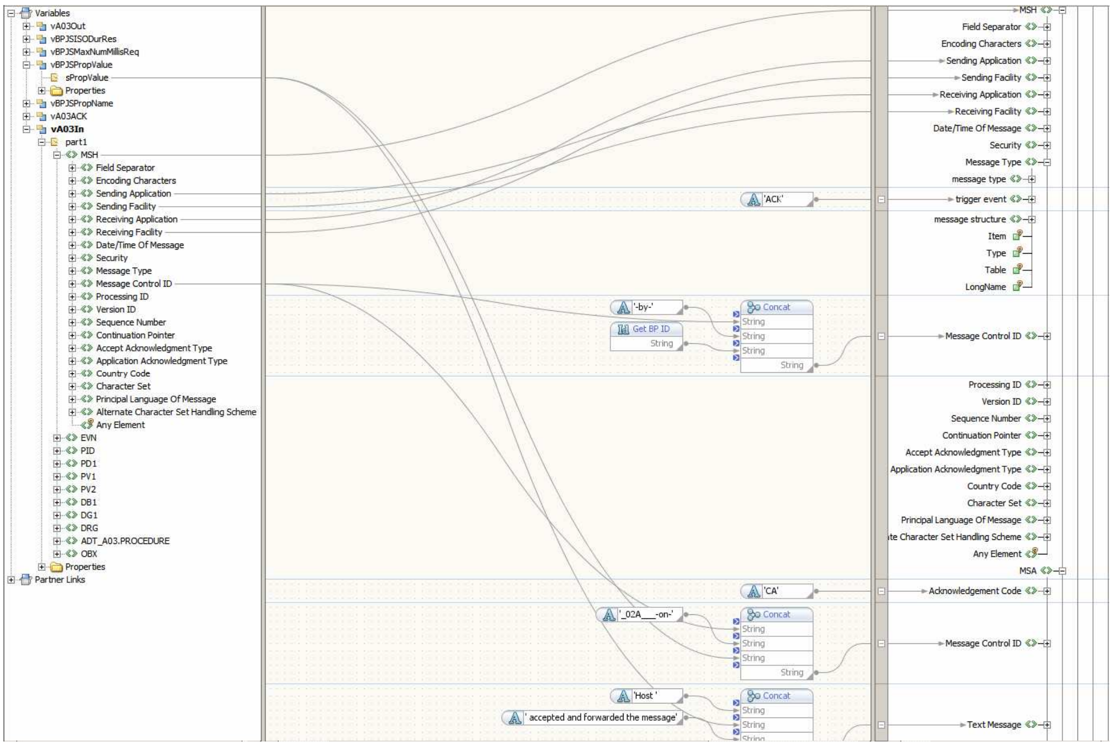
The HL7 BC is configured to receive HL7 ADT A03 request messages and to return HL7 ACK response messages.

The JMS BC is configured as a one-way service sending messages to a JMS Queue hosted on a shared JMS Server.

The Composite Application Service Assembly below illustrates connectivity aspects of the solution.



The process sends the entire HL7 BC payload to the JMS Server, and then constructs a HL7 ACK message using data from the HL7 ADT A03 message which it is handling. It also modifies the Message Control ID to embed the name of the host on which it is executing and to embed the “message type”.



The details in the illustration above are too small to see what is going on, however all project sources are available for download at http://mediacast.sun.com/users/Michael.Czapski-Sun/media/GFESB_HA_LB_NetBeans_projects.zip/details

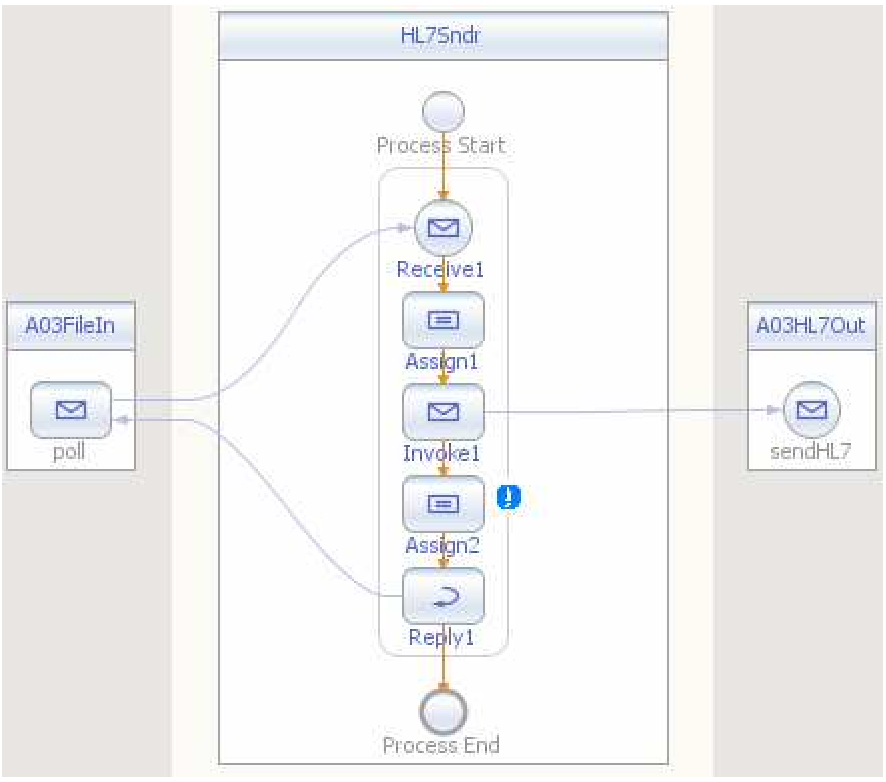
Both the JMSRcvr and the WSRcvr follow the same pattern as the HL7Rcvr, varying only the inbound Binding Component. Message processing is identical.

Overview of Sender Projects

Unlike the Drainer and the Receiver projects, which were virtually identical, the Sender projects vary somewhat in order to accommodate different recovery and redelivery handling capabilities of the sending binding components. All three protocols, HL7 MLLP, SOAP over HTTP and JMS are TCP based. How they deal with service disruption varies from binding component to binding component.

In general, a Sender reads a file containing multiple HL7 ADT A03 records and sends each as a separate message to the receiver either via a load balancer or via a JMS Server. Each sender waits for a response, the HL7 ACK message, and writes that ACK message to a separate file, whose name is derived from the message control ID of the ACK message.

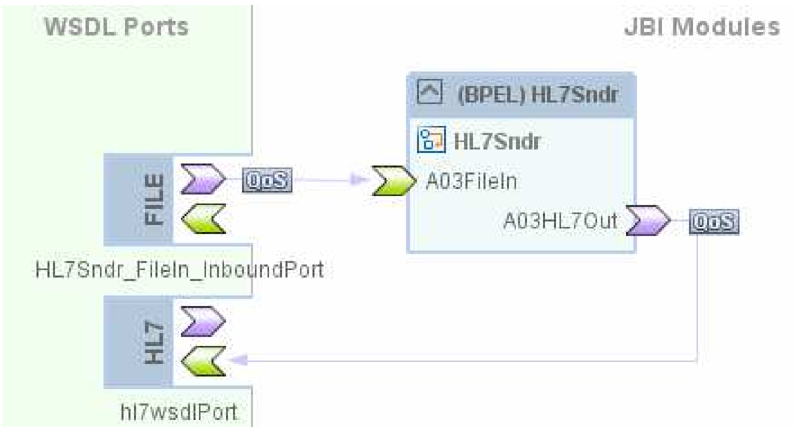
The HL7Sndr, discussed here, is the simplest of the three, and superficially not unlike a receiver.



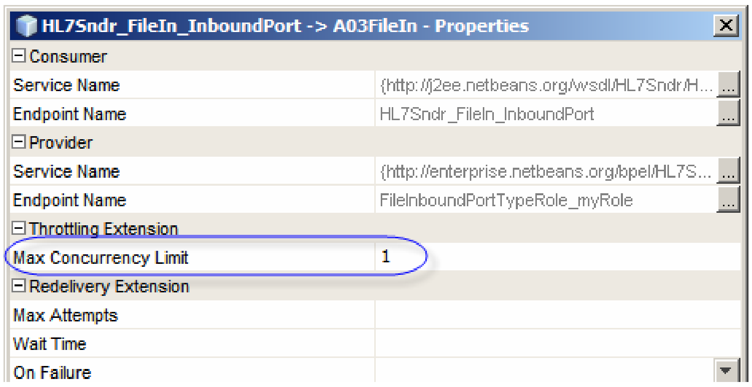
The HL7 BC is configured to send HL7 ADT A03 request messages and to receive HL7 ACK response messages.

The File BC is configured to poll a designated directory for files whose name matches a pattern and to write responses to the same directory.

The Composite Application Service Assembly below illustrates connectivity aspects of the solution.



Note the QoS icons indicating that QoS properties are changed from their defaults. The QoS properties on the File BC side are configured to serialize message handling. In conjunction with the sender waiting for an acknowledgement this causes one message at a time to be processed.

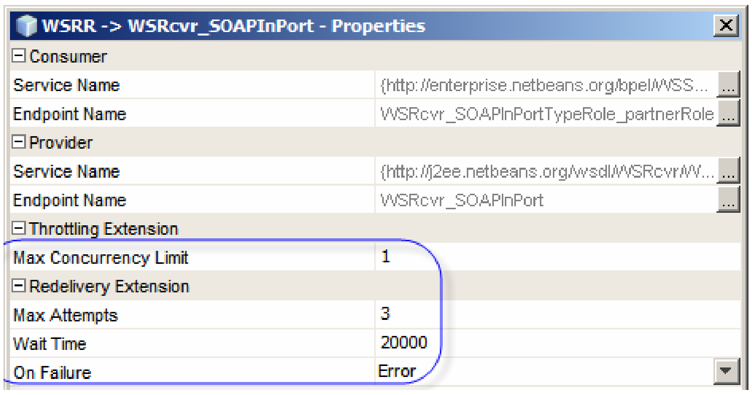

HL7Sndr_FileIn_InboundPort -> A03FileIn - Properties

⊟ Consumer	
Service Name	{http://j2ee.netbeans.org/wsdl/HL7Sndr/H...
Endpoint Name	HL7Sndr_FileIn_InboundPort
⊟ Provider	
Service Name	{http://enterprise.netbeans.org/bpel/HL7S...
Endpoint Name	FileInboundPortTypeRole_myRole
⊟ Throttling Extension	
Max Concurrency Limit	1
⊟ Redelivery Extension	
Max Attempts	
Wait Time	
On Failure	

This is the same for all senders regardless of protocol.

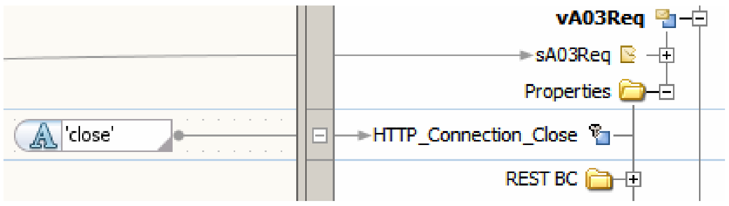
For the HL7 BC the QoS properties are configured to define timings and counts, affecting redelivery handling and discussed in detail in section “HL7 BC and Load Balancer Timing Parameters”. The HL7 BC, unlike other BCs, implements explicit, HL7 protocol mandated, redelivery handling logic so there is no need and no use configuring redelivery handling parameters through the CASA QoS properties.

WSSndr CASA QoS parameters fore the HTTP BC are set to provide serialization and to provide redeliver handling configuration.


WSRR -> WSRcvr_SOAPInPort - Properties

Consumer	
Service Name	{http://enterprise.netbeans.org/bpel/WSS...
Endpoint Name	WSRcvr_SOAPInPortTypeRole_partnerRole
Provider	
Service Name	{http://j2ee.netbeans.org/wsdl/WSRcvr/W...
Endpoint Name	WSRcvr_SOAPInPort
Throttling Extension	
Max Concurrency Limit	1
Redelivery Extension	
Max Attempts	3
Wait Time	20000
On Failure	Error

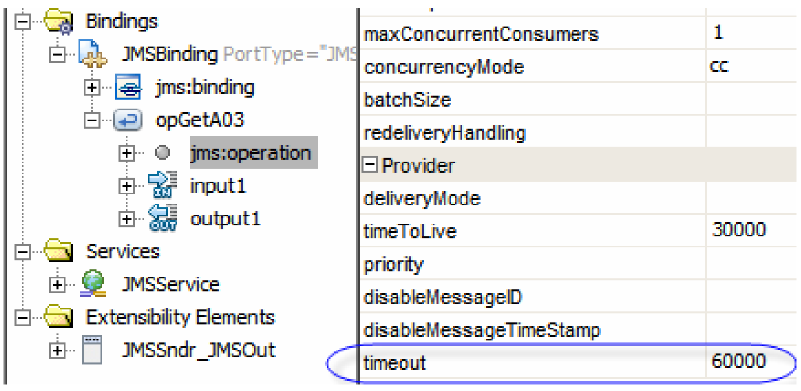
In addition, the WSSender BPEL process explicitly overrides the default HTTP “connection: keep-alive” header with “connection: close”, to ensure each request is issued over a different TCP session and to allow the load balancer to do its work.



Together with the redelivery handling parameters in the CASA QoS for the HTTP BC, this ensures that the lack of a timely acknowledgement is recognized by the BC, causes redelivery handling to be invoked, and the request to be re-issued.

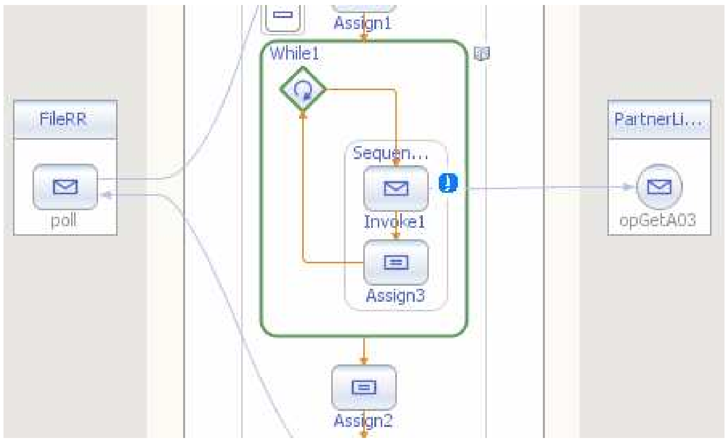
The HTTP BC timing parameters are further discussed in section “WS Projects”.

The JMSSndr is more complex because the response timeout exception is not propagated to the JMS BC in such a way that explicit redelivery handling functionality configured through the QoS parameters is invoked. The QoS parameters on the JMS BC side are not configured. Instead, the JMS BC returns an empty response message to the BPEL process once the timeout period, configured in the JMS BC WSDL extension, expires.



maxConcurrentConsumers	1
concurrencyMode	cc
batchSize	
redeliveryHandling	
Provider	
deliveryMode	
timeToLive	30000
priority	
disableMessageID	
disableMessageTimeStamp	
timeout	60000

Since an empty response does not trigger a fault, explicit logic is required to allow the JMSSndr to recognize the failure and explicitly resend the message a pre-configured number of times.



The JMS BC timing parameters are further discussed in section “JMS Projects”.

All project sources are available for download at http://mediacast.sun.com/users/Michael.Czapski-Sun/media/GFESB_HA_LB_NetBeans_projects.zip/details

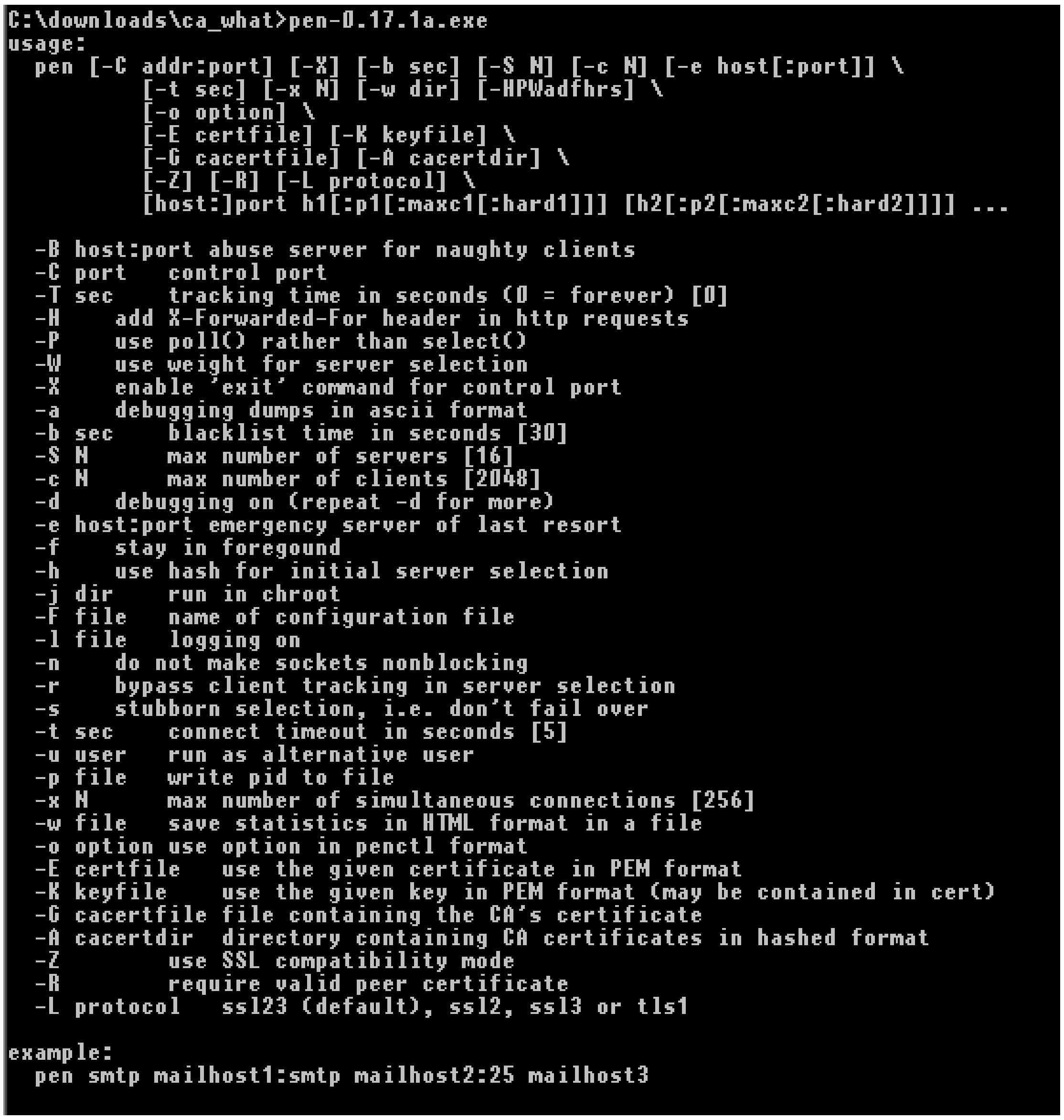
Load Balancer Preparation

For this exercise I use the VMware host, the machine in which the VMs run, as the load balancer host. The load balancer software I use is the PEN Load Balancer, obtainable from http://siag.nu/pen/. There are distributions for Windows, variety of Linux flavors and other popular Operating Systems.

Pen distribution for Windows can be downloaded from http://siag.nu/pub/pen/ - I use pen-0.17.1a.exe. It is a Windows executable compiled with Cygwin libraries. You may need cygwin.dll (http://www.cygwin.com/) to run it.

Whichever operating system you use, the command line options used to support load balancing discussed in this document are the same.

Here is a dump of the pen help message.



```
C:\downloads\ca_what>pen-0.17.1a.exe
usage:
  pen [-C addr:port] [-X] [-b sec] [-S N] [-c N] [-e host[:port]] \
          [-t sec] [-x N] [-w dir] [-HPWadfhrs] \
          [-o option] \
          [-E certfile] [-K keyfile] \
          [-G cacertfile] [-A cacertdir] \
          [-Z] [-R] [-L protocol] \
          [host:]port h1[:p1[:maxc1[:hard1]]] [h2[:p2[:maxc2[:hard2]]]] ...

  -B host:port abuse server for naughty clients
  -C port   control port
  -T sec    tracking time in seconds (0 = forever) [0]
  -H    add X-Forwarded-For header in http requests
  -P    use poll() rather than select()
  -W    use weight for server selection
  -X    enable 'exit' command for control port
  -a    debugging dumps in ascii format
  -b sec    blacklist time in seconds [30]
  -S N      max number of servers [16]
  -c N      max number of clients [2048]
  -d    debugging on (repeat -d for more)
  -e host:port emergency server of last resort
  -f    stay in foregound
  -h    use hash for initial server selection
  -j dir    run in chroot
  -F file   name of configuration file
  -l file   logging on
  -n    do not make sockets nonblocking
  -r    bypass client tracking in server selection
  -s    stubborn selection, i.e. don't fail over
  -t sec    connect timeout in seconds [5]
  -u user   run as alternative user
  -p file   write pid to file
  -x N      max number of simultaneous connections [256]
  -w file   save statistics in HTML format in a file
  -o option use option in penctl format
  -E certfile   use the given certificate in PEM format
  -K keyfile    use the given key in PEM format (may be contained in cert)
  -G cacertfile file containing the CA's certificate
  -A cacertdir  directory containing CA certificates in hashed format
  -Z        use SSL compatibility mode
  -R        require valid peer certificate
  -L protocol   ssl23 (default), ssl2, ssl3 or tls1

example:
  pen smtp mailhost1:smtp mailhost2:25 mailhost3
```

The pen load balancer can be run as a daemon/service and can support multiple configurations in the same instance. I chose to run separate instances of pen, in the foreground (-f), to clearly separate each load balancing configuration and to show message exchange (-a –ddddf) for each configuration separately.

The pen load balancer is only one of the critical components in the load balancing and high availability configuration which uses it. The sender and the receiver must also be correctly configured to achieve the desired effect. This is discussed in the sections pertaining to the JBI solutions themselves.

Load Balancer configuration for the HL7 BC-based Solution

The HL7 BC-based solution uses the following command line options for the load balancing configuration:

```
pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -ddddd -f -l
pen.log -p pen.pid -r -w pen.stats.html 34001 gfesb02
gfesb03
```

Critical options are:

-r	Use Round Robin load balancing algorithm
34001	Listen on port 34001 – this is configured in the HL7 BCs

Any tcp connection requests to the host running pen, to port 34001, will be redirected to hosts gfesb02 and gfesb03, to their port 34001.

All other options are not critical. Peruse the pen man page to see what they are and what they do.

Load Balancer configuration for the HTTP BC-based Solution

The HTTP BC-based solution uses the following command line options for the load balancing configuration:

```
pen-0.17.1a.exe -C 44001 -X -a -S 2 -d -ddddd -f -l
pen.log -p pen.pid -r -w pen.stats.html -t 5 -b 60 9080
gfesb02 gfesb03
```

Critical options are:

-r	Use Round Robin load balancing algorithm
-t 5	Time out connection request after 5 seconds
-b 60	Blacklist an inactive target for 60 seconds
9080	Listen on port 9080 for connection requests

Any tcp connection requests to the host running pen, to port 9080, will be redirected to hosts gfesb02 and gfesb03, to their port 9080.

Appliance Preparation

Following instructions in Blog entries “GlassFish ESB v2.x Field Notes - Preparing Basic JeOS Appliance for GlassFish ESB LB and HA Testing”, at http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_x_field and “GlassFish ESB v2.2 Field Notes - Installig GlassFish ESB on the Basic JeOS Appliance for LB and HA Testing” at http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_2_field, create three appliances whose host names are gfesb01, gfesb02 and gfesb03. When finished, the three appliances will be identical.

The host gfesb01 will be used for the shared JMS. Hosts gfesb02 and gfesb03 will be a part of a heterogeneous non-Cluster. To provide a collective name for both we will use an alias hostname of ghesbha. Add this alias, pointing to the host on which the load balancer will be running, to the /etc/hosts file on gfesb01, which will be connecting to the non-cluster via the load balancer.

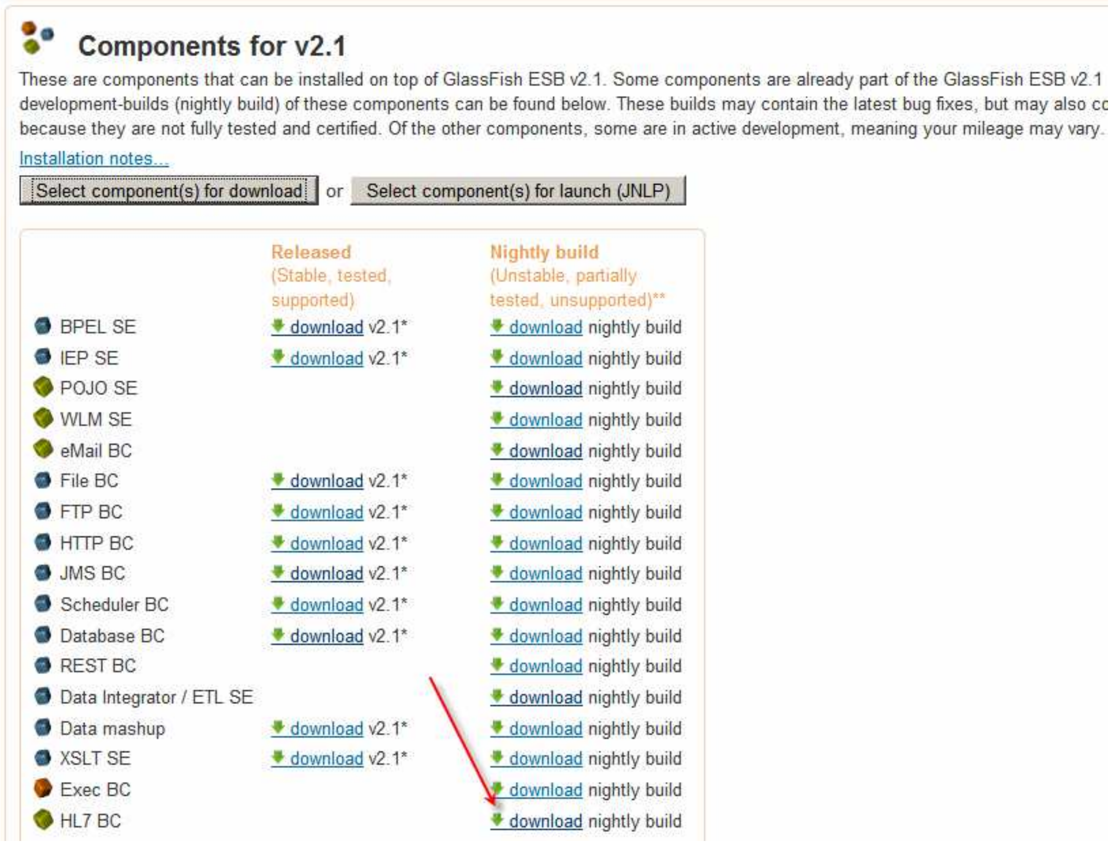
My host environment is based on a Toshiba Tecra M5 with 3.2Gb of real memory, running Windows XP, SP3. To run three virtual machines on that platform requires careful management of physical memory. I reconfigured the gfesb01 appliance to use 1280Mb of memory (it has 6 projects deployed), and gfesb02 and gfesb03 to use 640Mb of memory each (each runs three projects). Between them the three VMs use

2.5Gb of real host memory, leaving just enough for the host OS to run, with the few essential applications used to interact with the vm environment.

Appliance Customization

Some of the GlassFish ESB projects, which will be deployed to the GalssFish ESB appliances, will process HL7 v2 messages using the HL7 Binding Component. To facilitate that, GlassFish ESB runtime in each of the appliances must have the HL7 Binding Component installed. The basic appliance prepared so far has the GlassFish ESB v2.2 standard runtime. This runtime does not have the HL7 BC.

Download the HL7 BC from https://open-esb.dev.java.net/Downloads.html. To do so, navigate to the site, click the "Select component(s) for download" button and click the "download" link corresponding to the HL7 BC (make sure it is from the v2.1 distribution).



Components for v2.1

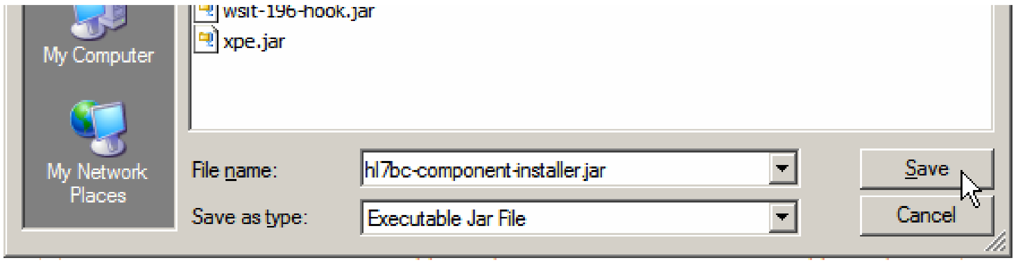
These are components that can be installed on top of GlassFish ESB v2.1. Some components are already part of the GlassFish ESB v2.1 development-builds (nightly build) of these components can be found below. These builds may contain the latest bug fixes, but may also cc because they are not fully tested and certified. Of the other components, some are in active development, meaning your mileage may vary.

Installation notes...

Select component(s) for download or Select component(s) for launch (JNLP)

	Released (Stable, tested, supported)	Nightly build (Unstable, partially tested, unsupported)**
BPEL SE	download v2.1*	download nightly build
IEP SE	download v2.1*	download nightly build
POJO SE		download nightly build
WLM SE		download nightly build
eMail BC		download nightly build
File BC	download v2.1*	download nightly build
FTP BC	download v2.1*	download nightly build
HTTP BC	download v2.1*	download nightly build
JMS BC	download v2.1*	download nightly build
Scheduler BC	download v2.1*	download nightly build
Database BC	download v2.1*	download nightly build
REST BC		download nightly build
Data Integrator / ETL SE		download nightly build
Data mashup	download v2.1*	download nightly build
XSLT SE	download v2.1*	download nightly build
Exec BC		download nightly build
HL7 BC		download nightly build

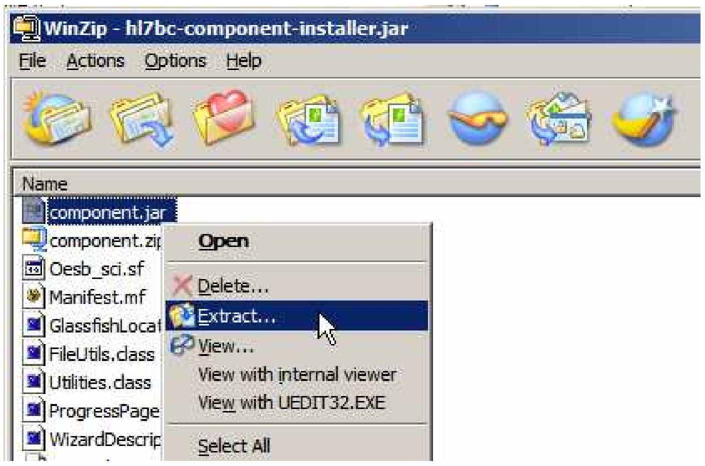
Save the hl7-component-installer.jar to a convenient location.



File name: hl7bc-component-installer.jar

Save as type: Executable Jar File

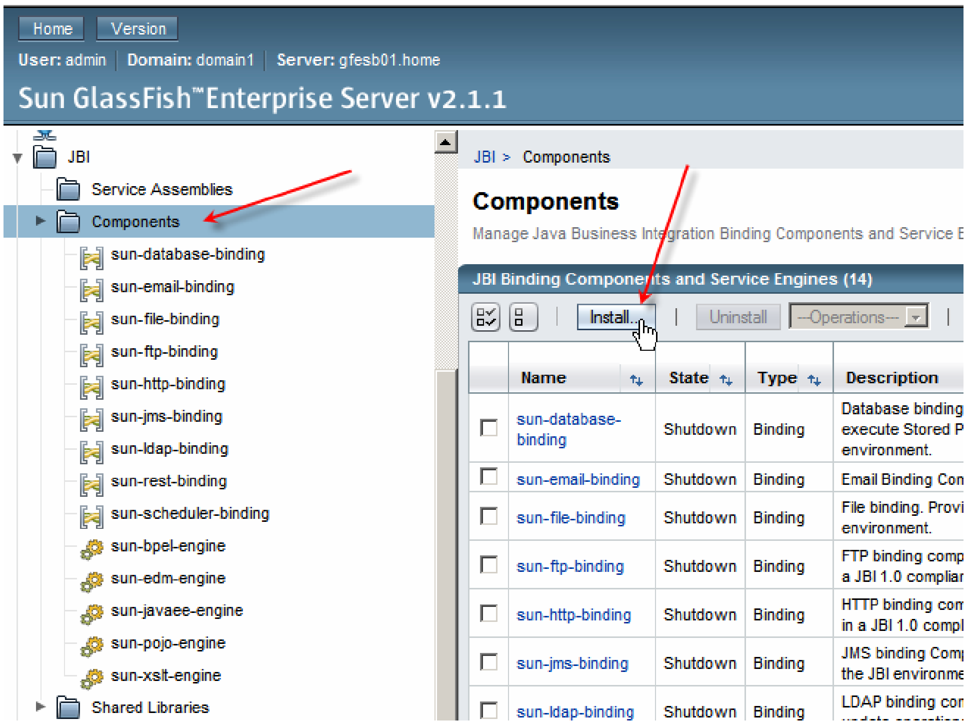
Extract the JAR named component.jar from the hl7-component-installer.jar.



To remember what it is, rename component.jar to hl7_component.jar.

Install the HL7 BC on each of the three virtual machines – for each virtual machine:

1. Make sure the Virtual machine is started and the GlaassFsih ESB instance is running (as it should since the machine is configured to start it at boot)
2. Start the GalssFish Applicatin Server Admin Console on the GlassFish ESB instance.
3. Expand the JBI node in the node tree, click on the Components node and click on the Install button on the right hand window



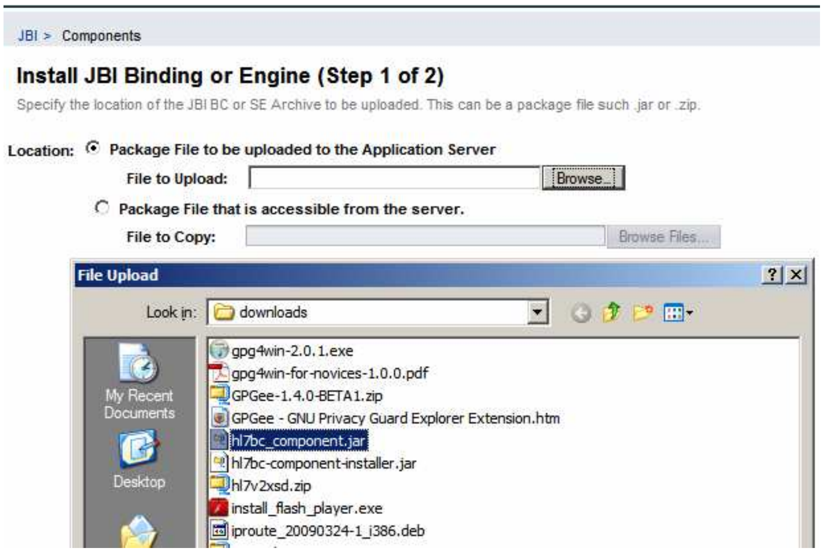
Components

Manage Java Business Integration Binding Components and Service E

JBI Binding Components and Service Engines (14)

Name	State	Type	Description
sun-database-binding	Shutdown	Binding	Database binding execute Stored P environment.
sun-email-binding	Shutdown	Binding	Email Binding Con
sun-file-binding	Shutdown	Binding	File binding. Provi environment.
sun-ftp-binding	Shutdown	Binding	FTP binding comp a JBI 1.0 compliar
sun-http-binding	Shutdown	Binding	HTTP binding com in a JBI 1.0 compl
sun-jms-binding	Shutdown	Binding	JMS binding Comp the JBI environme
sun-ldap-binding	Shutdown	Binding	LDAP binding cor

4. Click on the Browse button, locate and select the hl7_component.jar archive (previously extracted from hl7-component-installer.jar and renamed form component.jar)



JBI > Components

Install JBI Binding or Engine (Step 1 of 2)

Specify the location of the JBI BC or SE Archive to be uploaded. This can be a package file such .jar or .zip.

Location: (●) Package File to be uploaded to the Application Server

File to Upload: [ ] Browse...

( ) Package File that is accessible from the server.

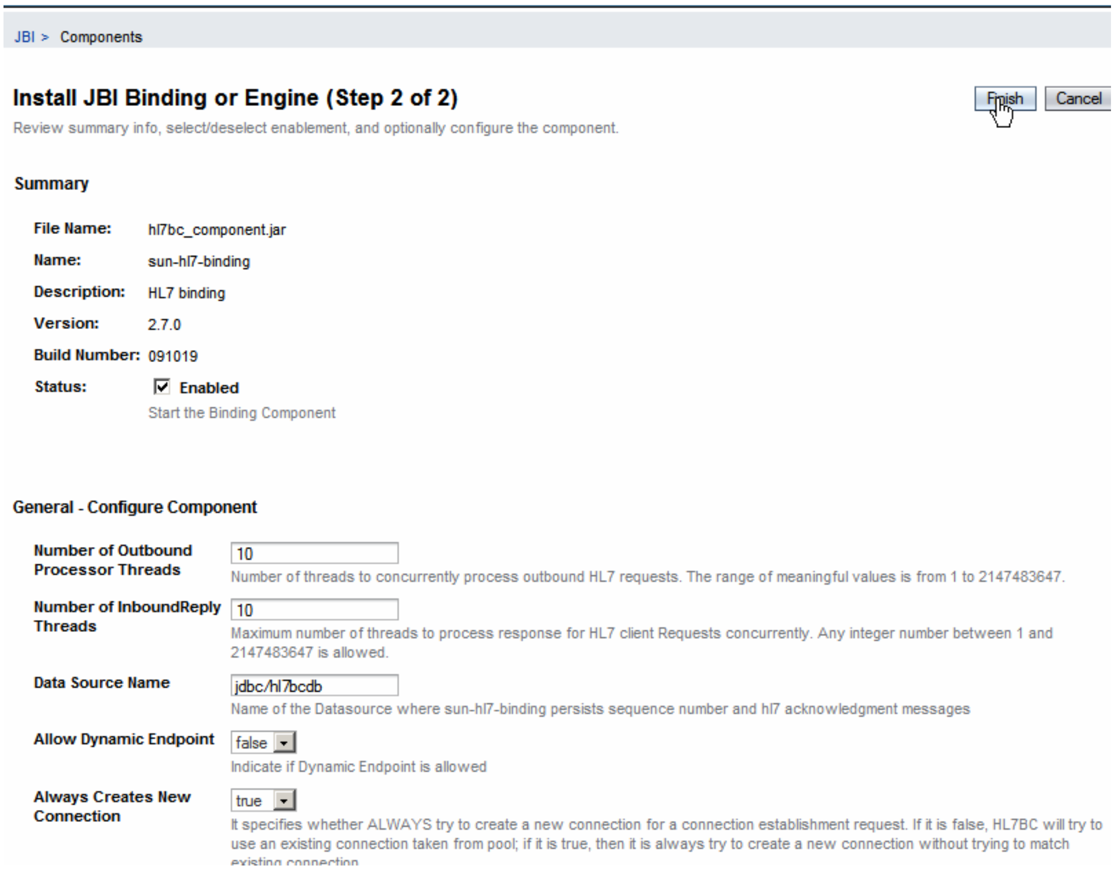
File to Copy: [ ] Browse Files...

File Upload

Look in: downloads

- gpg4win-2.0.1.exe
- gpg4win-for-novices-1.0.0.pdf
- GPGee-1.4.0-BETA1.zip
- GPGee - GNU Privacy Guard Explorer Extension.htm
- hl7bc_component.jar
- hl7bc-component-installer.jar
- hl7v2xsd.zip
- install_flash_player.exe
- iproute_20090324-1_i386.deb

5. Click Next
6. Accept all default values by clicking Finish



JBI > Components

Install JBI Binding or Engine (Step 2 of 2)

Review summary info, select/deselect enablement, and optionally configure the component.

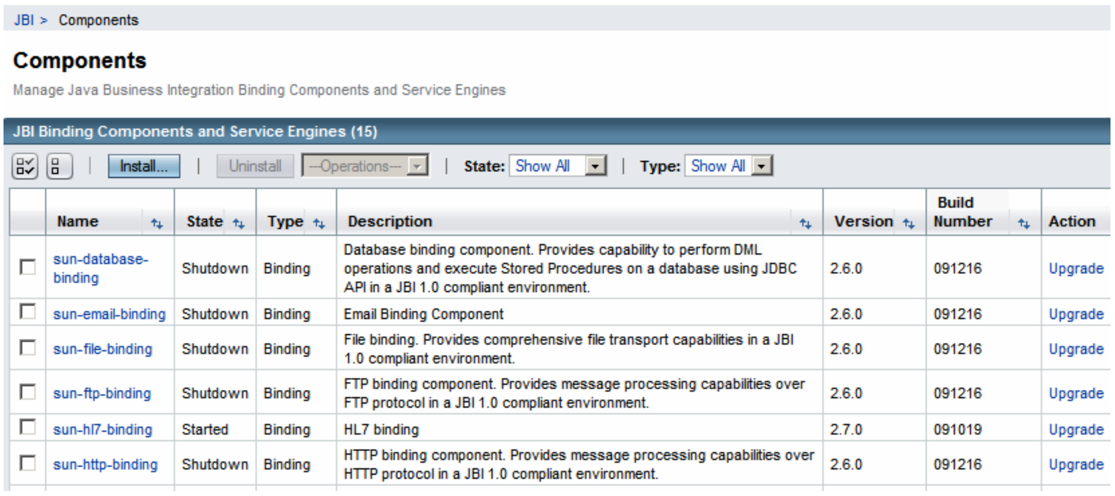
Summary

File Name:	hl7bc_component.jar
Name:	sun-hl7-binding
Description:	HL7 binding
Version:	2.7.0
Build Number:	091019
Status:	☑ Enabled Start the Binding Component

General - Configure Component

Number of Outbound Processor Threads	10 Number of threads to concurrently process outbound HL7 requests. The range of meaningful values is from 1 to 2147483647.
Number of InboundReply Threads	10 Maximum number of threads to process response for HL7 client Requests concurrently. Any integer number between 1 and 2147483647 is allowed.
Data Source Name	jdbc/hl7bcdb Name of the Datasource where sun-hl7-binding persists sequence number and hl7 acknowledgment messages
Allow Dynamic Endpoint	false Indicate if Dynamic Endpoint is allowed
Always Creates New Connection	true It specifies whether ALWAYS try to create a new connection for a connection establishment request. If it is false, HL7BC will try to use an existing connection taken from pool; if it is true, then it is always try to create a new connection without trying to match existing connection

The HL7 Binding Component should now be installed and ready to use.



Components

Manage Java Business Integration Binding Components and Service Engines

JBI Binding Components and Service Engines (15)

Install... Uninstall --Operations-- State: Show All Type: Show All

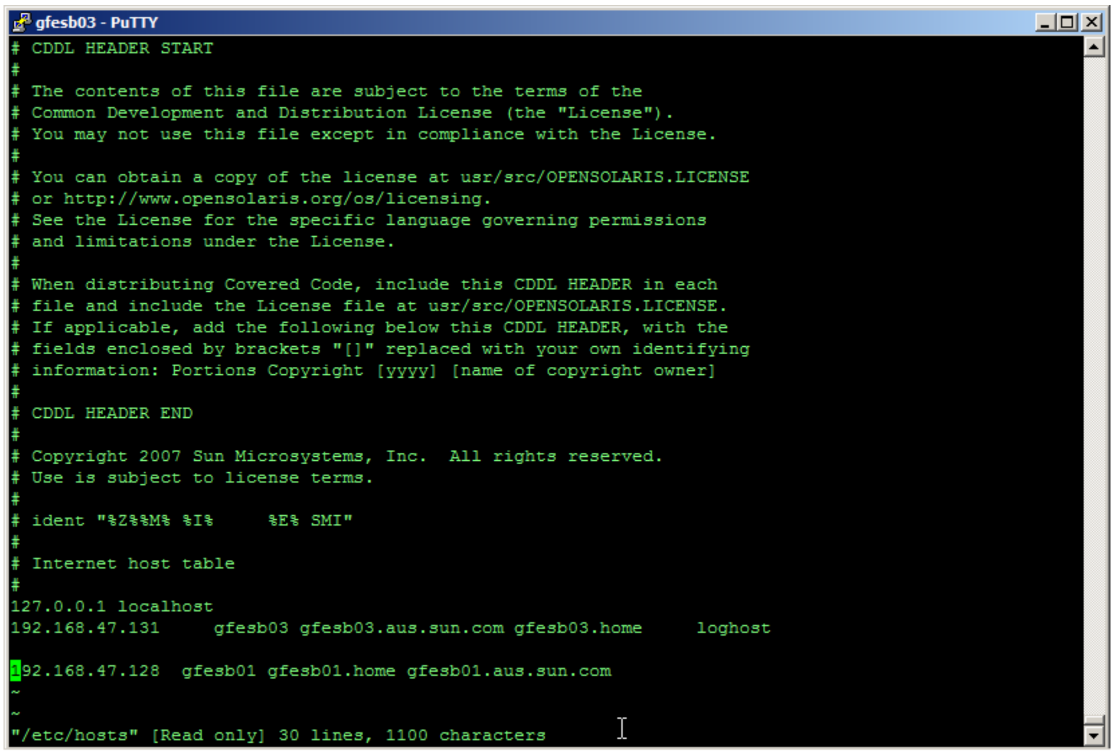
	Name	State	Type	Description	Version	Build Number	Action
☐	sun-database-binding	Shutdown	Binding	Database binding component. Provides capability to perform DML operations and execute Stored Procedures on a database using JDBC API in a JBI 1.0 compliant environment.	2.6.0	091216	Upgrade
☐	sun-email-binding	Shutdown	Binding	Email Binding Component	2.6.0	091216	Upgrade
☐	sun-file-binding	Shutdown	Binding	File binding. Provides comprehensive file transport capabilities in a JBI 1.0 compliant environment.	2.6.0	091216	Upgrade
☐	sun-ftp-binding	Shutdown	Binding	FTP binding component. Provides message processing capabilities over FTP protocol in a JBI 1.0 compliant environment.	2.6.0	091216	Upgrade
☐	sun-hl7-binding	Started	Binding	HL7 binding	2.7.0	091019	Upgrade
☐	sun-http-binding	Shutdown	Binding	HTTP binding component. Provides message processing capabilities over HTTP protocol in a JBI 1.0 compliant environment.	2.6.0	091216	Upgrade

Remember to do this for all three hosts, gfesb01, GTFESB02 and gfesb03.

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 gfesb02 and gfesb03 edit /etc/hosts and add an address mapping entry for gfesb01:



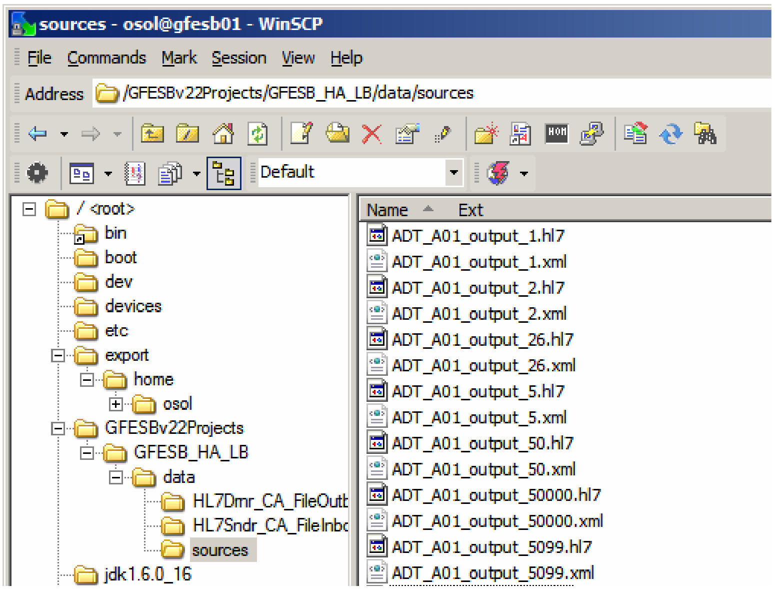
```
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#
# Internet host table
#
127.0.0.1 localhost
192.168.47.131     gfesb03 gfesb03.aus.sun.com gfesb03.home     loghost

192.168.47.128  gfesb01 gfesb01.home gfesb01.aus.sun.com
~
~
"/etc/hosts" [Read only] 30 lines, 1100 characters
```

On gfesb01 also add an address mapping entry for name gfesbha corresponding to the address of the host on which the load balancer will be run.

On gfesb01 create the following directory hierarchy, making sure that it is owned bu user osol and group staff:

`/GFESBv22Projects/GFESB_HA_LB/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_messages_sources.zip. This archive can be obtained from http://mediacast.sun.com/users/Michael.Czapski-Sun/media/HL7_messages_sources.zip/details.



Only ADT_A03_output_*nnnnn*.hl7 files will be used. Keep or delete other files as you see fit. One or more of the files in the `data` directory will be used to exercise the HL7xxxx solution.

On gfesb01 create the following directory hierarchy:

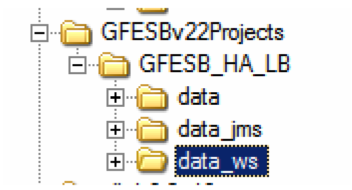
`/GFESBv22Projects/GFESB_HA_LB/data_jms`

From the `data/sources` directory copy ADT_A03_output_*nnnnn*.hl7 files to the data_jms directory. These will be used to exercise the JMSxxxx solution.

On gfesb01 create the following directory hierarchy:

`/GFESBv22Projects/GFESB_HA_LB/data_ws`

From the `data/sources` directory copy ADT_A03_output_*nnnnn*.hl7 files to the data_ws directory. These will be used to exercise the WSxxxx solution.



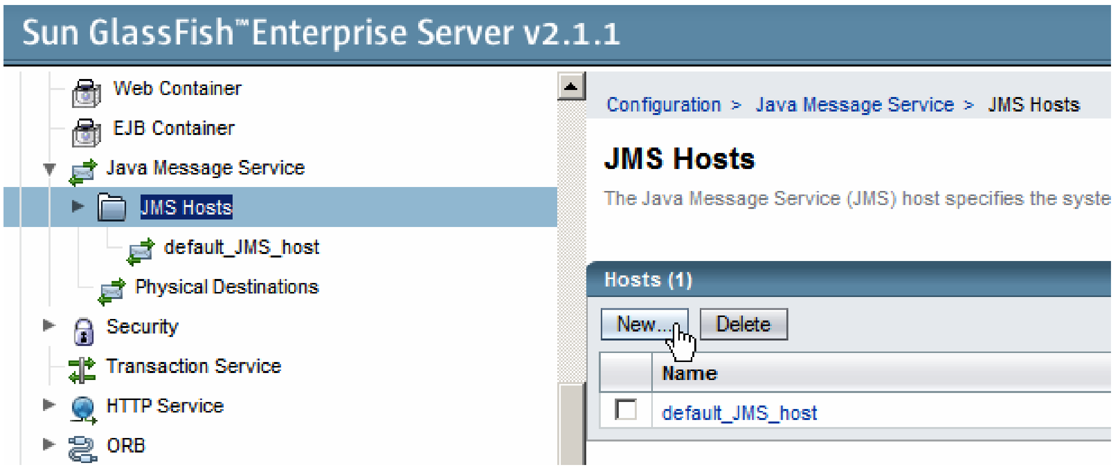
Shared Resources Configuration

All receivers will send messages to a shared, highly-available JMS infrastructure. To address the same infrastructure, each of the two Application Servers, on gfesb02 and gfesb03, must have a JMS Connection Pool which points to that shared JMS infrastructure.

Strictly speaking creation of a connection pool is not necessary. Each sender to JMS and receiver from JMS could be simply configured with the URL of the form mq://jmshost:jmsport, however this would require changes to the WSDL each time the host or the port was changed, with build and deploy for each affected project. By providing a JNDI-referenced pool we externalize these things to the Application Server so the service assembly can be deployed to any application server with the right pool.

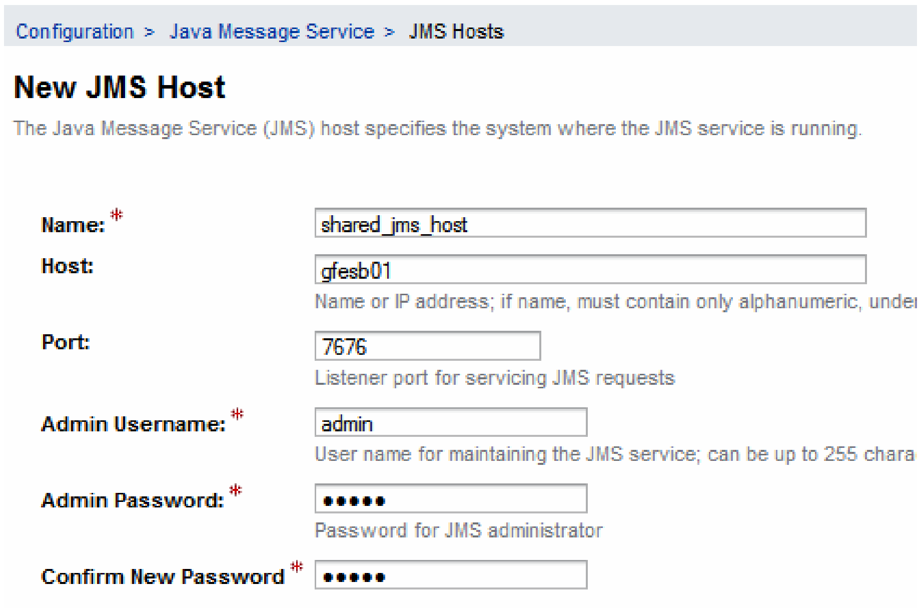
On each of gfesb02 and gfesb03, configure the resources as follows:

1. Create a New JMS Host – Configuration → Java Message Service →JMS Hosts



The Java Message Service (JMS) host specifies the syste

   a. Name: **shared_jms_host**
   b. Host: **gfesb01**
   c. Port: **7676** (or whatever port the host was configured to use for JMS – check that by looking at configuration of the default_JMS_host on gfesb01).



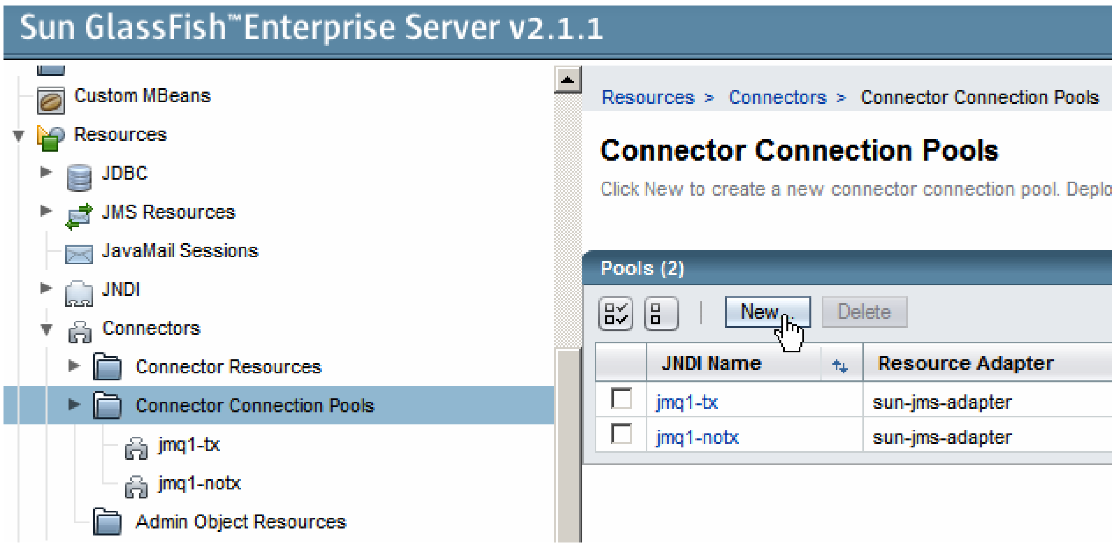
Configuration > Java Message Service > JMS Hosts

New JMS Host

The Java Message Service (JMS) host specifies the system where the JMS service is running.

Name: *	shared_jms_host
Host:	gfesb01 Name or IP address; if name, must contain only alphanumeric, undei
Port:	7676 Listener port for servicing JMS requests
Admin Username: *	admin User name for maintaining the JMS service; can be up to 255 chara
Admin Password: *	••••• Password for JMS administrator
Confirm New Password *	•••••

2. Create a New Connection Pool – Resources → Connectors → Connector Connection Pools



Sun GlassFish™ Enterprise Server v2.1.1

Resources > Connectors > Connector Connection Pools

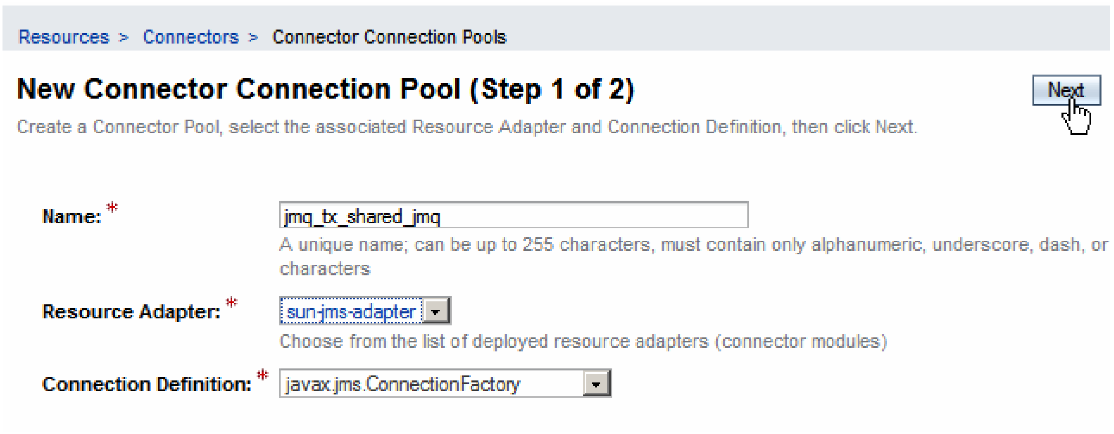
Connector Connection Pools

Click New to create a new connector connection pool. Deplo

Pools (2)

	JNDI Name	Resource Adapter
☐	jmq1-tx	sun-jms-adapter
☐	jmq1-notx	sun-jms-adapter

a. Name: **jmq_tx_shared_jmq**
b. Resource Adapter: sun-jms-adapter
c. Connection Definition: javax.jms.ConnectionFactory
d. Click Next



Resources > Connectors > Connector Connection Pools

New Connector Connection Pool (Step 1 of 2)

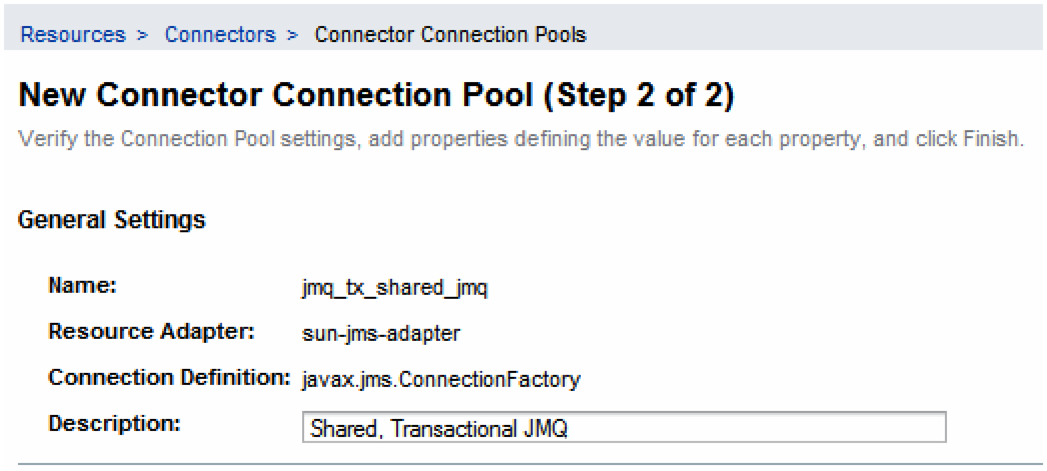
Create a Connector Pool, select the associated Resource Adapter and Connection Definition, then click Next.

Name: * jmq_tx_shared_jmq
A unique name; can be up to 255 characters, must contain only alphanumeric, underscore, dash, or characters

Resource Adapter: * sun-jms-adapter
Choose from the list of deployed resource adapters (connector modules)

Connection Definition: * javax.jms.ConnectionFactory

e. Description: Shared**, Transactional JMQ**



Resources > Connectors > Connector Connection Pools

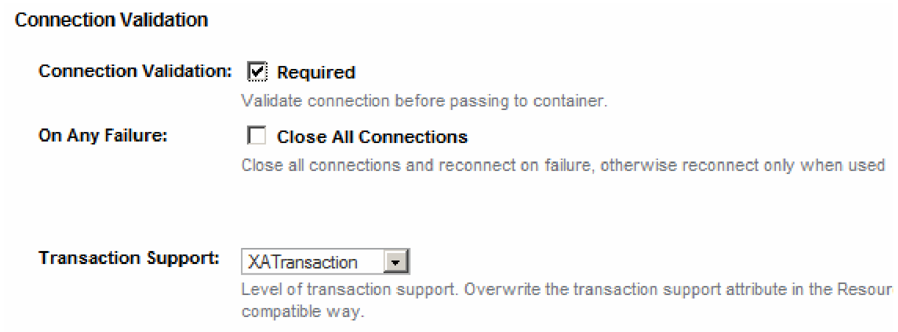
New Connector Connection Pool (Step 2 of 2)

Verify the Connection Pool settings, add properties defining the value for each property, and click Finish.

General Settings

Name:	jmq_tx_shared_jmq
Resource Adapter:	sun-jms-adapter
Connection Definition:	javax.jms.ConnectionFactory
Description:	Shared, Transactional JMQ

f. Connection Validation: **Required**
g. Transaction Support: **XA Transaction**



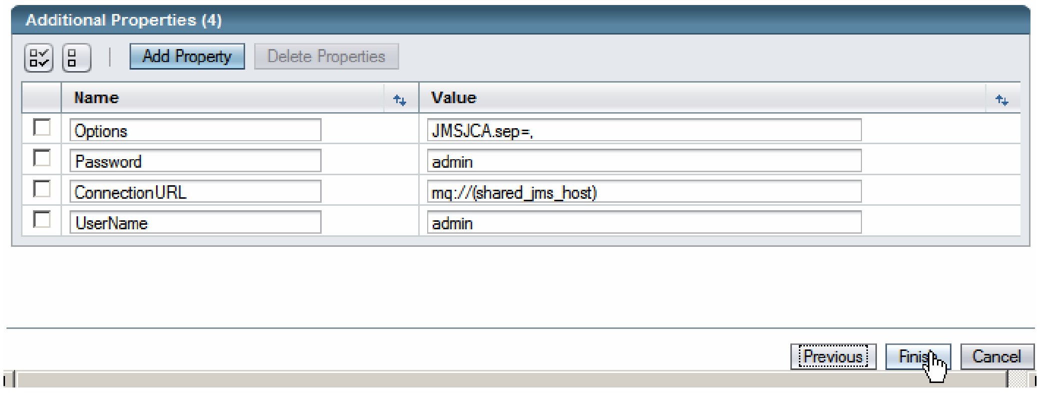
Connection Validation

Connection Validation: ☑ Required
Validate connection before passing to container.

On Any Failure: ☐ Close All Connections
Close all connections and reconnect on failure, otherwise reconnect only when used

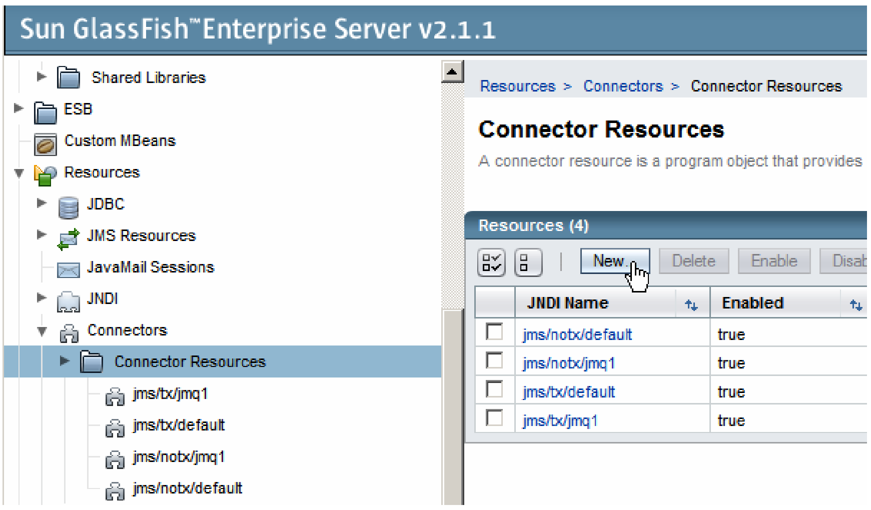
Transaction Support: XATransaction
Level of transaction support. Overwrite the transaction support attribute in the Resour compatible way.

h. Additional Properties:
   i. Options: JMSJCA.sep=,
   ii. Password: admin
   iii. ConnectionURL: mq://(shared_jms_host)
   iv. UserName: admin



Name	Value
Options	JMSJCA.sep=,
Password	admin
ConnectionURL	mq://{shared_jms_host}
UserName	admin

3. Create a New Connector Resource – Resources → Connectors → Connector Resources:

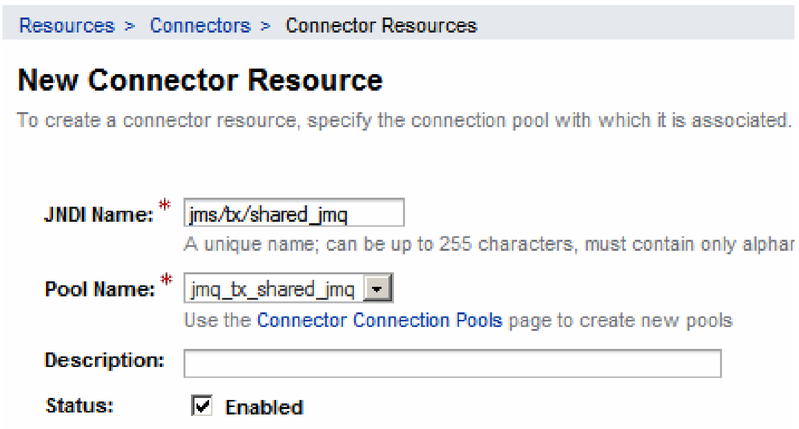


Connector Resources

A connector resource is a program object that provides

JNDI Name	Enabled
jms/notx/default	true
jms/notx/jmq1	true
jms/tx/default	true
jms/tx/jmq1	true

a. Name: **jms/tx/shared_jmq**
b. Pool Name: **jmq_tx_shared_jmq**
c. Status: Enabled
d. Click OK



New Connector Resource

To create a connector resource, specify the connection pool with which it is associated.

JNDI Name: * jms/tx/shared_jmq
A unique name; can be up to 255 characters, must contain only alphar

Pool Name: * jmq_tx_shared_jmq
Use the Connector Connection Pools page to create new pools

Description:

Status: ☑ Enabled

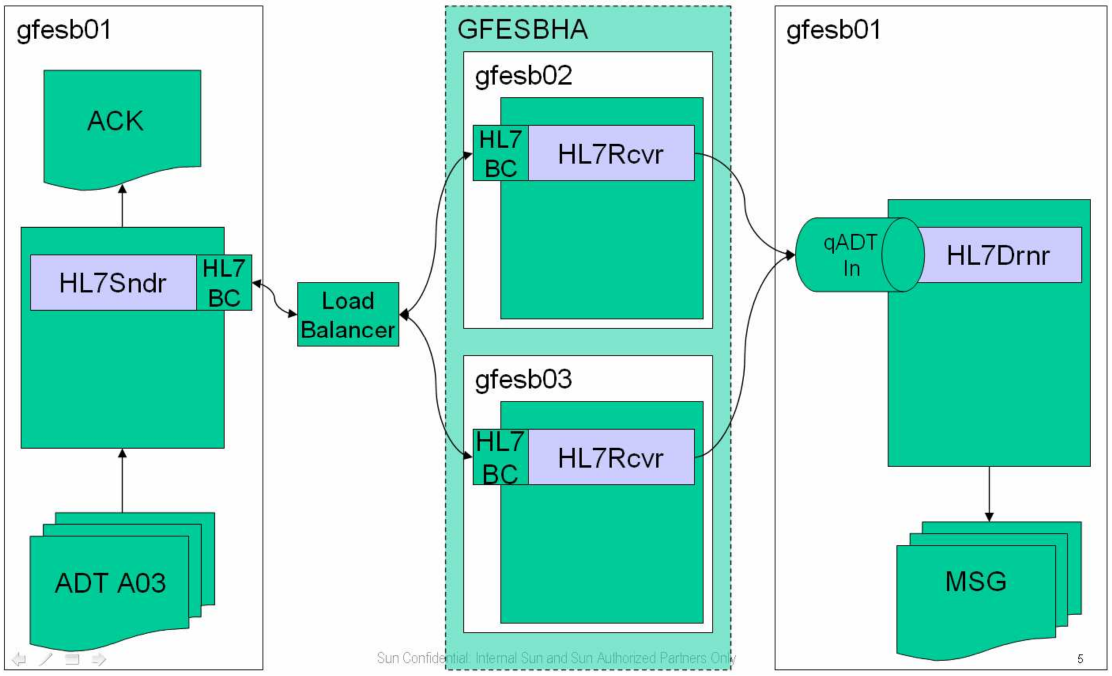
These resources will be used in receiver projects to identify the shared JMS infrastructure and to interact with it.

Project Deployment

Archive GFESB_HA_LB_deployables_SAs.zip, at http://mediacast.sun.com/users/Michael.Czapski-Sun/media/GFESB_HA_LB_deployables_SAs.zip/details, contains the ready-to-deploy service assemblies for all projects used in the exercise. Download the archive and unzip it to a convenient directory on a host that has a modern web browser and connectivity to the virtual appliances gfesb01, gfesb02 and gfesb03.

HL7 Projects

Projects HL7Sndr_CA and HL7Drnr_CA will be deployed to gfesb01. Project HL7Rcvr_CA will be deployed to gfesb02 and gfesb03.



There are at least three different ways to deploy projects to a remote server. The two discussed below are using the NetBeans tooling on the development machine to deploy to remote instances of the GlassFish Application Server and using the GlassFish Application Server Admin Console on each of the remote GlassFish instances.

If all the virtual machines and the development environment are co-located on the same physical host the memory requirements of all components are additive. On my Toshiba Tecra M5 with 3.2Gb of useable memory I could not run both the NetBeans IDE and the three VMs all at once. I resorted to starting two out of three VMs, deploying HL7Drnr to gfesb01 and HL7Rcvr to one of the gfesb02 or gfesb03, then shutting gfesb02/03 down, starting the other and deploying to it, then deploying HL7Sndr to gfesb01. Once deployments were completed I shut down NetBeans to recover the physical memory I needed to run the three VMs concurrently.

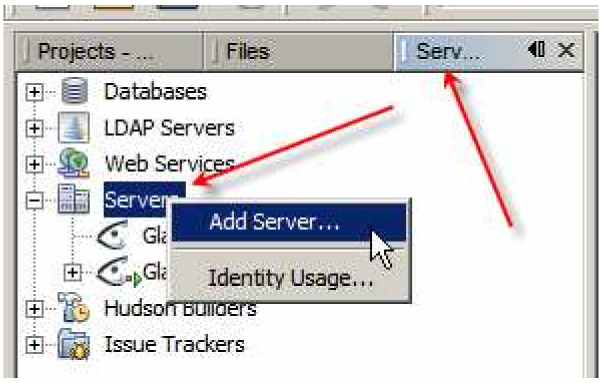
Using the GlassFish Application Server Admin Console is somewhat more cumbersome, in that one has to start the Console individually on each VM, but has the advantage of smaller memory footprint. I was able to deploy components while all three VMs were running.

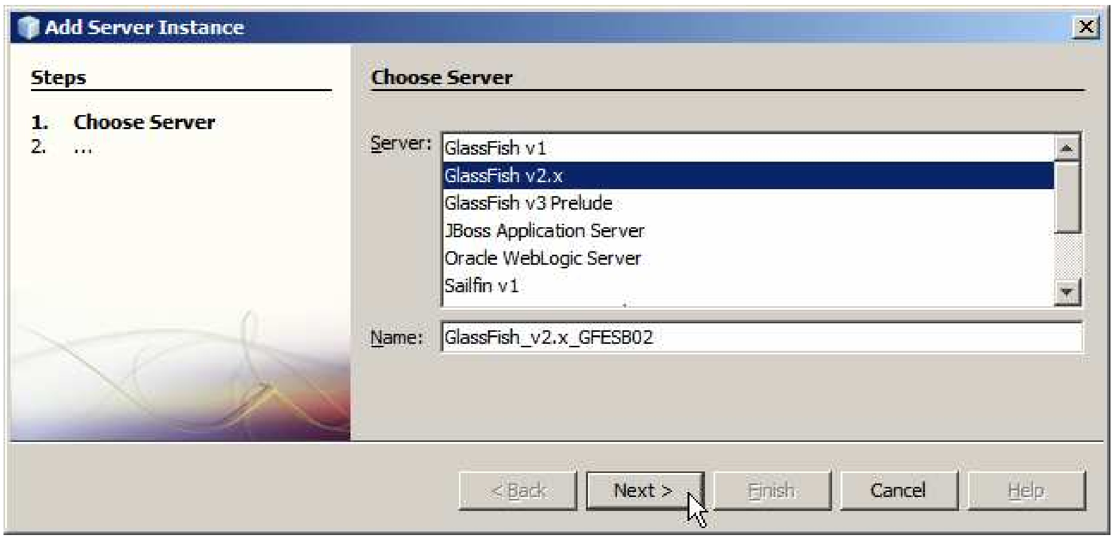
You will choose the method that is best for you.

Using the NetBeans IDE

If using NetBeans to deploy Service Assemblies it is necessary to add each of the target servers to the Servers list in NetBeans' Services tab.

Repeat the steps below for gfesb01, gfesb02 and gfesb03.




Add Server Instance

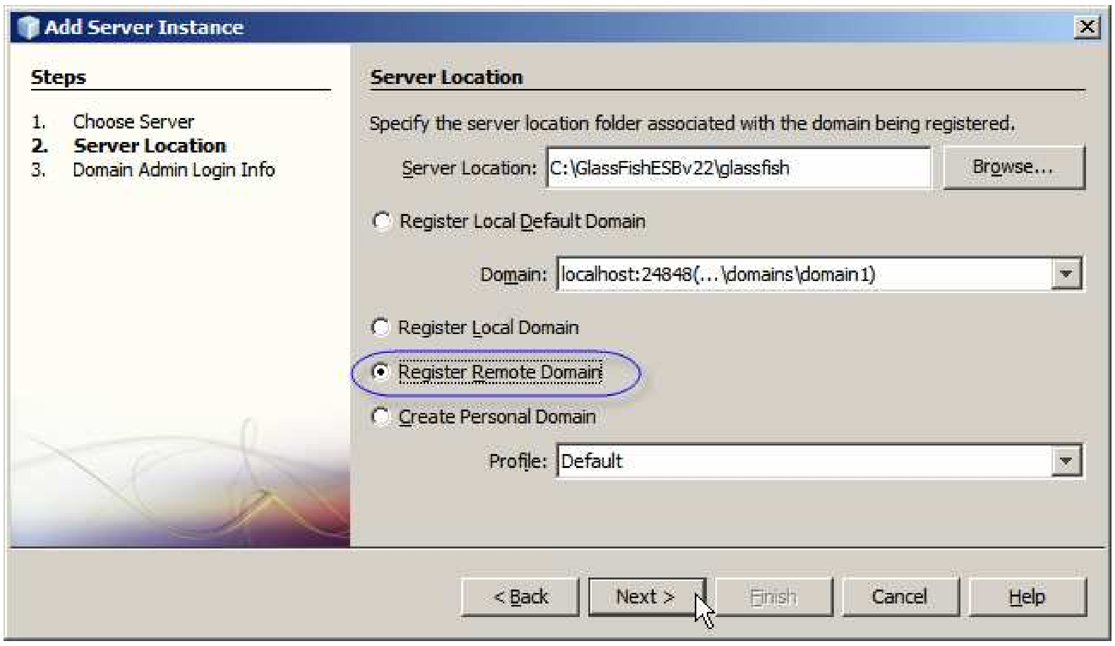
Steps

1. Choose Server
2. ...

Choose Server

Server:
GlassFish v1
GlassFish v2.x
GlassFish v3 Prelude
JBoss Application Server
Oracle WebLogic Server
Sailfin v1

Name: GlassFish_v2.x_GFESB02


Add Server Instance

Steps
1. Choose Server
2. **Server Location**
3. Domain Admin Login Info

Server Location

Specify the server location folder associated with the domain being registered.

Server Location: C:\GlassFishESBv22\glassfish

( ) Register Local Default Domain

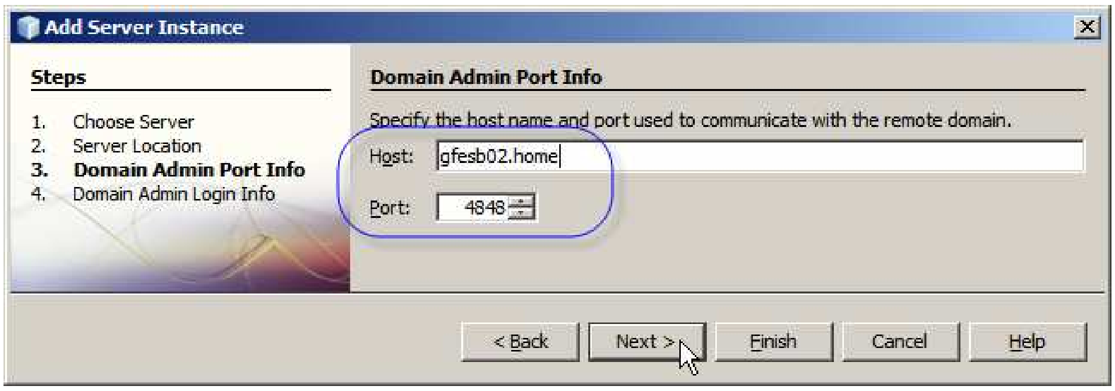
Domain: localhost:24848(...\domains\domain1)

( ) Register Local Domain

[✓ Register Remote Domain]

( ) Create Personal Domain

Profile: Default


Add Server Instance

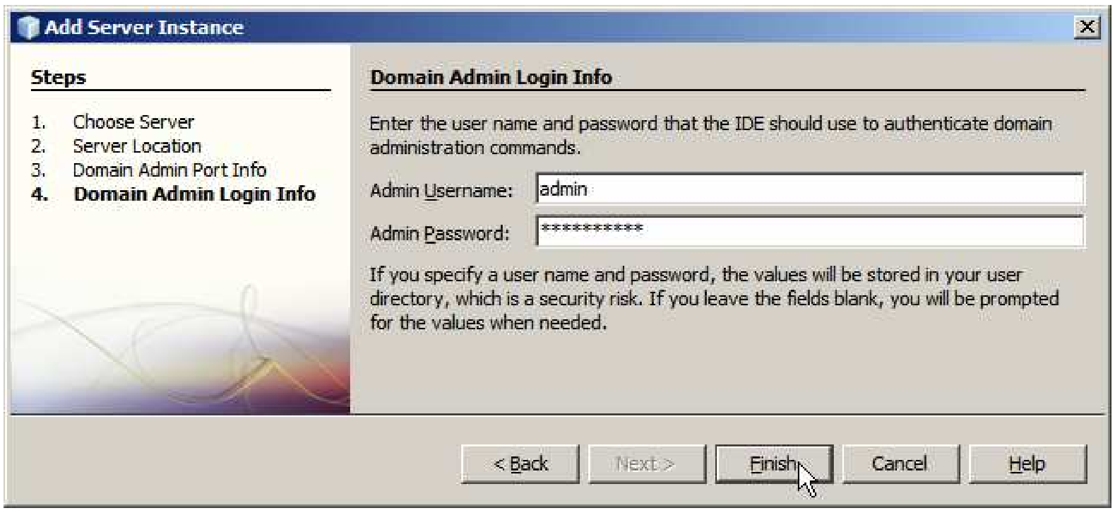
Steps
1. Choose Server
2. Server Location
3. **Domain Admin Port Info**
4. Domain Admin Login Info

Domain Admin Port Info

Specify the host name and port used to communicate with the remote domain.

Host: gfesb02.home

Port: 4848


Add Server Instance

Steps
1. Choose Server
2. Server Location
3. Domain Admin Port Info
4. **Domain Admin Login Info**

Domain Admin Login Info

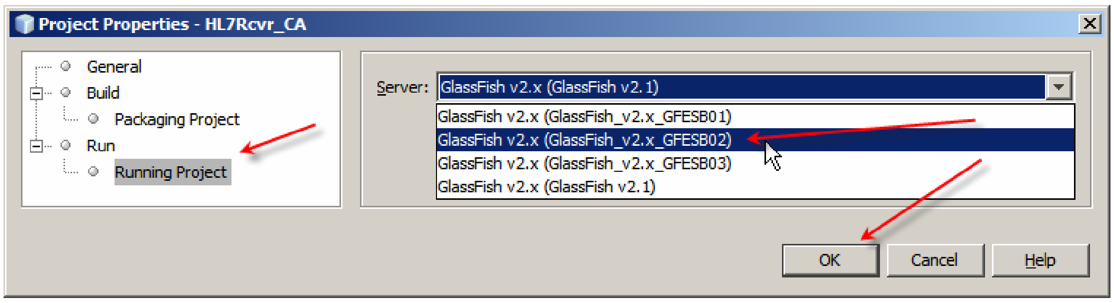
Enter the user name and password that the IDE should use to authenticate domain administration commands.

Admin Username: admin

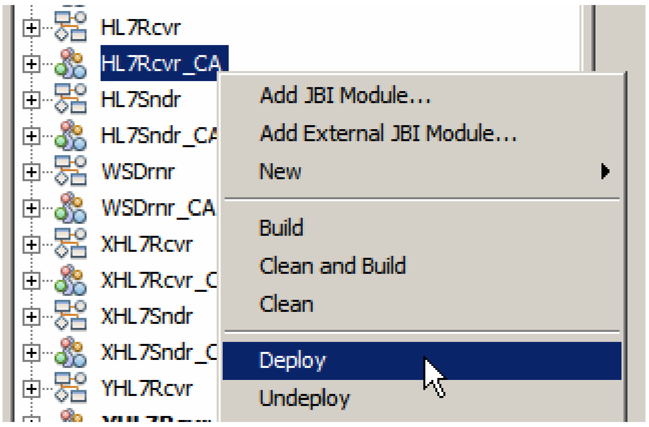
Admin Password: ***********

If you specify a user name and password, the values will be stored in your user directory, which is a security risk. If you leave the fields blank, you will be prompted for the values when needed.

To deploy HL7Rvcr to gfesb02, for example. Right-click HL7Rcvr_CA, choose Properties, select “Running Project”, select “GlassFish v2.x (GlassFish v2.x gfesb02)” (or gfesb03) and click OK.


Project Properties - HL7Rcvr_CA
General
Build
Packaging Project
Run
Running Project
Server: GlassFish v2.x (GlassFish v2.1)
GlassFish v2.x (GlassFish_v2.x_GFESB01)
GlassFish v2.x (GlassFish_v2.x_GFESB02)
GlassFish v2.x (GlassFish_v2.x_GFESB03)
GlassFish v2.x (GlassFish v2.1)
OK
Cancel
Help

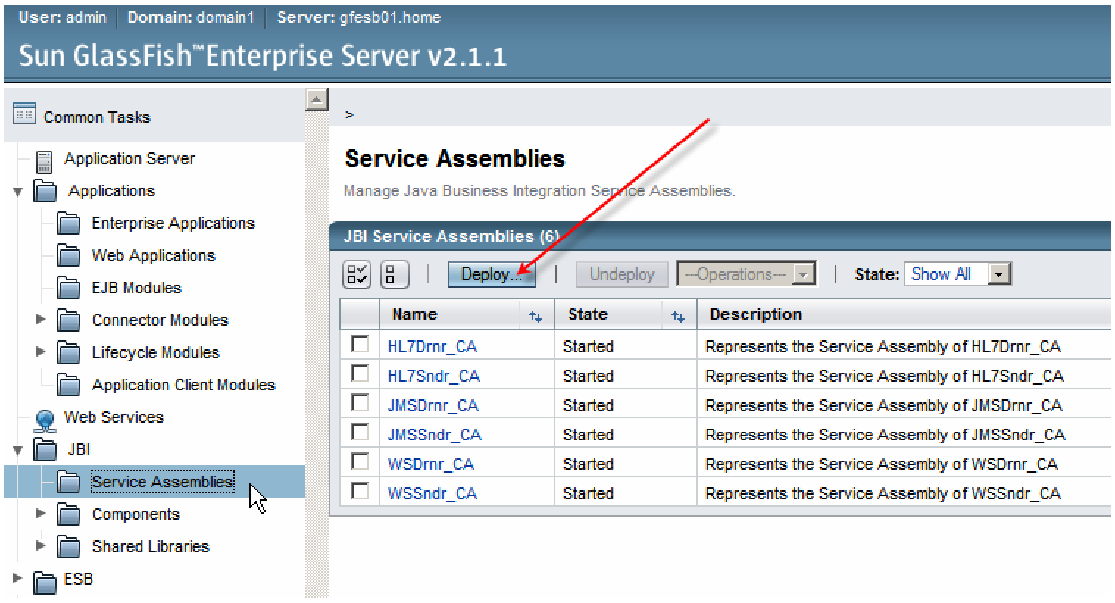
Once the deployment target is set, which is what just happened, right-click HL7Rcvr_CA and choose Deploy.



Deploy HL7Sndr_CA and HL7Drnr_CA to gfesb01.
Deploy HL7Rcvr_CA to gfesb02 and gfesb03.

Using the GlassFish Application Server Admin Console.

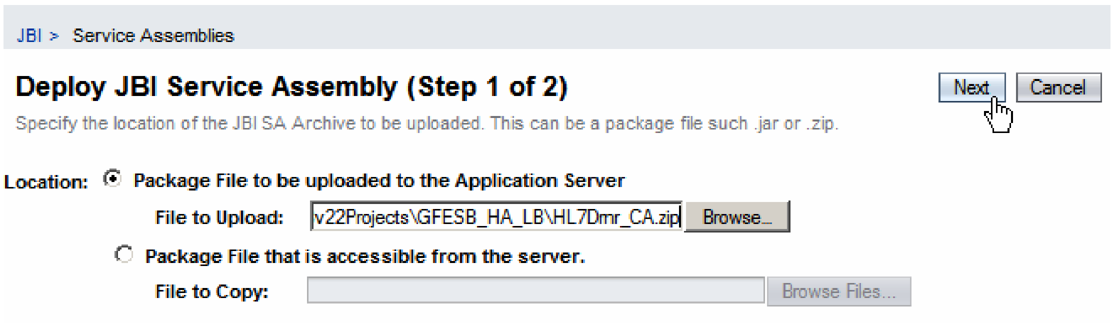
Use a modern Web Browser to Start the GlassFish Application Server Admin Console on gfesb01 (http://gfesb01:4848). Log in as user admin with password adminadmin (or whatever password you provided at installation time). Select JBI → Service Assemblies node.



Name	State	Description
HL7Drnr_CA	Started	Represents the Service Assembly of HL7Drnr_CA
HL7Sndr_CA	Started	Represents the Service Assembly of HL7Sndr_CA
JMSDrnr_CA	Started	Represents the Service Assembly of JMSDrnr_CA
JMSSndr_CA	Started	Represents the Service Assembly of JMSSndr_CA
WSDrnr_CA	Started	Represents the Service Assembly of WSDrnr_CA
WSSndr_CA	Started	Represents the Service Assembly of WSSndr_CA

Click “Deploy” in the right hand window.

Navigate to where the HL7Drnr_CA.zip Composite Application Service Assembly is stored, choose it and click Next.



Deploy JBI Service Assembly (Step 1 of 2)

Specify the location of the JBI SA Archive to be uploaded. This can be a package file such .jar or .zip.

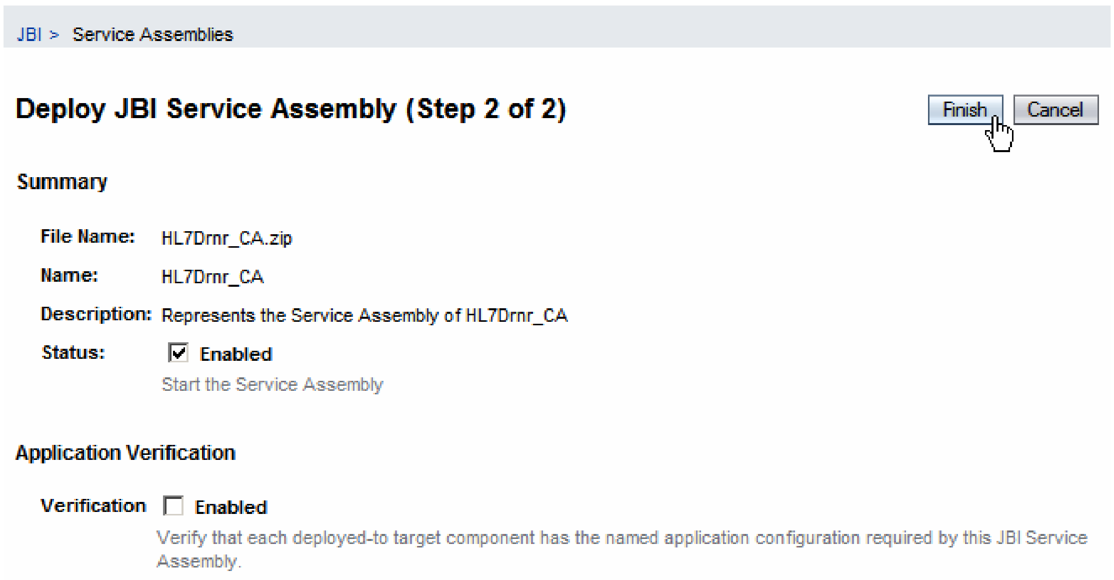
Location: [✓ Package File to be uploaded to the Application Server]

File to Upload: v22Projects\GFESB_HA_LB\HL7Drnr_CA.zip

Package File that is accessible from the server.

File to Copy:

This will cause the HL7Drnr_CA.zip to be transferred to the target host, ready to be deployed in the next step.



Deploy JBI Service Assembly (Step 2 of 2)

Summary

File Name: HL7Drnr_CA.zip

Name: HL7Drnr_CA

Description: Represents the Service Assembly of HL7Drnr_CA

Status: ☑ Enabled

Start the Service Assembly

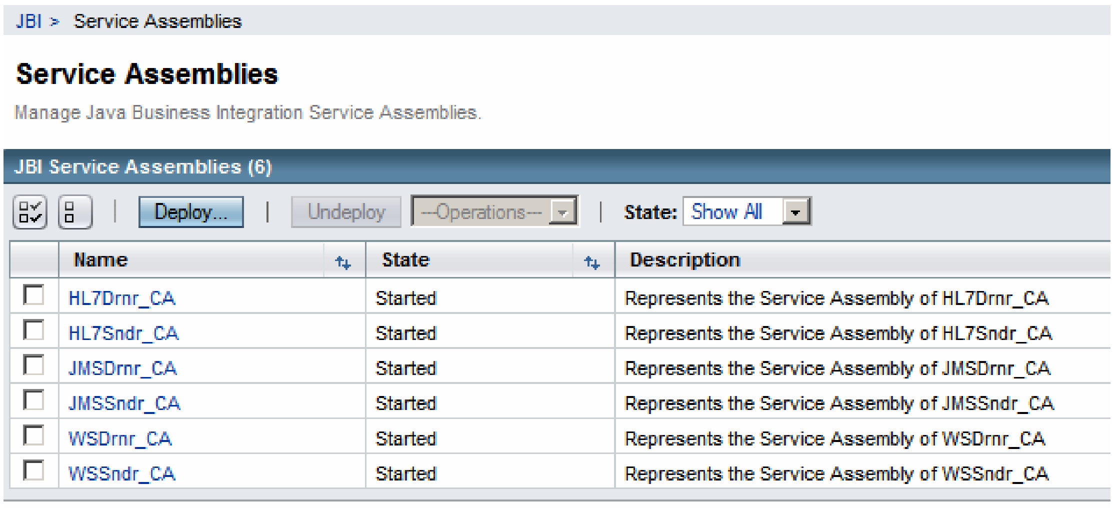
Application Verification

Verification ☐ Enabled

Verify that each deployed-to target component has the named application configuration required by this JBI Service Assembly.

Confirm the details and click Finish to deploy the Service Assembly.

Once deployed, the Service Assembly will appear in the list.



	Name	State	Description
☐	HL7Drnr_CA	Started	Represents the Service Assembly of HL7Drnr_CA
☐	HL7Sndr_CA	Started	Represents the Service Assembly of HL7Sndr_CA
☐	JMSDrnr_CA	Started	Represents the Service Assembly of JMSDrnr_CA
☐	JMSSndr_CA	Started	Represents the Service Assembly of JMSSndr_CA
☐	WSDrnr_CA	Started	Represents the Service Assembly of WSDrnr_CA
☐	WSSndr_CA	Started	Represents the Service Assembly of WSSndr_CA

Follow this method to deploy the HL7Sndr_CA.zip to gfesb01 and HL7Rvcr_CA.zip to gfesb02 and gfesb03.

HL7 BC and Load Balancer Timing Parameters

The set of HL7 BC-based projects use the HL7 BC retry processing functionality to detect connection disruption and take steps to resubmit a message. Selected timing parameters in the HL7 BC in the sender are coordinated with the PEN load balancer's timing parameters and the expected duration of HL7Rcvr's processing to support seamless failover.

As discussed in section "Load Balancer configuration for the HL7 BC-based Solution", pen load balancer command line I use looks like this:

```
pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -ddddd -f -l
pen.log -p pen.pid -r -w pen.stats.html 34001 gfesb02
gfesb03
```

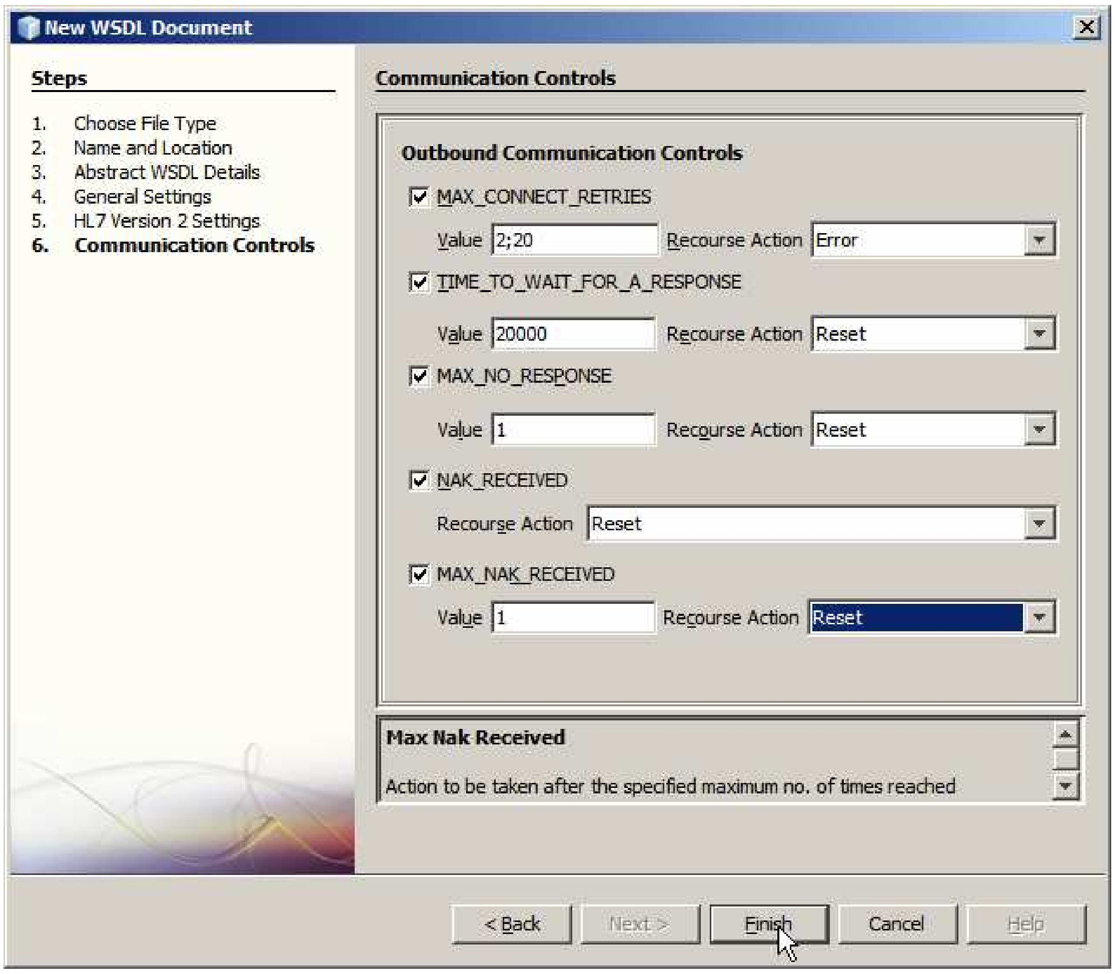
By default pen's connect timeout is 5 seconds. If it fails to connect to the server in that time it will blacklist the server for a default period of 30 seconds. With round robin load balancing algorithm in operation (-r) a failed server will add 5 seconds to the time it takes to send a message every 30 seconds because pen will try to determine if the server came back to life and can be used for processing.

The HL7Rcvr (and all other receivers) introduce a random delay of between 0 and 10 seconds for each message they process. This is to simulate real work being done and slow everything down to the human speed. This means that a receiver may on occasion take in excess of 10 seconds to process a message.

Since the sender expects an application acknowledgement it must be prepared to wait for that acknowledgement in face of delays introduced by the receiver and these introduced by the load balancer.

A connection timeout of at least 5 seconds, preferably 10 seconds, and response timeout of at least 15 seconds (5 + 10), preferably 30 seconds, is a minimum requirement. These timing parameters require tuning based on experience with the solution behavior.

Sender timings are defined in the HL7Sndr's HL7 BC configuration, represented by the HL7Sndr_HL7Out.wsdl in the NetBeans project HL7Sndr.


New WSDL Document

Steps
1. Choose File Type
2. Name and Location
3. Abstract WSDL Details
4. General Settings
5. HL7 Version 2 Settings
6. **Communication Controls**

Communication Controls

Outbound Communication Controls

☑ MAX_CONNECT_RETRIES
Value: 2;20 — Recourse Action: Error

☑ TIME_TO_WAIT_FOR_A_RESPONSE
Value: 20000 — Recourse Action: Reset

☑ MAX_NO_RESPONSE
Value: 1 — Recourse Action: Reset

☑ NAK_RECEIVED
Recourse Action: Reset

☑ MAX_NAK_RECEIVED
Value: 1 — Recourse Action: Reset

Max Nak Received
Action to be taken after the specified maximum no. of times reached

In particular:

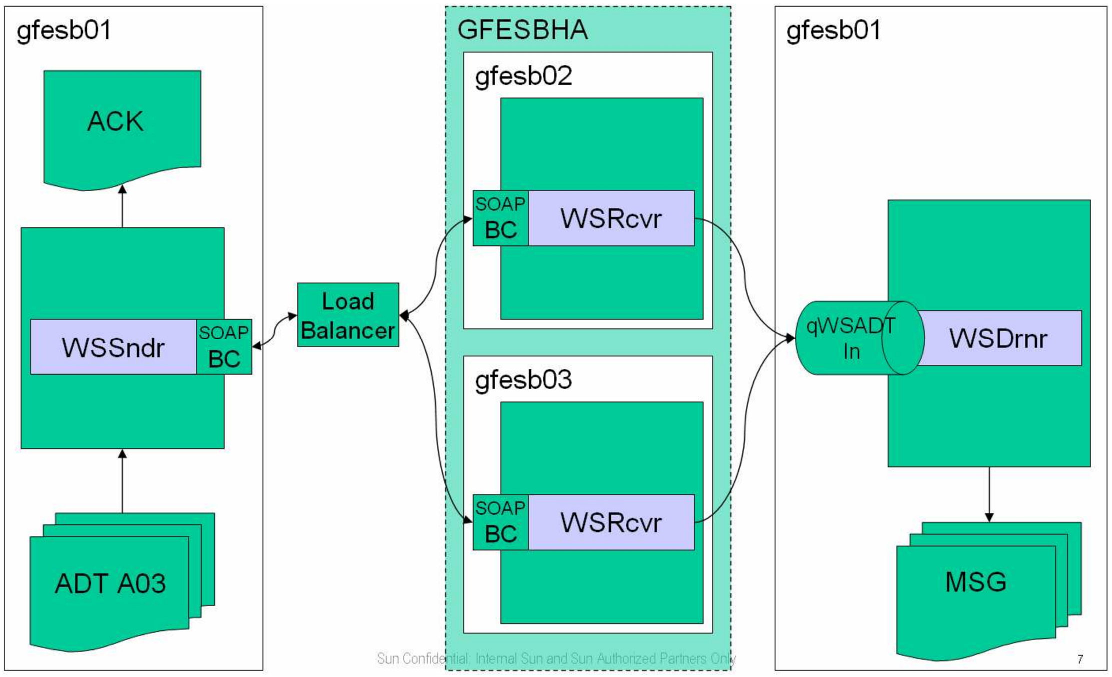
MAX_CONNECT_RETRIES, defined as "retry up to 2 times with a 20 second delay between retries" (2;20), will cause the HL7 BC to retry a connection to the load balancer, consequently to an alternate host, if a connection fails. The 20 second delay is more then is required to allow the pen load balancer to try a connection to the server, timeout if it is not available, and try to connect to another server.

TIME_TO_WAIT_FOR_RESPONSE, defined as 20000 milliseconds (20 seconds) is twice the maximum expected processing time (random value of up to 10 seconds) plus the load balancer connect timeout (5 seconds) and has a margin of 5 seconds added to minimize the potential of loosing a response that happens to take more then 15 seconds to return.

The HL7 BC configuration in the sender must account not only for the expected maximum time it may take to return an application acknowledgement but also for the potential delays introduced by the load balancer and the network infrastructure.

WS Projects

WSSndr_CA and WSDrnr_CA must be deployed to gfesb01. WSRcvr_CA must be deployed to gfesb02 and gfesb03. The steps to deploy the projects are the same as for the HL7 BC-based projects discussed in previous sections.



The HTTP BC-based projects use the WSSndr CASA redelivery handling QoS Properties to configure detection of connection disruption and message resubmission, in conjunction with modification of the default HTTP connection header.

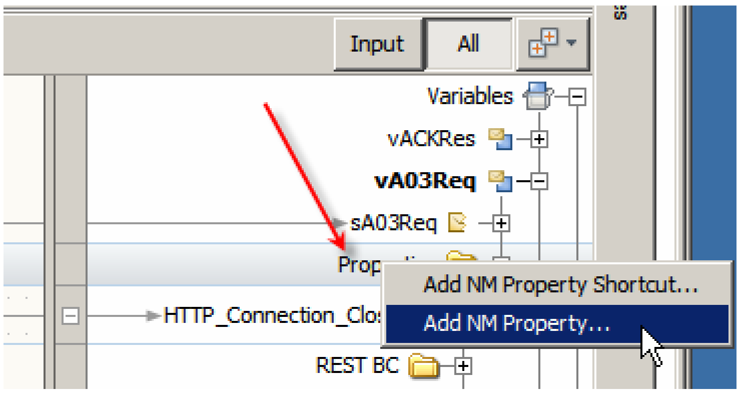
Timing parameters in the HTTP BC CASA QoS properties are coordinated with t eh pen load balancer timing parameters and the expected duration of WSRcvr's processing to support seamless failover.

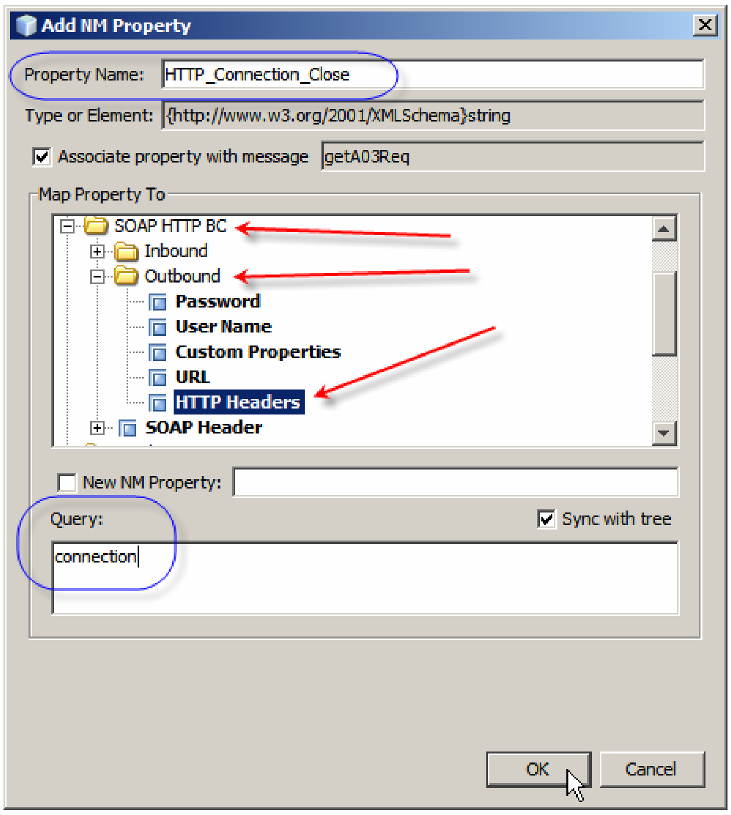
As discussed in section "Load Balancer configuration for the HTTP BC-based Solution", pen lpad balancer command line used for this set of projects looks like this:

```
pen-0.17.1a.exe -C 44001 -X -a -S 2 -d -ddddd -f -l
pen.log -p pen.pid -r -w pen.stats.html -t 5 -b 60 9080
gfesb02 gfesb03
```

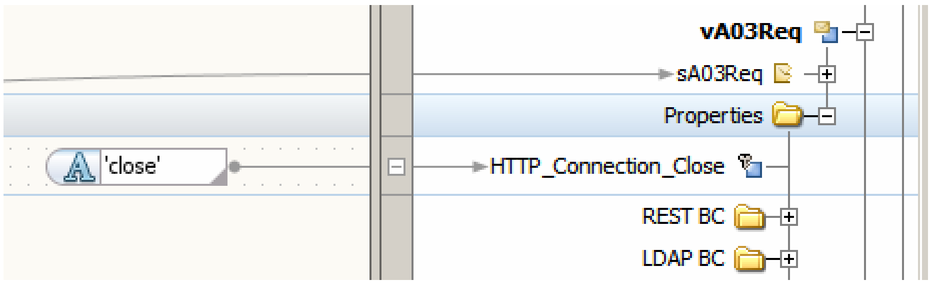
The pen load balancer is configured for round robin load balancing (-r). Each request will be submitted to a different server, if HTTP headers permit. By default the HTTP "connection: keep-alive" header is added by the infrastructure underlying the HTTP BC and the load balancer keeps the single connection established by the sender for all requests the sender sends. When the receiver fails the load balancer closes the connection and the sender re-tries, causing the load balancer to open a connection to the surviving receiver. This is fine for fail-over but not for load balancing. Forcing connection closure after each request, HTTP header "connection: close", allows the load balancer to load balance requests at the cost of creating and destroying connections for each request.

The WSSndr BPEL process implements the HTTP header modification using the NMProperties functionality.





Property Name: HTTP_Connection_Close
Type or Element: {http://www.w3.org/2001/XMLSchema}string
☑ Associate property with message getA03Req
☐ New NM Property:
Query: ☑ Sync with tree
connection

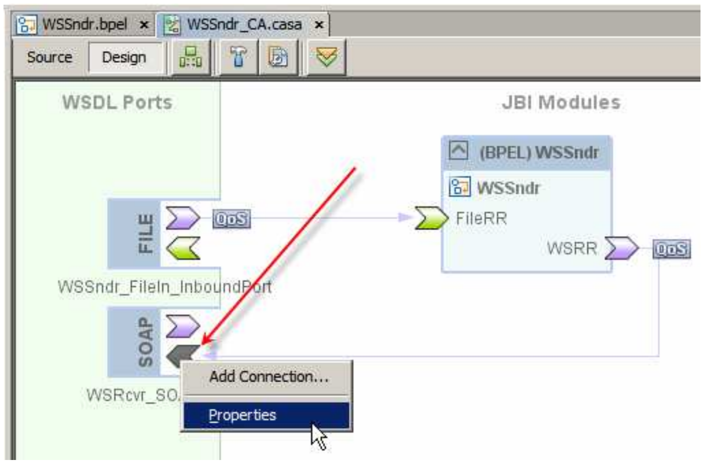


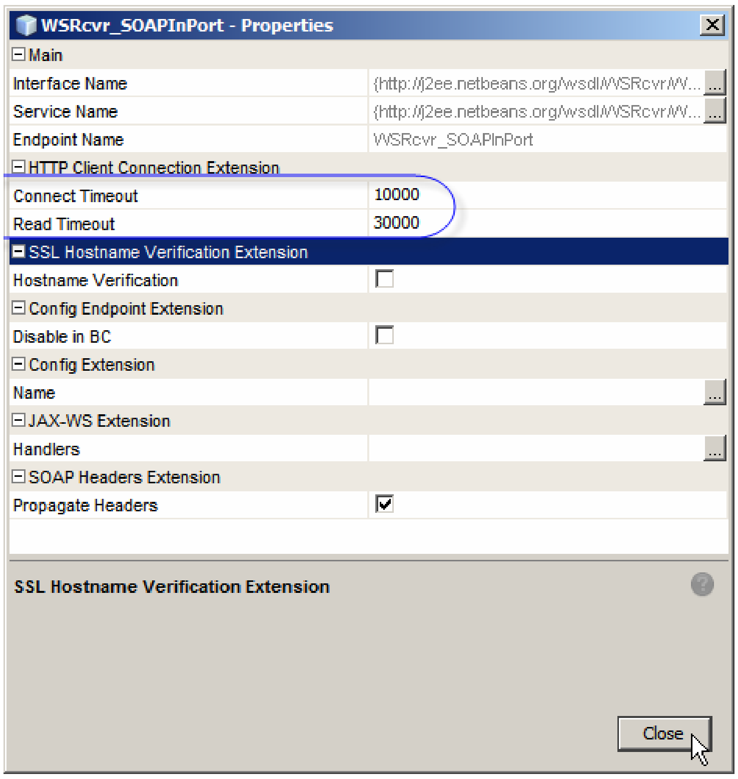
The read timeout (-t 5) is an arbitrary number of seconds we are prepared to wait before concluding the server is not going to accept the connection and giving up on the server for the time defined by the blacklist period.

Blacklist period of 60 seconds (-b 60) is a compromise between the frequency at which a potentially dead server is probed, which costs –t *x* seconds, and the time it takes for a server that is ready to process workload to be used by the load balancer, which may cost up to –b *x* seconds.

Accommodating the load balancer and WSRcvr timings, the WSSndr HTTP BC configuration for connect and read timeout is:

Read Timeout = 30000 (Max 10000 for WSRcvr processing x 2 + 5000 for PEN Read Timeout x 2)




WSRcvr_SOAPInPort - Properties

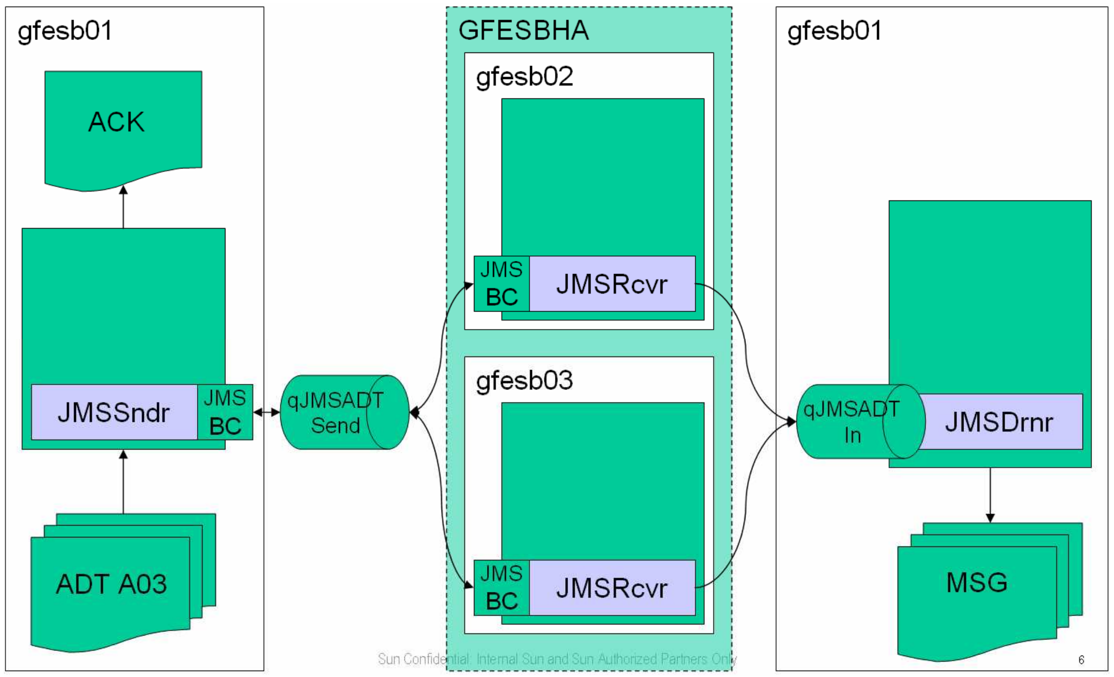
Main	
Interface Name	{http://j2ee.netbeans.org/wsdl/WSRcvr/W...
Service Name	{http://j2ee.netbeans.org/wsdl/WSRcvr/W...
Endpoint Name	WSRcvr_SOAPInPort
HTTP Client Connection Extension	
Connect Timeout	10000
Read Timeout	30000
SSL Hostname Verification Extension	
Hostname Verification	☐
Config Endpoint Extension	
Disable in BC	☐
Config Extension	
Name	
JAX-WS Extension	
Handlers	
SOAP Headers Extension	
Propagate Headers	☑

SSL Hostname Verification Extension

The HTTP BC configuration in the sender must account not only for the expected maximum time it may take to return an application acknowledgement but also for the potential delays introduced by the load balancer and the network infrastructure, and any HTTP header-based connection longevity requirements.

JMS Projects

JMSSndr_CA and JMSDrnr_CA must be deployed to gfesb01. JMSRcvr_CA must be deployed to gfesb02 and gfesb03. The steps to deploy the projects are the same as for the HL7 BC-based projects.



A JMS-based sender typically sends a message to the JMS Server, receives a “transport acknowledgment” from the JMS Server and continues, regardless of whether the message got delivered to the next component. To ensure delivery past the JMS Server, the JMSSndr client and the JMSRcvr service cooperate at the application level. The JMSSndr expects the JMSRcvr to return a specific acknowledgment message when it processes the message it received. This means that when the JMSRcvr fails the JMSSndr will block waiting for the application acknowledgment it will never get.

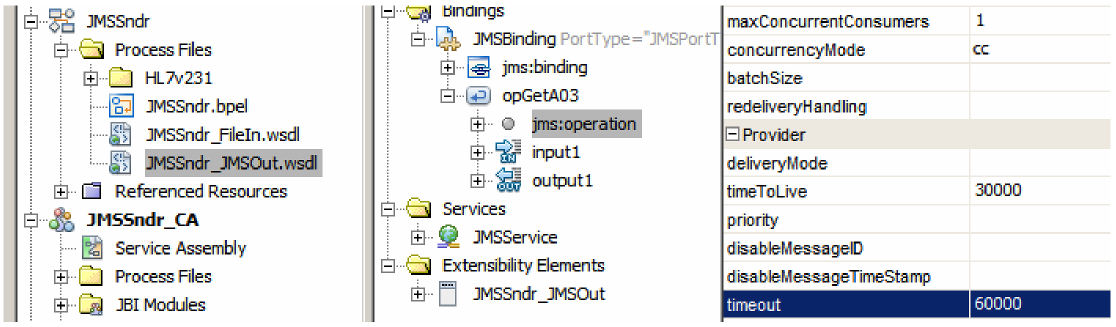
The sender must be configured to retry delivery if it does not receive an application acknowledgement in the expected time. This is a bit trickier then would initially appear. The JMSSndr submits a message to the JMS Server, and then waits for the acknowledgement message (request/reply). When the JMS server receives a message, the message becomes available to the JMSRcvr, which pick it up and starts processing it.

Redelivery handling configuration in the JMS BC and redelivery handling configuration in the Composite Application Service Assembly (CASA) are designed to deal JMS BC’s inability to deliver to the JMS Server. By the time the JMSRcvr received a message the JMS BC in JMSSndr will have successfully delivered the message. JMS BC-level and CASA-level redelivery handling logic will not be effective.

If the JMSRcvr host fails, the acknowledgement message will not get sent back to the JMS Server and will not become available to the JMSSndr. JMS BC in the JMSSndr will eventually time out.

As apparently operating in GlassFish ESB v2.2, the JMSSndr will not get a fault on timeout, as expected, but rather it will get an empty message. The JMS BC-based sender must implement explicit logic to recognize that the acknowledgment message is empty, meaning the send failed, and explicitly retry sending the message.

The only configuration parameters that are effective are time to live (before message is expired and thrown out by the messaging service) and Timeout (waiting for a reply). The latter controls how much time will elapse before the JMSSndr receives an empty ACK message.



```
JMSSndr
    Process Files
        HL7v231
        JMSSndr.bpel
        JMSSndr_FileIn.wsdl
        JMSSndr_JMSOut.wsdl
    Referenced Resources
JMSSndr_CA
    Service Assembly
    Process Files
    JBI Modules
```

```
Bindings
    JMSBinding PortType="JMSPortT
        jms:binding
        opGetA03
            jms:operation
            input1
            output1
Services
    JMSService
Extensibility Elements
    JMSSndr_JMSOut
```

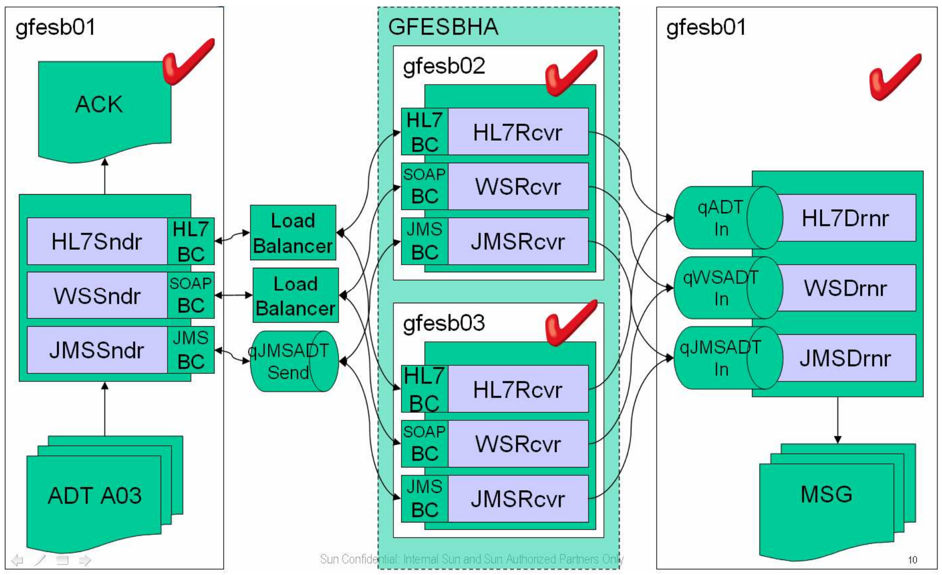
maxConcurrentConsumers	1
concurrencyMode	cc
batchSize	
redeliveryHandling	
⊟ Provider	
deliveryMode	
timeToLive	30000
priority	
disableMessageID	
disableMessageTimeStamp	
timeout	60000

Because there is no load balancer involved, the JMS BC configuration in the sender must account only for the expected maximum time it may take to return an application acknowledgement.

The implementation of retry functionality in the JMSSndr is briefly discussed in section “Overview of Sender Projects”.

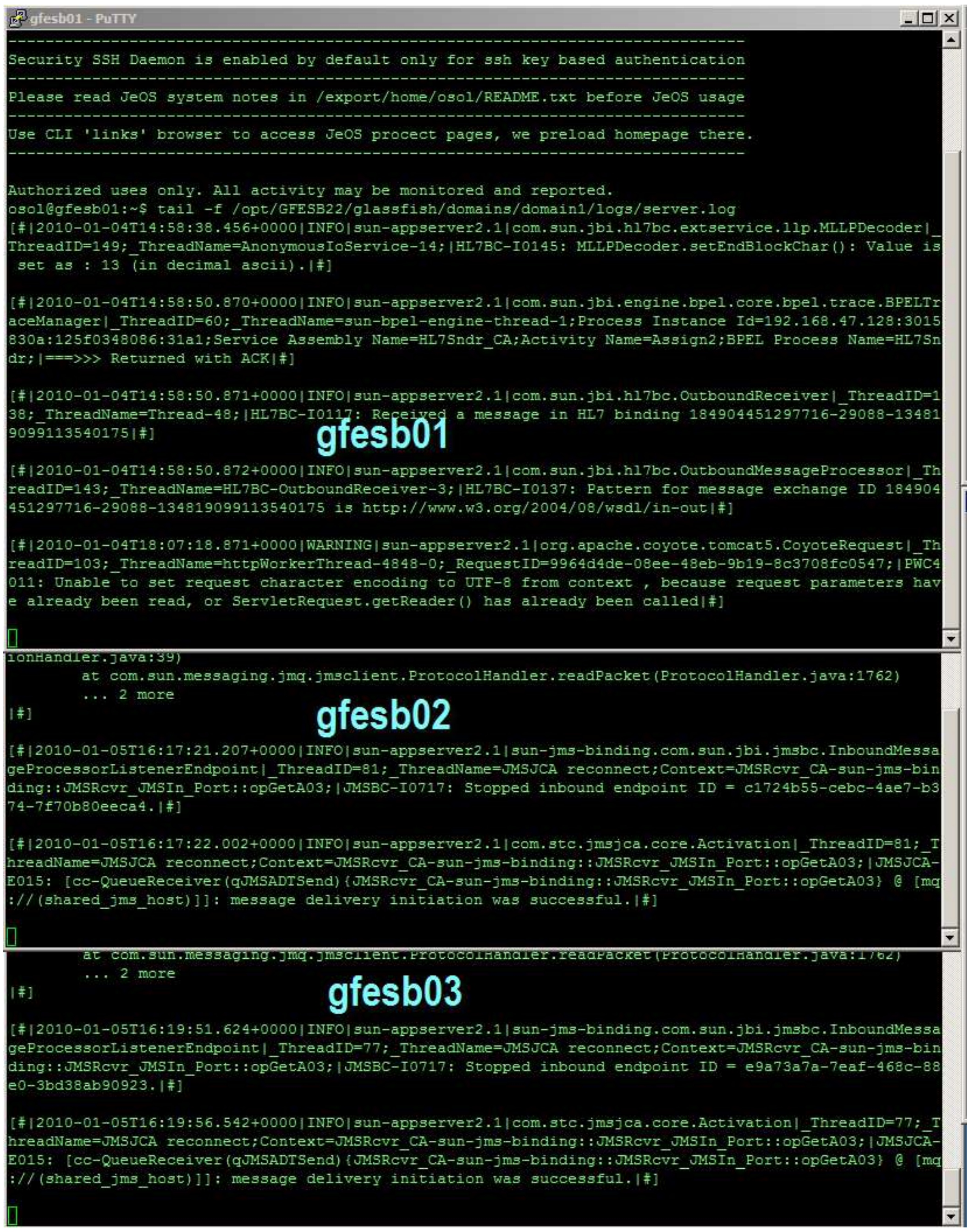
Test Preparation

The environment must be brought to the state ready for testing in the configuration shown below.



To prepare the environment perform the following steps:

1. Start gfesb01 VM until it shows the IP address
2. Start Putty/SSH Client on gfesb01 and tail server.log (run the unix tail program, or a Windows equivalent, to continuously display server.log as it is being written to)
3. Start gfesb02 VM until it shows the IP address
4. Start Putty/SSH Client on gfesb02 and tail server.log
5. Start gfesb03 VM until it shows the IP address
6. Start Putty/SSH Client on gfesb03 and tail server.log
7. Move the Putty/SSH Client console windows around in such a way that the bottom 1/3rd of each is shown, with a "tail" command continuously showing the server.log on each of the hosts.

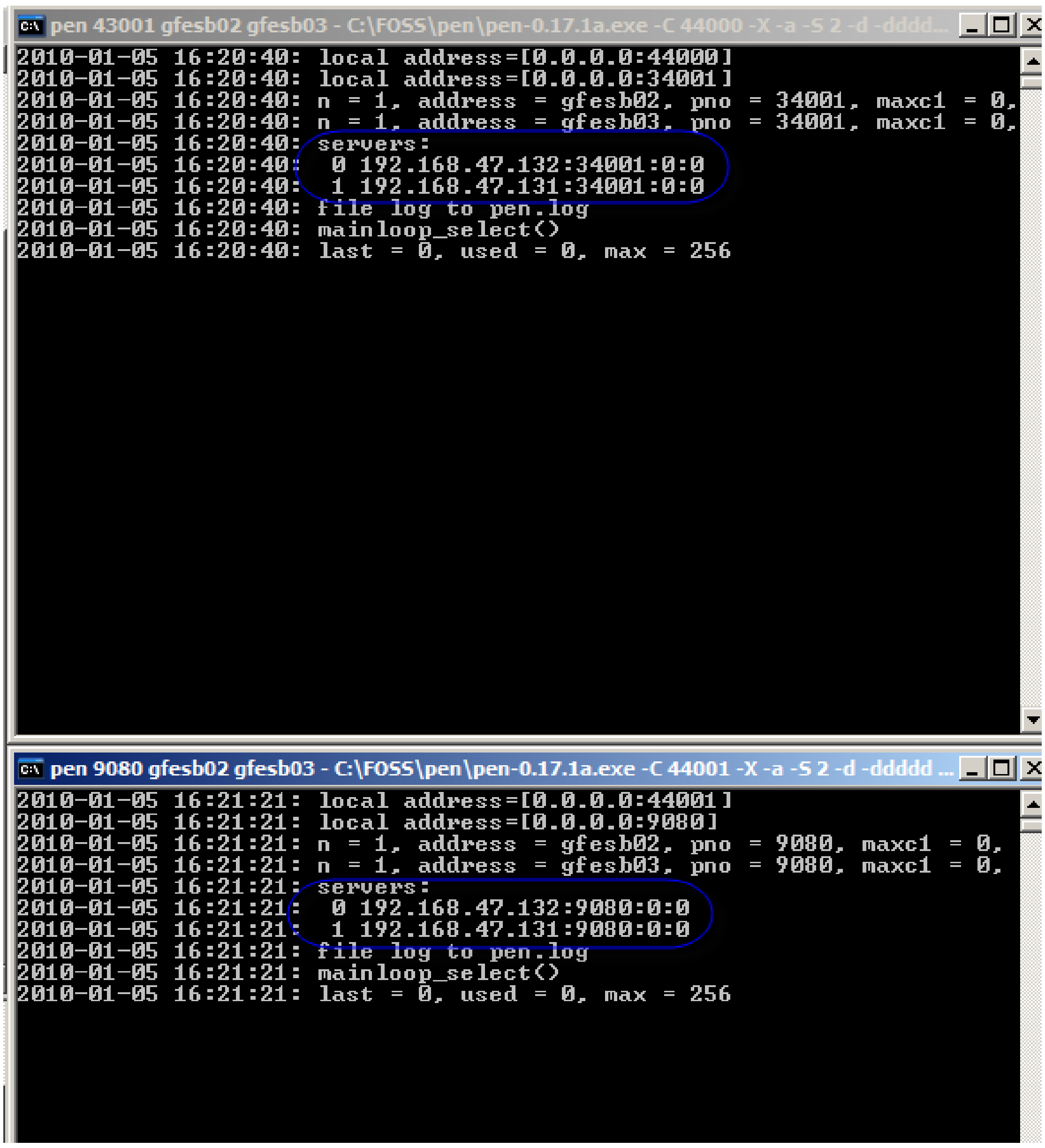


```
--------------------------------------------------------------------------------
Security SSH Daemon is enabled by default only for ssh key based authentication
--------------------------------------------------------------------------------
Please read JeOS system notes in /export/home/osol/README.txt before JeOS usage
--------------------------------------------------------------------------------
Use CLI 'links' browser to access JeOS procect pages, we preload homepage there.
--------------------------------------------------------------------------------

Authorized uses only. All activity may be monitored and reported.
osol@gfesb01:~$ tail -f /opt/GFESB22/glassfish/domains/domain1/logs/server.log
[#|2010-01-04T14:58:38.456+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.llp.MLLPDecoder|_
ThreadID=149;_ThreadName=AnonymousIoService-14;|HL7BC-I0145: MLLPDecoder.setEndBlockChar(): Value is
 set as : 13 (in decimal ascii).|#]

[#|2010-01-04T14:58:50.870+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=60;_ThreadName=sun-bpel-engine-thread-1;Process Instance Id=192.168.47.128:3015
830a:125f0348086:31a1;Service Assembly Name=HL7Sndr_CA;Activity Name=Assign2;BPEL Process Name=HL7Sn
dr;|===>>> Returned with ACK|#]

[#|2010-01-04T14:58:50.871+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.OutboundReceiver|_ThreadID=1
38;_ThreadName=Thread-48;|HL7BC-I0117: Received a message in HL7 binding 184904451297716-29088-13481
9099113540175|#]

[#|2010-01-04T14:58:50.872+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.OutboundMessageProcessor|_Th
readID=143;_ThreadName=HL7BC-OutboundReceiver-3;|HL7BC-I0137: Pattern for message exchange ID 184904
451297716-29088-134819099113540175 is http://www.w3.org/2004/08/wsdl/in-out|#]

[#|2010-01-04T18:07:18.871+0000|WARNING|sun-appserver2.1|org.apache.coyote.tomcat5.CoyoteRequest|_Th
readID=103;_ThreadName=httpWorkerThread-4848-0;_RequestID=9964d4de-08ee-48eb-9b19-8c3708fc0547;|PWC4
011: Unable to set request character encoding to UTF-8 from context , because request parameters hav
e already been read, or ServletRequest.getReader() has already been called|#]

```

```
ionHandler.java:39)
        at com.sun.messaging.jmq.jmsclient.ProtocolHandler.readPacket(ProtocolHandler.java:1762)
        ... 2 more
|#]

[#|2010-01-05T16:17:21.207+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.InboundMessa
geProcessorListenerEndpoint|_ThreadID=81;_ThreadName=JMSJCA reconnect;Context=JMSRcvr_CA-sun-jms-bin
ding::JMSRcvr_JMSIn_Port::opGetA03;|JMSBC-I0717: Stopped inbound endpoint ID = c1724b55-cebc-4ae7-b3
74-7f70b80eeca4.|#]

[#|2010-01-05T16:17:22.002+0000|INFO|sun-appserver2.1|com.stc.jmsjca.core.Activation|_ThreadID=81;_T
hreadName=JMSJCA reconnect;Context=JMSRcvr_CA-sun-jms-binding::JMSRcvr_JMSIn_Port::opGetA03;|JMSJCA-
E015: [cc-QueueReceiver(qJMSADTSend){JMSRcvr_CA-sun-jms-binding::JMSRcvr_JMSIn_Port::opGetA03} @ [mq
://(shared_jms_host)]]: message delivery initiation was successful.|#]

```

```
        at com.sun.messaging.jmq.jmsclient.ProtocolHandler.readPacket(ProtocolHandler.java:1762)
        ... 2 more
|#]

[#|2010-01-05T16:19:51.624+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.InboundMessa
geProcessorListenerEndpoint|_ThreadID=77;_ThreadName=JMSJCA reconnect;Context=JMSRcvr_CA-sun-jms-bin
ding::JMSRcvr_JMSIn_Port::opGetA03;|JMSBC-I0717: Stopped inbound endpoint ID = e9a73a7a-7eaf-468c-88
e0-3bd38ab90923.|#]

[#|2010-01-05T16:19:56.542+0000|INFO|sun-appserver2.1|com.stc.jmsjca.core.Activation|_ThreadID=77;_T
hreadName=JMSJCA reconnect;Context=JMSRcvr_CA-sun-jms-binding::JMSRcvr_JMSIn_Port::opGetA03;|JMSJCA-
E015: [cc-QueueReceiver(qJMSADTSend){JMSRcvr_CA-sun-jms-binding::JMSRcvr_JMSIn_Port::opGetA03} @ [mq
://(shared_jms_host)]]: message delivery initiation was successful.|#]

```

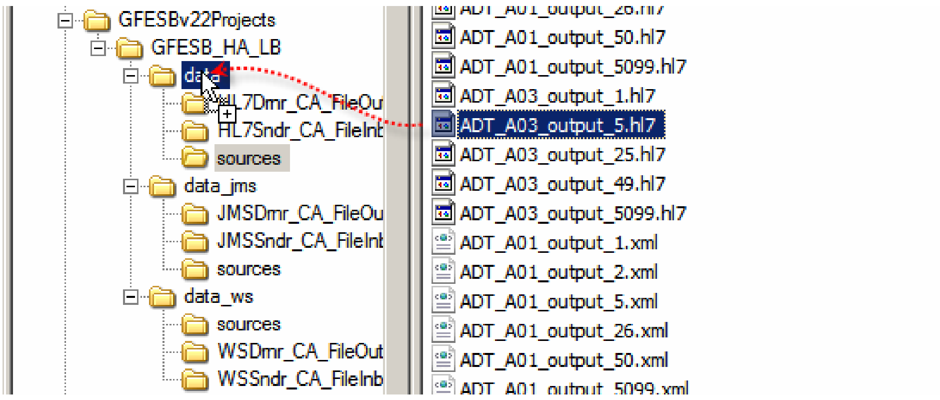
8. Start pen load balancer on port 34001 for the HL7 BC-based projects
9. Start pen load balancer on port 9080 for the HTTP BC-based projects
10. Move pen console windows in such a way such that they are both visible. Note that for pen server 0 is gfesb02 and server 1 is gfesb03.



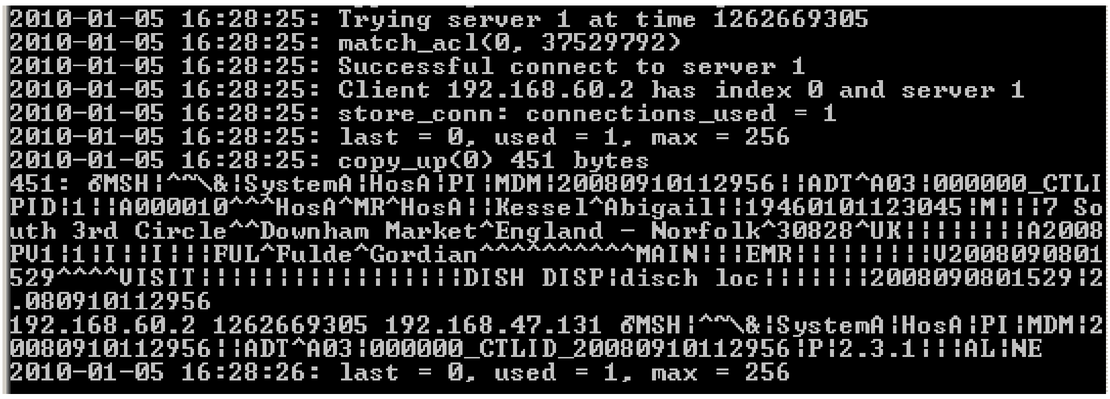
```
pen 43001 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -dddd...
2010-01-05 16:20:40: local address=[0.0.0.0:44000]
2010-01-05 16:20:40: local address=[0.0.0.0:34001]
2010-01-05 16:20:40: n = 1, address = gfesb02, pno = 34001, maxc1 = 0,
2010-01-05 16:20:40: n = 1, address = gfesb03, pno = 34001, maxc1 = 0,
2010-01-05 16:20:40: servers:
2010-01-05 16:20:40:  0 192.168.47.132:34001:0:0
2010-01-05 16:20:40:  1 192.168.47.131:34001:0:0
2010-01-05 16:20:40: file log to pen.log
2010-01-05 16:20:40: mainloop_select()
2010-01-05 16:20:40: last = 0, used = 0, max = 256
```

```
pen 9080 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44001 -X -a -S 2 -d -ddddd ...
2010-01-05 16:21:21: local address=[0.0.0.0:44001]
2010-01-05 16:21:21: local address=[0.0.0.0:9080]
2010-01-05 16:21:21: n = 1, address = gfesb02, pno = 9080, maxc1 = 0,
2010-01-05 16:21:21: n = 1, address = gfesb03, pno = 9080, maxc1 = 0,
2010-01-05 16:21:21: servers:
2010-01-05 16:21:21:  0 192.168.47.132:9080:0:0
2010-01-05 16:21:21:  1 192.168.47.131:9080:0:0
2010-01-05 16:21:21: file log to pen.log
2010-01-05 16:21:21: mainloop_select()
2010-01-05 16:21:21: last = 0, used = 0, max = 256
```

11. Submit 5 message sets for HL7xxx, WSxxx and JMSxxx on gfesb01 by copying the ADT_A03_output_5.hl7 file to data, data_jms and data_ws directories, to prime the infrastructure and make sure all components work

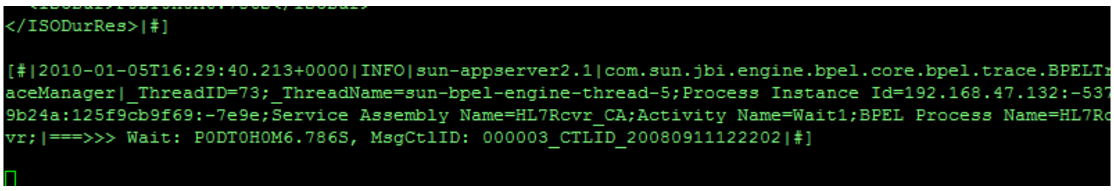


12. Observe messages being sent through the pen load balancer



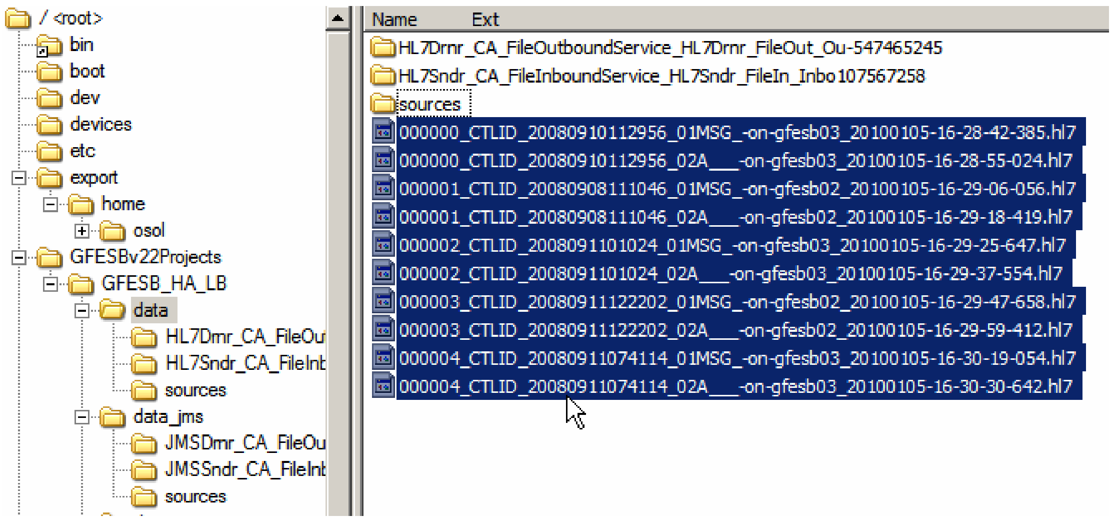
```
2010-01-05 16:28:25: Trying server 1 at time 1262669305
2010-01-05 16:28:25: match_acl(0, 37529792)
2010-01-05 16:28:25: Successful connect to server 1
2010-01-05 16:28:25: Client 192.168.60.2 has index 0 and server 1
2010-01-05 16:28:25: store_conn: connections_used = 1
2010-01-05 16:28:25: last = 0, used = 1, max = 256
2010-01-05 16:28:25: copy_up(0) 451 bytes
451: ♂MSH|^~\&|SystemA|HosA|PI|MDM|20080910112956||ADT^A03|000000_CTLI
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7 So
uth 3rd Circle^^Downham Market^England - Norfolk^30828^UK||||||||A2008
PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^^MAIN|||EMR|||||||||V2008090801
529^^^^VISIT|||||||||||||||||DISH DISP|disch loc|||||||20080908015291|2
.080910112956
192.168.60.2 1262669305 192.168.47.131 ♂MSH|^~\&|SystemA|HosA|PI|MDM|2
0080910112956||ADT^A03|000000_CTLID_20080910112956|P|2.3.1|||AL|NE
2010-01-05 16:28:26: last = 0, used = 1, max = 256
```

13. Observe messages being processed by the gfesb02 / gfesb03



```
</ISODurRes>|#]

[#|2010-01-05T16:29:40.213+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=73;_ThreadName=sun-bpel-engine-thread-5;Process Instance Id=192.168.47.132:-537
9b24a:125f9cb9f69:-7e9e;Service Assembly Name=HL7Rcvr_CA;Activity Name=Wait1;BPEL Process Name=HL7Rc
vr;|===>>> Wait: P0DT0H0M6.786S, MsgCtlID: 000003_CTLID_20080911122202|#]
```

14. once all messages are processed, clear output directories data, data_jms and data_ws of messages



```
/ <root>
  bin
  boot
  dev
  devices
  etc
  export
    home
      osol
  GFESBv22Projects
    GFESB_HA_LB
      data
        HL7Dmr_CA_FileOu
        HL7Sndr_CA_FileInt
        sources
      data_jms
        JMSDmr_CA_FileOu
        JMSSndr_CA_FileInt
        sources
```

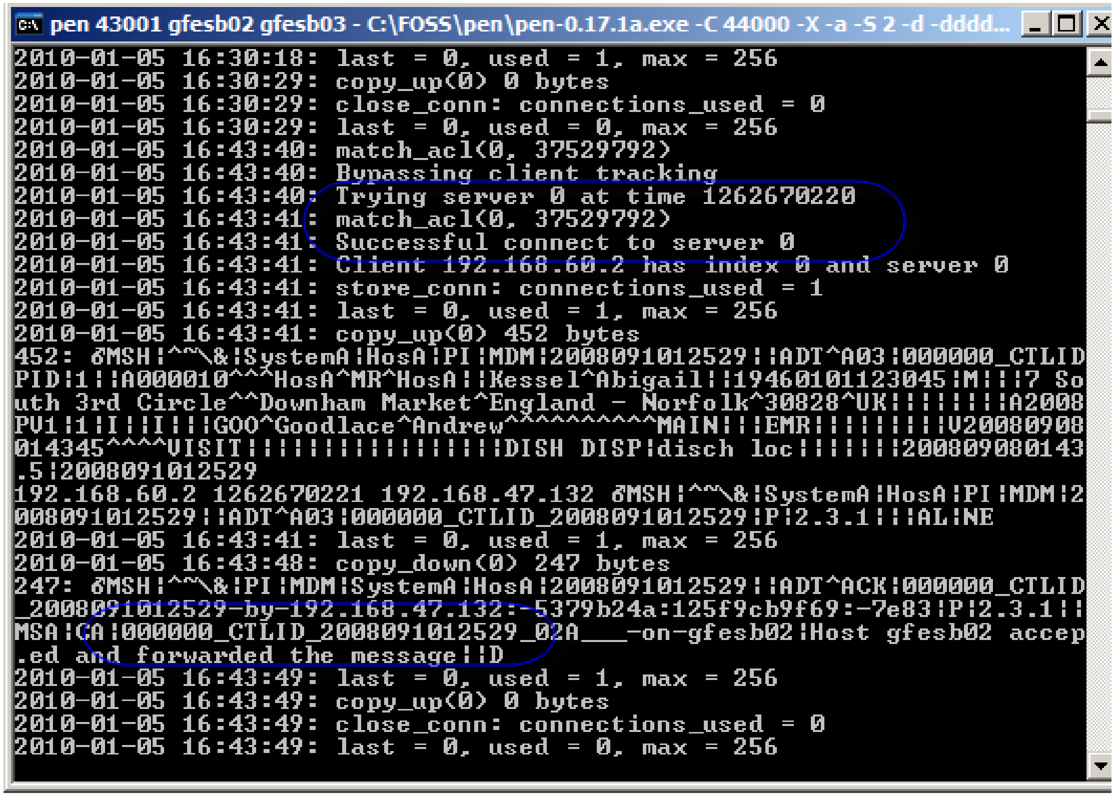
```
Name      Ext
HL7Drnr_CA_FileOutboundService_HL7Drnr_FileOut_Ou-547465245
HL7Sndr_CA_FileInboundService_HL7Sndr_FileIn_Inbo107567258
sources
000000_CTLID_20080910112956_01MSG_-on-gfesb03_20100105-16-28-42-385.hl7
000000_CTLID_20080910112956_02A___-on-gfesb03_20100105-16-28-55-024.hl7
000001_CTLID_20080908111046_01MSG_-on-gfesb02_20100105-16-29-06-056.hl7
000001_CTLID_20080908111046_02A___-on-gfesb02_20100105-16-29-18-419.hl7
000002_CTLID_2008091101024_01MSG_-on-gfesb03_20100105-16-29-25-647.hl7
000002_CTLID_2008091101024_02A___-on-gfesb03_20100105-16-29-37-554.hl7
000003_CTLID_20080911122202_01MSG_-on-gfesb02_20100105-16-29-47-658.hl7
000003_CTLID_20080911122202_02A___-on-gfesb02_20100105-16-29-59-412.hl7
000004_CTLID_20080911074114_01MSG_-on-gfesb03_20100105-16-30-19-054.hl7
000004_CTLID_20080911074114_02A___-on-gfesb03_20100105-16-30-30-642.hl7
```

Testing resilience of the HL7 BC-based Solution

While we can test machine failure will all three streams of messages being processed concurrently, testing a stream at a time, at least initially, will be better to verify correct operation of the configuration.

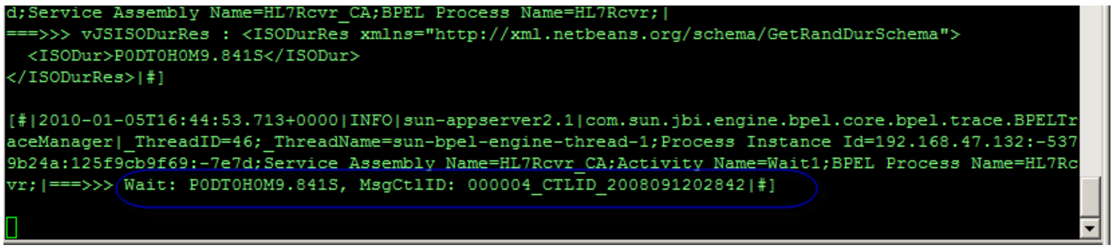
We intend to start with all machines and all components started and processing messages. This is what was happening in the preparation stage discussed in the previous section. We expect messages to be processed in a round robin manner, alternating between gfesb02 and gfesb03. Once 15 or so messages are processed, we will "crash" gfesb03 by closing the VMware Player window in which it runs. This will cause retry functionality, after timeout period, to be invoked. We expect message flow to continue without message loss and we expect messages to now be processed by gfesb02 only. This should be reflected in names of the output files. Once we see enough messages processed by gfesb02, we will boot gfesb03 again and wait for it to start the GlassFish Application Server and deploy all the receivers. As soon as the HL7Rcvr has been deployed and the most recent load balancer blacklist interval for the gfesb03 expired, we expect to see gfesb03 to start picking up messages again. We expect this to also be reflected in the names of the output files.

Let's begin by copying the file ADT_A03_output_101.hl7, containing 101 ADT A03 messages, to the data directory, and observing the console windows and the output directory until around 15 messages are processed.



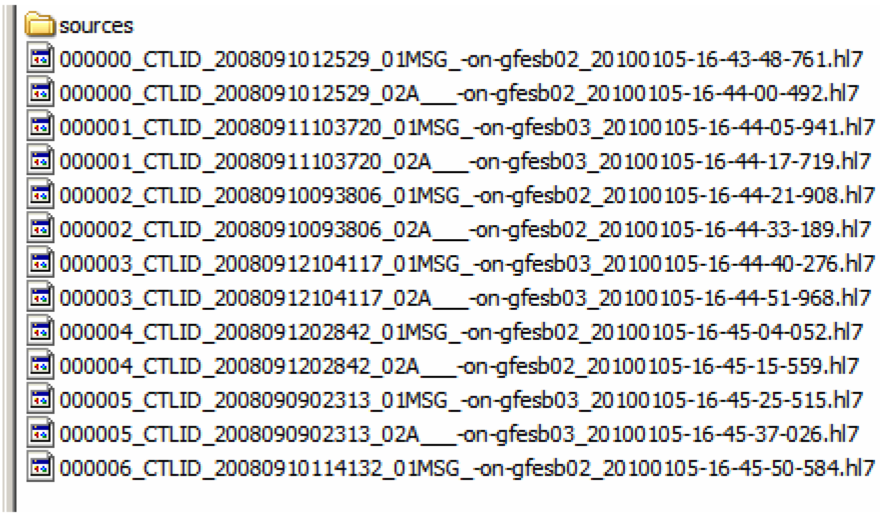
```
pen 43001 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -dddd...
2010-01-05 16:30:18: last = 0, used = 1, max = 256
2010-01-05 16:30:29: copy_up(0) 0 bytes
2010-01-05 16:30:29: close_conn: connections_used = 0
2010-01-05 16:30:29: last = 0, used = 0, max = 256
2010-01-05 16:43:40: match_acl(0, 37529792)
2010-01-05 16:43:40: Bypassing client tracking
2010-01-05 16:43:40: Trying server 0 at time 1262670220
2010-01-05 16:43:41: match_acl(0, 37529792)
2010-01-05 16:43:41: Successful connect to server 0
2010-01-05 16:43:41: Client 192.168.60.2 has index 0 and server 0
2010-01-05 16:43:41: store_conn: connections_used = 1
2010-01-05 16:43:41: last = 0, used = 1, max = 256
2010-01-05 16:43:41: copy_up(0) 452 bytes
452: ♂MSH¦^~\&¦SystemA¦HosA¦PI¦MDM¦20080910125291¦¦ADT^A03¦000000_CTLID
PID¦1¦¦A000010^^^HosA^MR^HosA¦¦Kessel^Abigail¦¦19460101123045¦M¦¦¦7 So
uth 3rd Circle^^Downham Market^England - Norfolk^30828^UK¦¦¦¦¦¦¦¦A2008
PV1¦1¦I¦¦I¦¦¦GOO^Goodlace^Andrew^^^^^^^^^^MAIN¦¦¦EMR¦¦¦¦¦¦¦¦¦V20080908
014345^^^^VISIT¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦DISH DISP¦disch loc¦¦¦¦¦¦¦200809080143
.5¦2008091012529
192.168.60.2 1262670221 192.168.47.132 ♂MSH¦^~\&¦SystemA¦HosA¦PI¦MDM¦2
008091012529¦¦ADT^A03¦000000_CTLID_2008091012529¦P¦2.3.1¦¦¦AL¦NE
2010-01-05 16:43:41: last = 0, used = 1, max = 256
2010-01-05 16:43:48: copy_down(0) 247 bytes
247: ♂MSH¦^~\&¦PI¦MDM¦SystemA¦HosA¦2008091012529¦¦ADT^ACK¦000000_CTLID
_2008091012529-by-192.168.47.132:-5379b24a:125f9cb9f69:-7e83¦P¦2.3.1¦¦
MSA¦CA¦000000_CTLID_2008091012529_02A___-on-gfesb02¦Host gfesb02 accep
.ed and forwarded the message¦¦D
2010-01-05 16:43:49: last = 0, used = 1, max = 256
2010-01-05 16:43:49: copy_up(0) 0 bytes
2010-01-05 16:43:49: close_conn: connections_used = 0
2010-01-05 16:43:49: last = 0, used = 0, max = 256
```

ACK for message ID 000000.



```
d;Service Assembly Name=HL7Rcvr_CA;BPEL Process Name=HL7Rcvr;|
===>>> vJSISODurRes : <ISODurRes xmlns="http://xml.netbeans.org/schema/GetRandDurSchema">
  <ISODur>P0DT0H0M9.841S</ISODur>
</ISODurRes>|#]

[#|2010-01-05T16:44:53.713+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=46;_ThreadName=sun-bpel-engine-thread-1;Process Instance Id=192.168.47.132:-537
9b24a:125f9cb9f69:-7e7d;Service Assembly Name=HL7Rcvr_CA;Activity Name=Wait1;BPEL Process Name=HL7Rc
vr;|===>>> Wait: P0DT0H0M9.841S, MsgCtlID: 000004_CTLID_2008091202842|#]
```

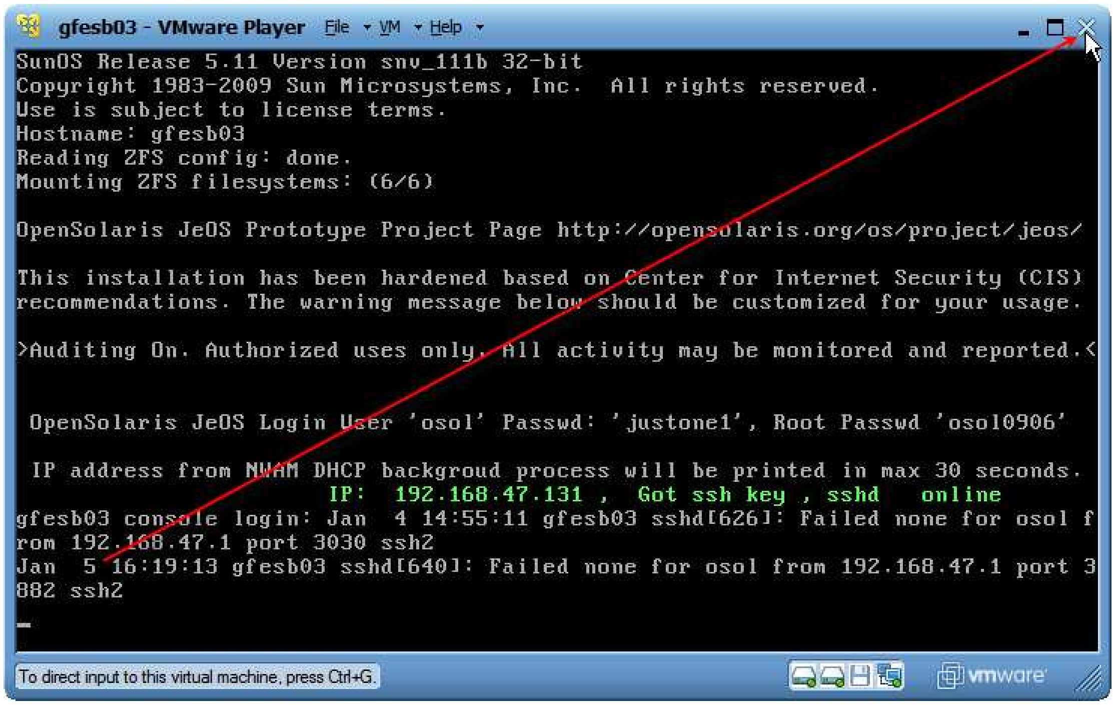
gfesb03 handles message 000004, waiting 9.841 seconds before returning the ACK.



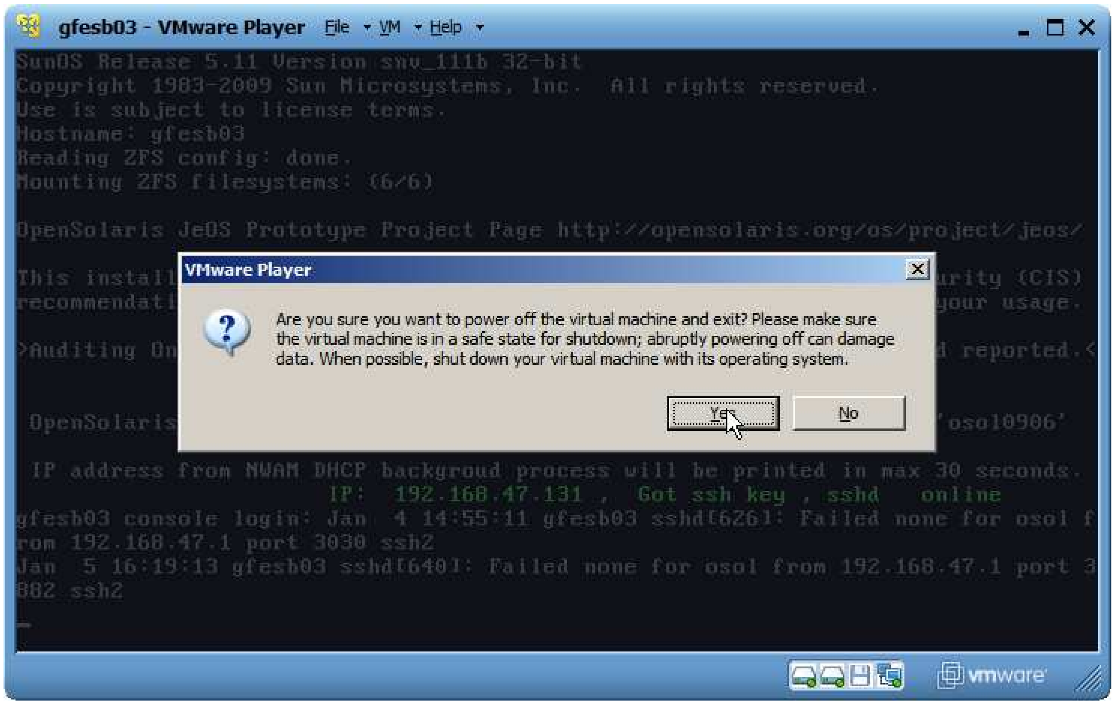
```
sources
000000_CTLID_2008091012529_01MSG_-on-gfesb02_20100105-16-43-48-761.hl7
000000_CTLID_2008091012529_02A___-on-gfesb02_20100105-16-44-00-492.hl7
000001_CTLID_20080911103720_01MSG_-on-gfesb03_20100105-16-44-05-941.hl7
000001_CTLID_20080911103720_02A___-on-gfesb03_20100105-16-44-17-719.hl7
000002_CTLID_20080910093806_01MSG_-on-gfesb02_20100105-16-44-21-908.hl7
000002_CTLID_20080910093806_02A___-on-gfesb02_20100105-16-44-33-189.hl7
000003_CTLID_20080912104117_01MSG_-on-gfesb03_20100105-16-44-40-276.hl7
000003_CTLID_20080912104117_02A___-on-gfesb03_20100105-16-44-51-968.hl7
000004_CTLID_2008091202842_01MSG_-on-gfesb02_20100105-16-45-04-052.hl7
000004_CTLID_2008091202842_02A___-on-gfesb02_20100105-16-45-15-559.hl7
000005_CTLID_2008090902313_01MSG_-on-gfesb03_20100105-16-45-25-515.hl7
000005_CTLID_2008090902313_02A___-on-gfesb03_20100105-16-45-37-026.hl7
000006_CTLID_20080910114132_01MSG_-on-gfesb02_20100105-16-45-50-584.hl7
```

Six messages processed, alternating between gfesb02 and gfesb03. All messages were processed in sequence, so far.

Let’s now crash the gfesb03 to simulate system failure.

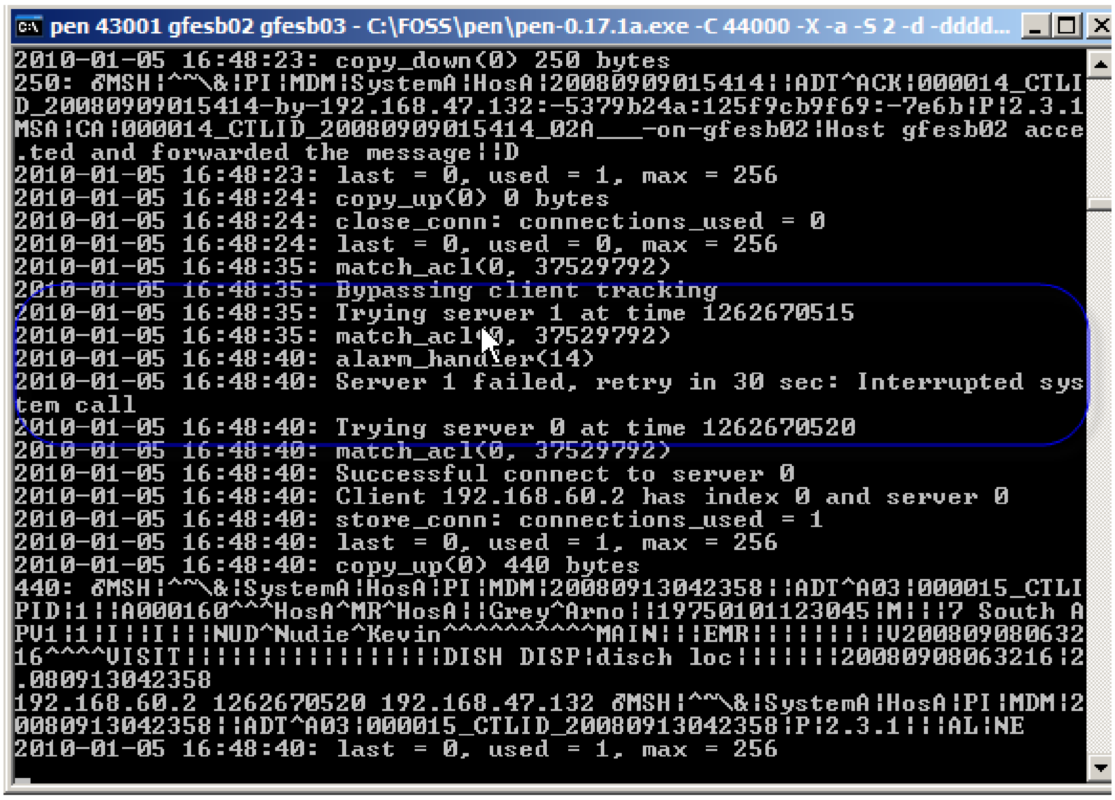


```
SunOS Release 5.11 Version snv_111b 32-bit
Copyright 1983-2009 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Hostname: gfesb03
Reading ZFS config: done.
Mounting ZFS filesystems: (6/6)

OpenSolaris JeOS Prototype Project Page http://opensolaris.org/os/project/jeos/

This installation has been hardened based on Center for Internet Security (CIS)
recommendations. The warning message below should be customized for your usage.

>Auditing On. Authorized uses only. All activity may be monitored and reported.<


 OpenSolaris JeOS Login User 'osol' Passwd: 'justone1', Root Passwd 'osol0906'

 IP address from NWAM DHCP backgroud process will be printed in max 30 seconds.
                       IP:  192.168.47.131 ,  Got ssh key , sshd   online
gfesb03 console login: Jan  4 14:55:11 gfesb03 sshd[626]: Failed none for osol f
rom 192.168.47.1 port 3030 ssh2
Jan  5 16:19:13 gfesb03 sshd[640]: Failed none for osol from 192.168.47.1 port 3
882 ssh2

_
```



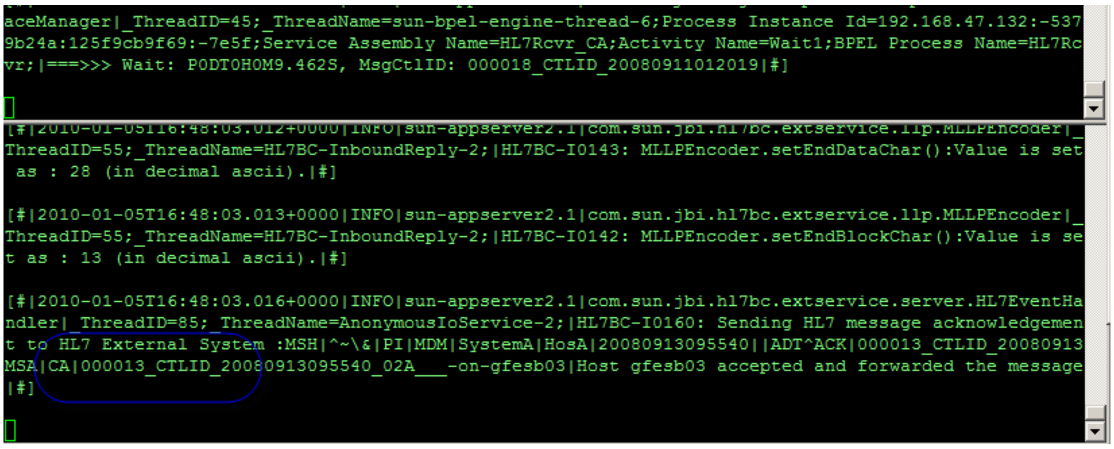
Are you sure you want to power off the virtual machine and exit? Please make sure the virtual machine is in a safe state for shutdown; abruptly powering off can damage data. When possible, shut down your virtual machine with its operating system.

Pen reports that server failed and tries another server.



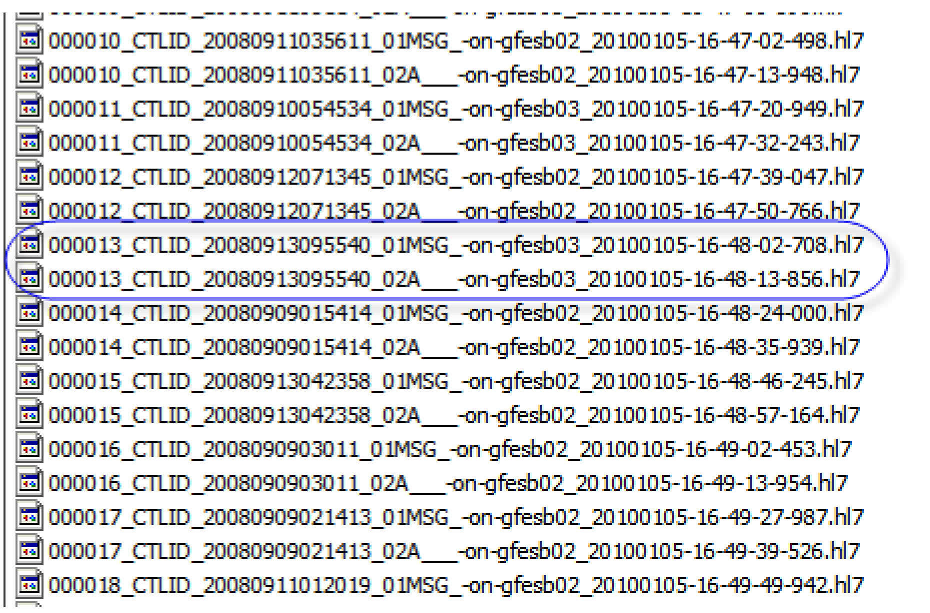
```
pen 43001 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -dddd...
2010-01-05 16:48:23: copy_down(0) 250 bytes
250: δMSH¦^~\&¦PI¦MDM¦SystemA¦HosA¦20080909015414¦¦ADT^ACK¦000014_CTLI
D_20080909015414-by-192.168.47.132:-5379b24a:125f9cb9f69:-7e6b¦P¦2.3.1
MSA¦CA¦000014_CTLID_20080909015414_02A___-on-gfesb02¦Host gfesb02 acce
.ted and forwarded the message¦¦D
2010-01-05 16:48:23: last = 0, used = 1, max = 256
2010-01-05 16:48:24: copy_up(0) 0 bytes
2010-01-05 16:48:24: close_conn: connections_used = 0
2010-01-05 16:48:24: last = 0, used = 0, max = 256
2010-01-05 16:48:35: match_acl(0, 37529792)
2010-01-05 16:48:35: Bypassing client tracking
2010-01-05 16:48:35: Trying server 1 at time 1262670515
2010-01-05 16:48:35: match_acl(0, 37529792)
2010-01-05 16:48:40: alarm_handler(14)
2010-01-05 16:48:40: Server 1 failed, retry in 30 sec: Interrupted sys
tem call
2010-01-05 16:48:40: Trying server 0 at time 1262670520
2010-01-05 16:48:40: match_acl(0, 37529792)
2010-01-05 16:48:40: Successful connect to server 0
2010-01-05 16:48:40: Client 192.168.60.2 has index 0 and server 0
2010-01-05 16:48:40: store_conn: connections_used = 1
2010-01-05 16:48:40: last = 0, used = 1, max = 256
2010-01-05 16:48:40: copy_up(0) 440 bytes
440: δMSH¦^~\&¦SystemA¦HosA¦PI¦MDM¦20080913042358¦¦ADT^A03¦000015_CTLI
PID¦1¦¦A000160^^^HosA^MR^HosA¦¦Grey^Arno¦¦19750101123045¦M¦¦¦7 South A
PV1¦1¦I¦¦I¦¦¦NUD^Nudie^Kevin^^^^^^^^^^MAIN¦¦EMR¦¦¦¦¦¦¦¦¦V200809080632
16^^^^VISIT¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦DISH DISP¦disch loc¦¦¦¦¦¦20080908063216¦2
.080913042358
192.168.60.2 1262670520 192.168.47.132 δMSH¦^~\&¦SystemA¦HosA¦PI¦MDM¦2
0080913042358¦¦ADT^A03¦000015_CTLID_20080913042358¦P¦2.3.1¦¦¦AL¦NE
2010-01-05 16:48:40: last = 0, used = 1, max = 256
```

Last message in progress at the gfesb03 at the time of the “failure” was message 000013.



```
aceManager|_ThreadID=45;_ThreadName=sun-bpel-engine-thread-6;Process Instance Id=192.168.47.132:-537
9b24a:125f9cb9f69:-7e5f;Service Assembly Name=HL7Rcvr_CA;Activity Name=Wait1;BPEL Process Name=HL7Rc
vr;|===>>> Wait: P0DT0H0M9.462S, MsgCtlID: 000018_CTLID_20080911012019|#]

[#|2010-01-05T16:48:03.012+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.llp.MLLPEncoder|_
ThreadID=55;_ThreadName=HL7BC-InboundReply-2;|HL7BC-I0143: MLLPEncoder.setEndDataChar():Value is set
 as : 28 (in decimal ascii).|#]

[#|2010-01-05T16:48:03.013+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.llp.MLLPEncoder|_
ThreadID=55;_ThreadName=HL7BC-InboundReply-2;|HL7BC-I0142: MLLPEncoder.setEndBlockChar():Value is se
t as : 13 (in decimal ascii).|#]

[#|2010-01-05T16:48:03.016+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.server.HL7EventHa
ndler|_ThreadID=85;_ThreadName=AnonymousIoService-2;|HL7BC-I0160: Sending HL7 message acknowledgemen
t to HL7 External System :MSH|^~\&|PI|MDM|SystemA|HosA|20080913095540||ADT^ACK|000013_CTLID_20080913
MSA|CA|000013_CTLID_20080913095540_02A___-on-gfesb03|Host gfesb03 accepted and forwarded the message
|#]
```

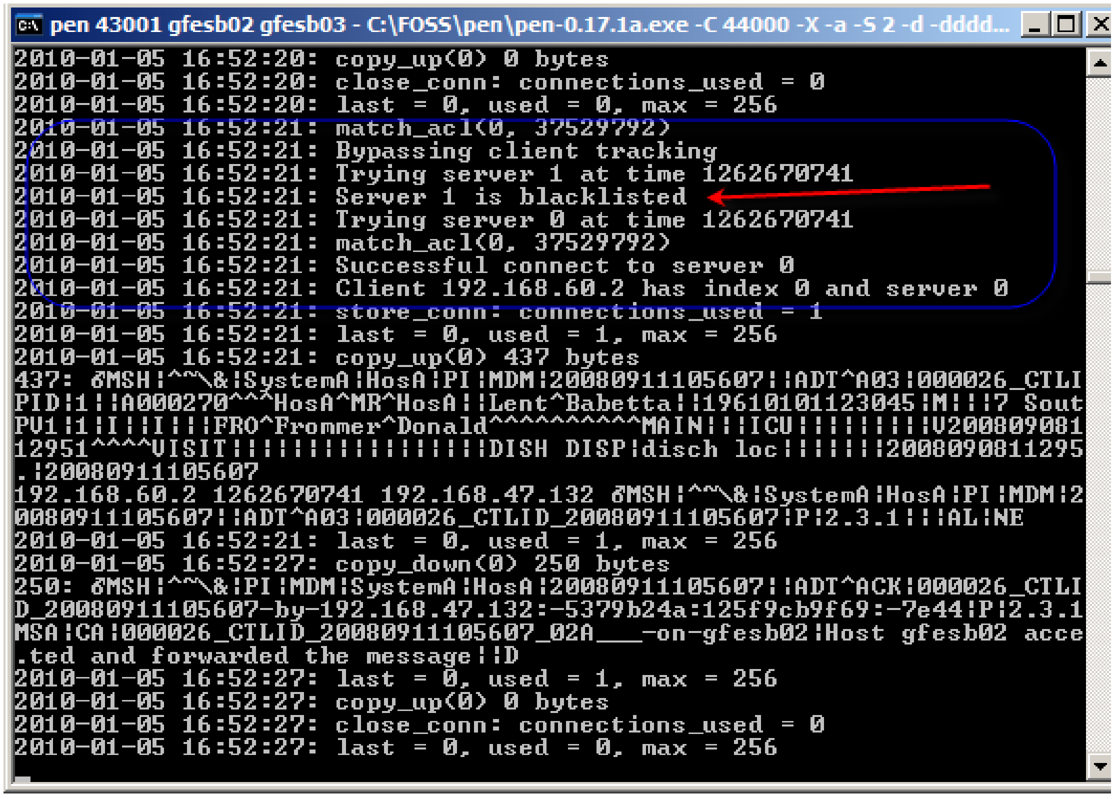
The messages are now being handled exclusively by gfesb02.



```
000010_CTLID_20080911035611_01MSG_-on-gfesb02_20100105-16-47-02-498.hl7
000010_CTLID_20080911035611_02A___-on-gfesb02_20100105-16-47-13-948.hl7
000011_CTLID_20080910054534_01MSG_-on-gfesb03_20100105-16-47-20-949.hl7
000011_CTLID_20080910054534_02A___-on-gfesb03_20100105-16-47-32-243.hl7
000012_CTLID_20080912071345_01MSG_-on-gfesb02_20100105-16-47-39-047.hl7
000012_CTLID_20080912071345_02A___-on-gfesb02_20100105-16-47-50-766.hl7
000013_CTLID_20080913095540_01MSG_-on-gfesb03_20100105-16-48-02-708.hl7
000013_CTLID_20080913095540_02A___-on-gfesb03_20100105-16-48-13-856.hl7
000014_CTLID_20080909015414_01MSG_-on-gfesb02_20100105-16-48-24-000.hl7
000014_CTLID_20080909015414_02A___-on-gfesb02_20100105-16-48-35-939.hl7
000015_CTLID_20080913042358_01MSG_-on-gfesb02_20100105-16-48-46-245.hl7
000015_CTLID_20080913042358_02A___-on-gfesb02_20100105-16-48-57-164.hl7
000016_CTLID_20080909030011_01MSG_-on-gfesb02_20100105-16-49-02-453.hl7
000016_CTLID_20080909030011_02A___-on-gfesb02_20100105-16-49-13-954.hl7
000017_CTLID_20080909021413_01MSG_-on-gfesb02_20100105-16-49-27-987.hl7
000017_CTLID_20080909021413_02A___-on-gfesb02_20100105-16-49-39-526.hl7
000018_CTLID_20080911012019_01MSG_-on-gfesb02_20100105-16-49-49-942.hl7
```

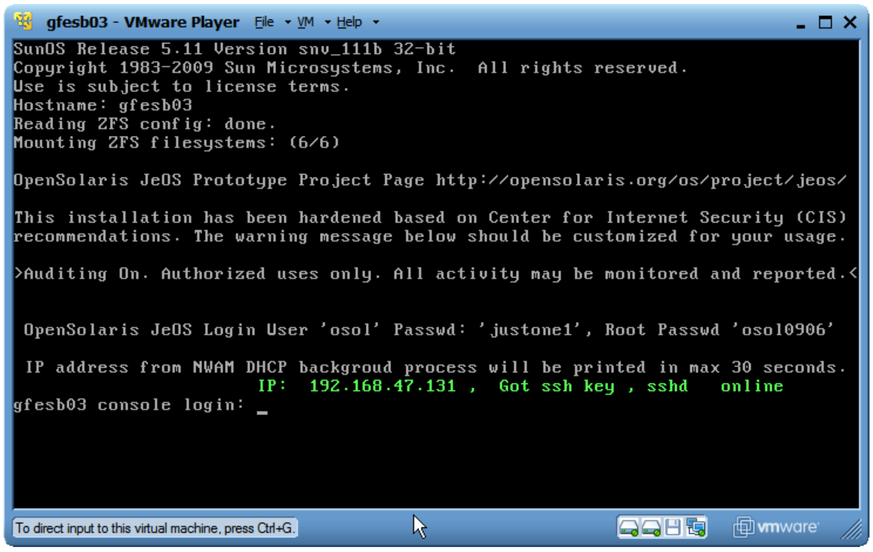
Output file names indicate no break in message sequence. No message loss occurred.

Every so often pen load balancer will try to connect to the failed server and if unsuccessful will blacklist it for the default 30 seconds before trying again.



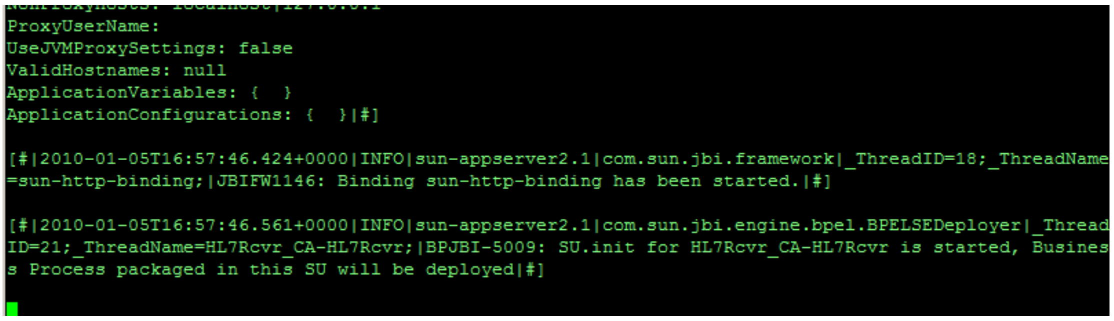
```
pen 43001 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -dddd...
2010-01-05 16:52:20: copy_up(0) 0 bytes
2010-01-05 16:52:20: close_conn: connections_used = 0
2010-01-05 16:52:20: last = 0, used = 0, max = 256
2010-01-05 16:52:21: match_acl(0, 37529792)
2010-01-05 16:52:21: Bypassing client tracking
2010-01-05 16:52:21: Trying server 1 at time 1262670741
2010-01-05 16:52:21: Server 1 is blacklisted
2010-01-05 16:52:21: Trying server 0 at time 1262670741
2010-01-05 16:52:21: match_acl(0, 37529792)
2010-01-05 16:52:21: Successful connect to server 0
2010-01-05 16:52:21: Client 192.168.60.2 has index 0 and server 0
2010-01-05 16:52:21: store_conn: connections_used = 1
2010-01-05 16:52:21: last = 0, used = 1, max = 256
2010-01-05 16:52:21: copy_up(0) 437 bytes
437: ♂MSH|^~\&|SystemA|HosA|PI|MDM|20080911105607||ADT^A03|000026_CTLI
PID|1||A000270^^^HosA^MR^HosA||Lent^Babetta||19610101123045|M|||7 Sout
PV1|1|I|||I|||FRO^Frommer^Donald^^^^^^^^^^MAIN|||ICU|||||||||V200809081
12951^^^^VISIT||||||||||||||||||DISH DISP|disch loc||||||20080908112951
.|20080911105607
192.168.60.2 1262670741 192.168.47.132 ♂MSH|^~\&|SystemA|HosA|PI|MDM|2
0080911105607||ADT^A03|000026_CTLID_20080911105607|P|2.3.1|||AL|NE
2010-01-05 16:52:21: last = 0, used = 1, max = 256
2010-01-05 16:52:27: copy_down(0) 250 bytes
250: ♂MSH|^~\&|PI|MDM|SystemA|HosA|20080911105607||ADT^ACK|000026_CTLI
D_20080911105607-by-192.168.47.132:-5379b24a:125f9cb9f69:-7e44|P|2.3.1
MSA|CA|000026_CTLID_20080911105607_02A___-on-gfesb02|Host gfesb02 acce
.ted and forwarded the message||D
2010-01-05 16:52:27: last = 0, used = 1, max = 256
2010-01-05 16:52:27: copy_up(0) 0 bytes
2010-01-05 16:52:27: close_conn: connections_used = 0
2010-01-05 16:52:27: last = 0, used = 0, max = 256
```

Let’s now boot gfesb03 to bring it back into service.



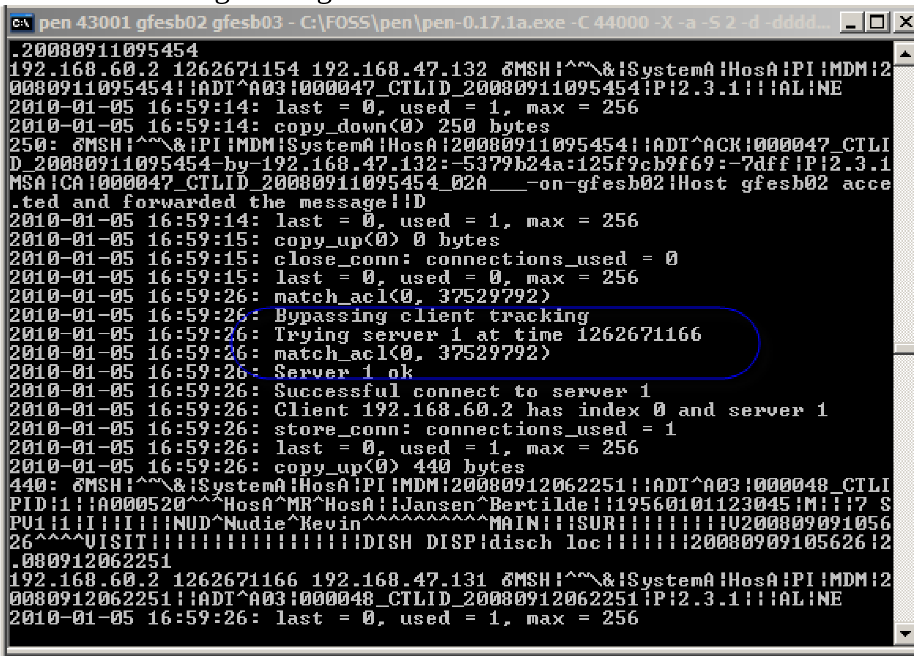
```
SunOS Release 5.11 Version snv_111b 32-bit
Copyright 1983-2009 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Hostname: gfesb03
Reading ZFS config: done.
Mounting ZFS filesystems: (6/6)

OpenSolaris JeOS Prototype Project Page http://opensolaris.org/os/project/jeos/

This installation has been hardened based on Center for Internet Security (CIS)
recommendations. The warning message below should be customized for your usage.

>Auditing On. Authorized uses only. All activity may be monitored and reported.<


 OpenSolaris JeOS Login User 'osol' Passwd: 'justone1', Root Passwd 'osol0906'

 IP address from NWAM DHCP backgroud process will be printed in max 30 seconds.
                       IP:  192.168.47.131 ,  Got ssh key , sshd   online
gfesb03 console login: _
```

Start a Putty/SSH Client session and tail server.log again, when the VMware Player console window shows the IP Address, to see where in the startup process the Application Server is. It will continue starting and deploying projects for a little while, before all are ready.



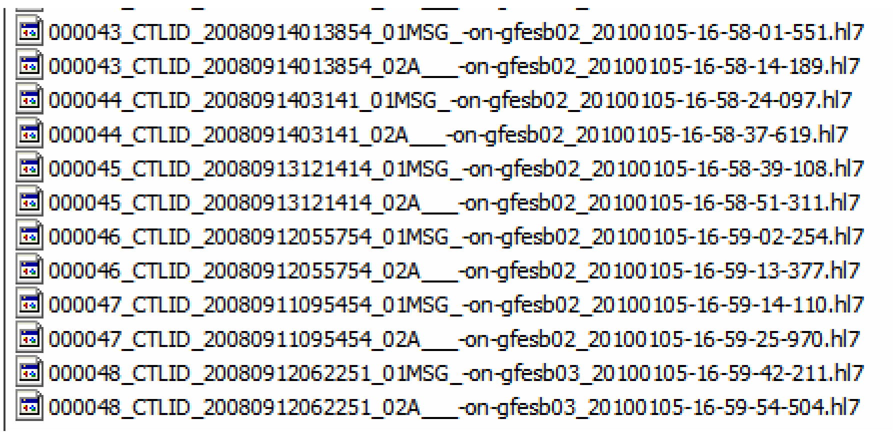
```
ProxyUserName:
UseJVMProxySettings: false
ValidHostnames: null
ApplicationVariables: {  }
ApplicationConfigurations: {  }|#]

[#|2010-01-05T16:57:46.424+0000|INFO|sun-appserver2.1|com.sun.jbi.framework|_ThreadID=18;_ThreadName
=sun-http-binding;|JBIFW1146: Binding sun-http-binding has been started.|#]

[#|2010-01-05T16:57:46.561+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.BPELSEDeployer|_Thread
ID=21;_ThreadName=HL7Rcvr_CA-HL7Rcvr;|BPJBI-5009: SU.init for HL7Rcvr_CA-HL7Rcvr is started, Busines
s Process packaged in this SU will be deployed|#]
```

Once the server becomes available and the load balancer retries, it gets a connection and starts sending messages.


pen 43001 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44000 -X -a -S 2 -d -dddd...

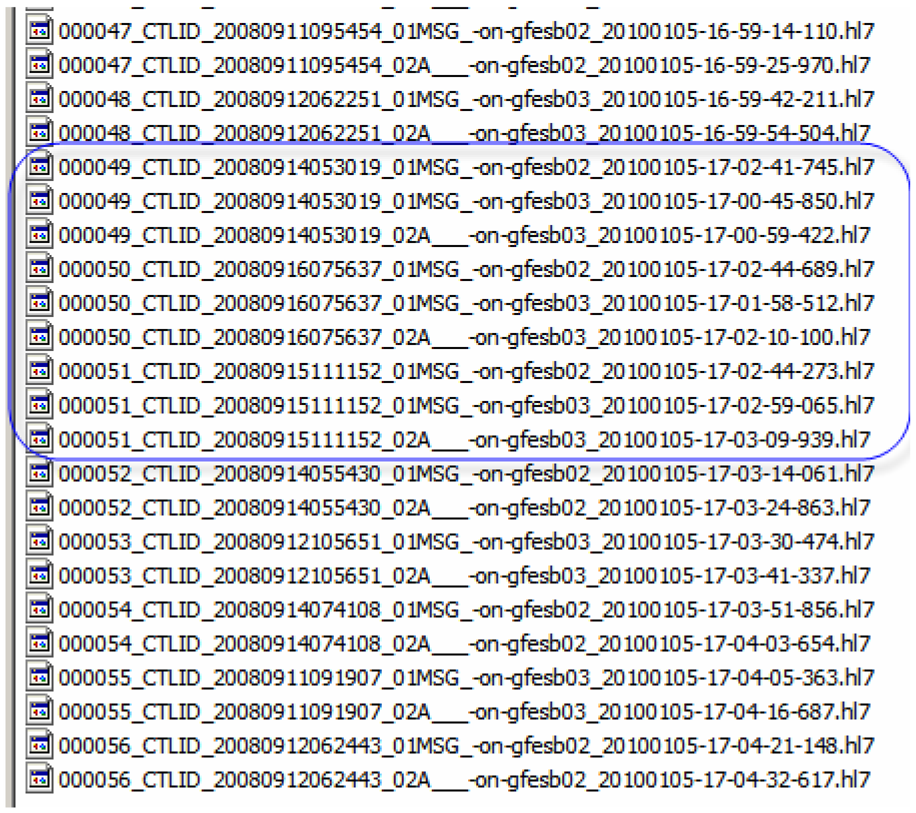
```
.20080911095454
192.168.60.2 1262671154 192.168.47.132 ♂MSH|^~\&|SystemA|HosA|PI|MDM|2
0080911095454||ADT^A03|000047_CTLID_20080911095454|P|2.3.1|||AL|NE
2010-01-05 16:59:14: last = 0, used = 1, max = 256
2010-01-05 16:59:14: copy_down(0) 250 bytes
250: ♂MSH|^~\&|PI|MDM|SystemA|HosA|20080911095454||ADT^ACK|000047_CTLI
D_20080911095454-by-192.168.47.132:-5379b24a:125f9cb9f69:-7dff|P|2.3.1
MSA|CA|000047_CTLID_20080911095454_02A___-on-gfesb02|Host gfesb02 acce
.ted and forwarded the message||D
2010-01-05 16:59:14: last = 0, used = 1, max = 256
2010-01-05 16:59:15: copy_up(0) 0 bytes
2010-01-05 16:59:15: close_conn: connections_used = 0
2010-01-05 16:59:15: last = 0, used = 0, max = 256
2010-01-05 16:59:26: match_acl(0, 37529792)
2010-01-05 16:59:26: Bypassing client tracking
2010-01-05 16:59:26: Trying server 1 at time 1262671166
2010-01-05 16:59:26: match_acl(0, 37529792)
2010-01-05 16:59:26: Server 1 ok
2010-01-05 16:59:26: Successful connect to server 1
2010-01-05 16:59:26: Client 192.168.60.2 has index 0 and server 1
2010-01-05 16:59:26: store_conn: connections_used = 1
2010-01-05 16:59:26: last = 0, used = 1, max = 256
2010-01-05 16:59:26: copy_up(0) 440 bytes
440: ♂MSH|^~\&|SystemA|HosA|PI|MDM|20080912062251||ADT^A03|000048_CTLI
PID|1||A000520^^^HosA^MR^HosA||Jansen^Bertilde||19560101123045|M|||7 S
PV1|1|I|||I|||NUD^Nudie^Kevin^^^^^^^^^^^MAIN|||SUR||||||||V200809091056
26^^^^VISIT||||||||||||||||||DISH DISP|disch loc||||||20080909105626|2
.080912062251
192.168.60.2 1262671166 192.168.47.131 ♂MSH|^~\&|SystemA|HosA|PI|MDM|2
0080912062251||ADT^A03|000048_CTLID_20080912062251|P|2.3.1|||AL|NE
2010-01-05 16:59:26: last = 0, used = 1, max = 256
```

The first message picked up by gfesb03, for my test, is message number 000048.



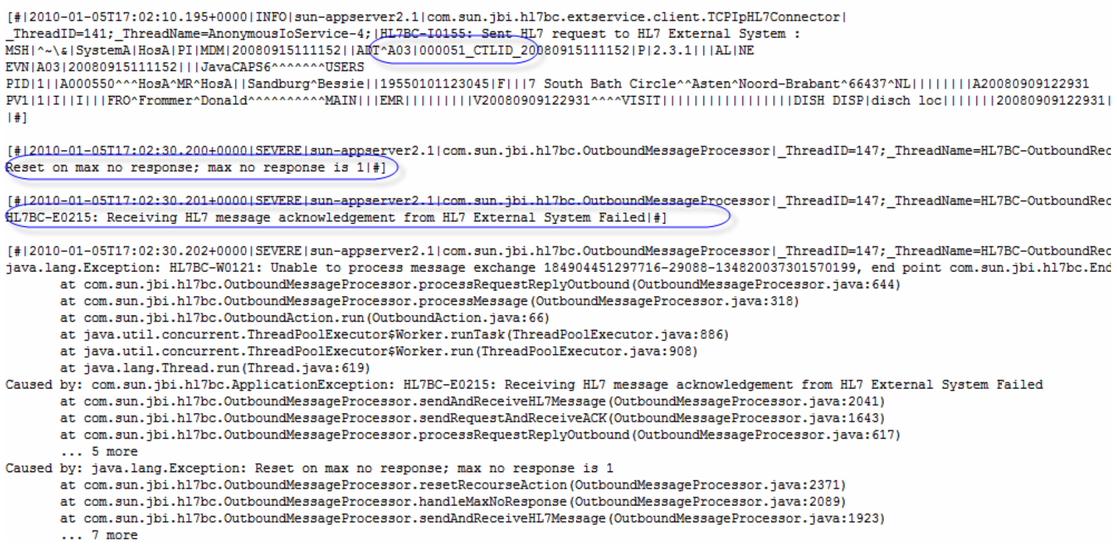
```
000043_CTLID_20080914013854_01MSG_-on-gfesb02_20100105-16-58-01-551.hl7
000043_CTLID_20080914013854_02A___-on-gfesb02_20100105-16-58-14-189.hl7
000044_CTLID_2008091403141_01MSG_-on-gfesb02_20100105-16-58-24-097.hl7
000044_CTLID_2008091403141_02A___-on-gfesb02_20100105-16-58-37-619.hl7
000045_CTLID_20080913121414_01MSG_-on-gfesb02_20100105-16-58-39-108.hl7
000045_CTLID_20080913121414_02A___-on-gfesb02_20100105-16-58-51-311.hl7
000046_CTLID_20080912055754_01MSG_-on-gfesb02_20100105-16-59-02-254.hl7
000046_CTLID_20080912055754_02A___-on-gfesb02_20100105-16-59-13-377.hl7
000047_CTLID_20080911095454_01MSG_-on-gfesb02_20100105-16-59-14-110.hl7
000047_CTLID_20080911095454_02A___-on-gfesb02_20100105-16-59-25-970.hl7
000048_CTLID_20080912062251_01MSG_-on-gfesb03_20100105-16-59-42-211.hl7
000048_CTLID_20080912062251_02A___-on-gfesb03_20100105-16-59-54-504.hl7
```

Once things settle down, message processing returns to the round robin pattern, alternating between gfesb02 and gfesb03.



```
000047_CTLID_20080911095454_01MSG_-on-gfesb02_20100105-16-59-14-110.hl7
000047_CTLID_20080911095454_02A___-on-gfesb02_20100105-16-59-25-970.hl7
000048_CTLID_20080912062251_01MSG_-on-gfesb03_20100105-16-59-42-211.hl7
000048_CTLID_20080912062251_02A___-on-gfesb03_20100105-16-59-54-504.hl7
000049_CTLID_20080914053019_01MSG_-on-gfesb02_20100105-17-02-41-745.hl7
000049_CTLID_20080914053019_01MSG_-on-gfesb03_20100105-17-00-45-850.hl7
000049_CTLID_20080914053019_02A___-on-gfesb03_20100105-17-00-59-422.hl7
000050_CTLID_20080916075637_01MSG_-on-gfesb02_20100105-17-02-44-689.hl7
000050_CTLID_20080916075637_01MSG_-on-gfesb03_20100105-17-01-58-512.hl7
000050_CTLID_20080916075637_02A___-on-gfesb03_20100105-17-02-10-100.hl7
000051_CTLID_20080915111152_01MSG_-on-gfesb02_20100105-17-02-44-273.hl7
000051_CTLID_20080915111152_01MSG_-on-gfesb03_20100105-17-02-59-065.hl7
000051_CTLID_20080915111152_02A___-on-gfesb03_20100105-17-03-09-939.hl7
000052_CTLID_20080914055430_01MSG_-on-gfesb02_20100105-17-03-14-061.hl7
000052_CTLID_20080914055430_02A___-on-gfesb02_20100105-17-03-24-863.hl7
000053_CTLID_20080912105651_01MSG_-on-gfesb03_20100105-17-03-30-474.hl7
000053_CTLID_20080912105651_02A___-on-gfesb03_20100105-17-03-41-337.hl7
000054_CTLID_20080914074108_01MSG_-on-gfesb02_20100105-17-03-51-856.hl7
000054_CTLID_20080914074108_02A___-on-gfesb02_20100105-17-04-03-654.hl7
000055_CTLID_20080911091907_01MSG_-on-gfesb03_20100105-17-04-05-363.hl7
000055_CTLID_20080911091907_02A___-on-gfesb03_20100105-17-04-16-687.hl7
000056_CTLID_20080912062443_01MSG_-on-gfesb02_20100105-17-04-21-148.hl7
000056_CTLID_20080912062443_02A___-on-gfesb02_20100105-17-04-32-617.hl7
```

Notice, in the listing above, that messages 000049, 000050 and 000051 were delivered twice. The time taken to deliver acknowledgements was exceeded and the HL7Sndr timed out and re-sent the message. In my severely memory-constrained environment this issue is easy to induce. In an environment which is not as constrained this should be a rare occurrence.

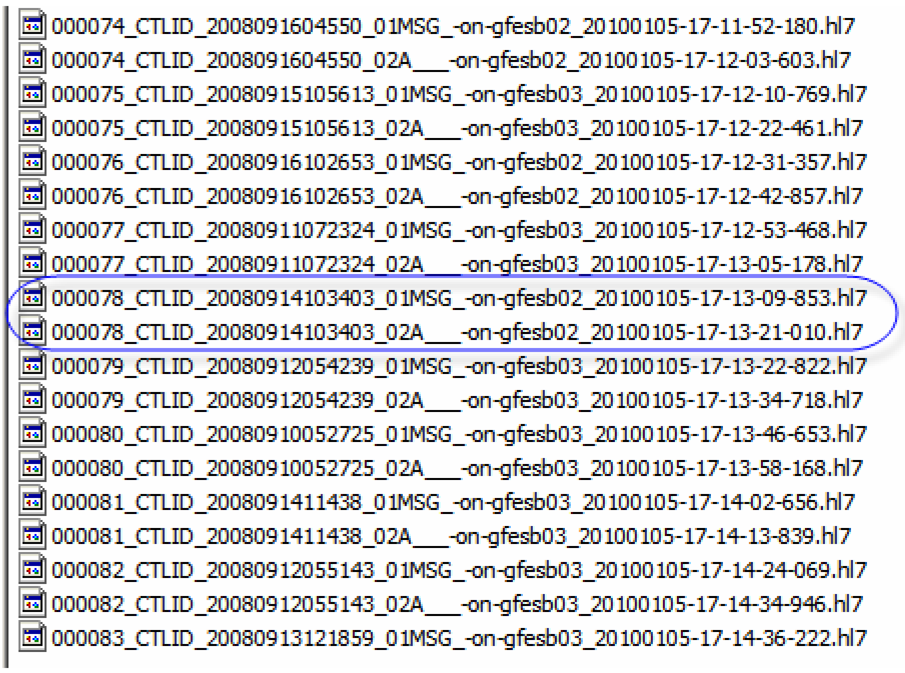


```
[#|2010-01-05T17:02:10.195+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.client.TCPIpHL7Connector|
_ThreadID=141;_ThreadName=AnonymousIoService-4;|HL7BC-I0155: Sent HL7 request to HL7 External System :
MSH|^~\&|SystemA|HosA|PI|MDM|20080915111152||ADT^A03|000051_CTLID_20080915111152|P|2.3.1|||AL|NE
EVN|A03|20080915111152|||JavaCAPS6^^^^^^^USERS
PID|1||A000550^^^HosA^MR^HosA||Sandburg^Bessie||19550101123045|F|||7 South Bath Circle^^Asten^Noord-Brabant^66437^NL|||||||A20080909122931
PV1|1|I||I|||FRO^Frommer^Donald^^^^^^^^^^MAIN|||EMR||||||||V20080909122931^^^^VISIT||||||||||||||||DISH DISP|disch loc||||||20080909122931|
|#]

[#|2010-01-05T17:02:30.200+0000|SEVERE|sun-appserver2.1|com.sun.jbi.hl7bc.OutboundMessageProcessor|_ThreadID=147;_ThreadName=HL7BC-OutboundRec
Reset on max no response; max no response is 1|#]

[#|2010-01-05T17:02:30.201+0000|SEVERE|sun-appserver2.1|com.sun.jbi.hl7bc.OutboundMessageProcessor|_ThreadID=147;_ThreadName=HL7BC-OutboundRec
HL7BC-E0215: Receiving HL7 message acknowledgement from HL7 External System Failed|#]

[#|2010-01-05T17:02:30.202+0000|SEVERE|sun-appserver2.1|com.sun.jbi.hl7bc.OutboundMessageProcessor|_ThreadID=147;_ThreadName=HL7BC-OutboundRec
java.lang.Exception: HL7BC-W0121: Unable to process message exchange 184904451297716-29088-134820037301570199, end point com.sun.jbi.hl7bc.End
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.processRequestReplyOutbound(OutboundMessageProcessor.java:644)
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.processMessage(OutboundMessageProcessor.java:318)
        at com.sun.jbi.hl7bc.OutboundAction.run(OutboundAction.java:66)
        at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
        at java.lang.Thread.run(Thread.java:619)
Caused by: com.sun.jbi.hl7bc.ApplicationException: HL7BC-E0215: Receiving HL7 message acknowledgement from HL7 External System Failed
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.sendAndReceiveHL7Message(OutboundMessageProcessor.java:2041)
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.sendRequestAndReceiveACK(OutboundMessageProcessor.java:1643)
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.processRequestReplyOutbound(OutboundMessageProcessor.java:617)
        ... 5 more
Caused by: java.lang.Exception: Reset on max no response; max no response is 1
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.resetRecourseAction(OutboundMessageProcessor.java:2371)
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.handleMaxNoResponse(OutboundMessageProcessor.java:2089)
        at com.sun.jbi.hl7bc.OutboundMessageProcessor.sendAndReceiveHL7Message(OutboundMessageProcessor.java:1923)
        ... 7 more
```

This highlights the importance of correctly calculating timing parameters and the consequences of message processing taking longer then expected. In these cases messages were duplicated, but none was lots. Whether this is acceptable depends on the nature of the solution and its ability to cope with duplicate messages.

Let’s now crash gfesb02 to make sure that gfesb0-3 will pick up and complete processing.

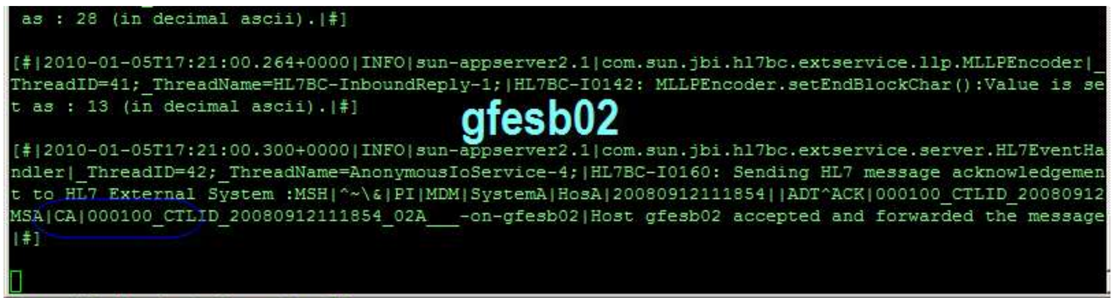
The last message gfesb02 processed, in my test, was 000078. Following messages are processed by gfesb03.



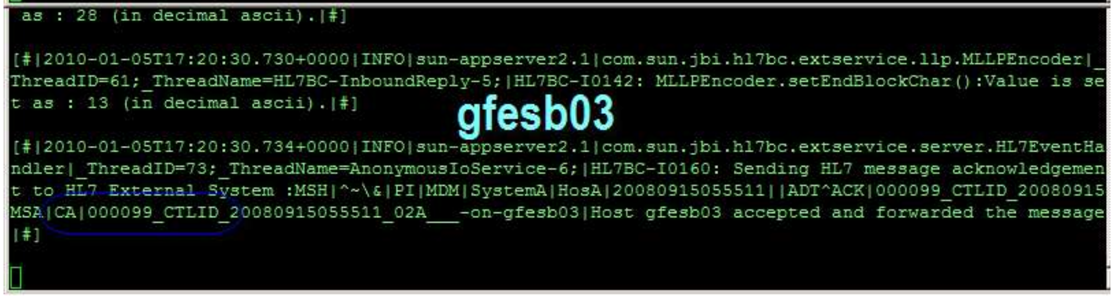
```
000074_CTLID_2008091604550_01MSG_-on-gfesb02_20100105-17-11-52-180.hl7
000074_CTLID_2008091604550_02A___-on-gfesb02_20100105-17-12-03-603.hl7
000075_CTLID_20080915105613_01MSG_-on-gfesb03_20100105-17-12-10-769.hl7
000075_CTLID_20080915105613_02A___-on-gfesb03_20100105-17-12-22-461.hl7
000076_CTLID_20080916102653_01MSG_-on-gfesb02_20100105-17-12-31-357.hl7
000076_CTLID_20080916102653_02A___-on-gfesb02_20100105-17-12-42-857.hl7
000077_CTLID_20080911072324_01MSG_-on-gfesb03_20100105-17-12-53-468.hl7
000077_CTLID_20080911072324_02A___-on-gfesb03_20100105-17-13-05-178.hl7
000078_CTLID_20080914103403_01MSG_-on-gfesb02_20100105-17-13-09-853.hl7
000078_CTLID_20080914103403_02A___-on-gfesb02_20100105-17-13-21-010.hl7
000079_CTLID_20080912054239_01MSG_-on-gfesb03_20100105-17-13-22-822.hl7
000079_CTLID_20080912054239_02A___-on-gfesb03_20100105-17-13-34-718.hl7
000080_CTLID_20080910052725_01MSG_-on-gfesb03_20100105-17-13-46-653.hl7
000080_CTLID_20080910052725_02A___-on-gfesb03_20100105-17-13-58-168.hl7
000081_CTLID_2008091411438_01MSG_-on-gfesb03_20100105-17-14-02-656.hl7
000081_CTLID_2008091411438_02A___-on-gfesb03_20100105-17-14-13-839.hl7
000082_CTLID_20080912055143_01MSG_-on-gfesb03_20100105-17-14-24-069.hl7
000082_CTLID_20080912055143_02A___-on-gfesb03_20100105-17-14-34-946.hl7
000083_CTLID_20080913121859_01MSG_-on-gfesb03_20100105-17-14-36-222.hl7
```

Let’s now boot gfesb02 in preparation for the Web Service (HTTP BC-based) test.

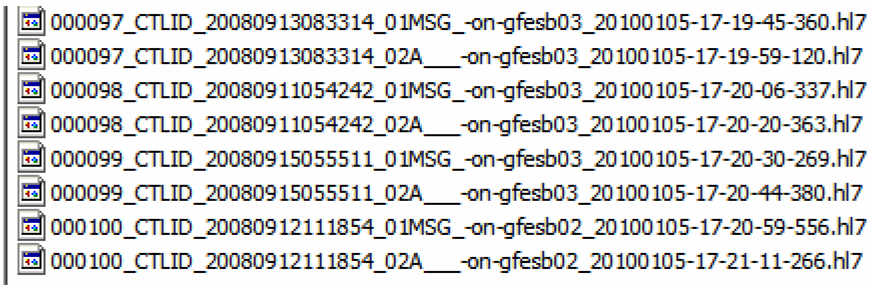
In my test gfesb02 started in time to join the fry and process the last message.



```
 as : 28 (in decimal ascii).|#]

[#|2010-01-05T17:21:00.264+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.llp.MLLPEncoder|_
ThreadID=41;_ThreadName=HL7BC-InboundReply-1;|HL7BC-I0142: MLLPEncoder.setEndBlockChar():Value is se
t as : 13 (in decimal ascii).|#]

[#|2010-01-05T17:21:00.300+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.server.HL7EventHa
ndler|_ThreadID=42;_ThreadName=AnonymousIoService-4;|HL7BC-I0160: Sending HL7 message acknowledgemen
t to HL7 External System :MSH|^~\&|PI|MDM|SystemA|HosA|20080912111854||ADT^ACK|000100_CTLID_20080912
MSA|CA|000100_CTLID_20080912111854_02A___-on-gfesb02|Host gfesb02 accepted and forwarded the message
|#]
```



```
 as : 28 (in decimal ascii).|#]

[#|2010-01-05T17:20:30.730+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.llp.MLLPEncoder|_
ThreadID=61;_ThreadName=HL7BC-InboundReply-5;|HL7BC-I0142: MLLPEncoder.setEndBlockChar():Value is se
t as : 13 (in decimal ascii).|#]

[#|2010-01-05T17:20:30.734+0000|INFO|sun-appserver2.1|com.sun.jbi.hl7bc.extservice.server.HL7EventHa
ndler|_ThreadID=73;_ThreadName=AnonymousIoService-6;|HL7BC-I0160: Sending HL7 message acknowledgemen
t to HL7 External System :MSH|^~\&|PI|MDM|SystemA|HosA|20080915055511||ADT^ACK|000099_CTLID_20080915
MSA|CA|000099_CTLID_20080915055511_02A___-on-gfesb03|Host gfesb03 accepted and forwarded the message
|#]
```



```
000097_CTLID_20080913083314_01MSG_-on-gfesb03_20100105-17-19-45-360.hl7
000097_CTLID_20080913083314_02A___-on-gfesb03_20100105-17-19-59-120.hl7
000098_CTLID_20080911054242_01MSG_-on-gfesb03_20100105-17-20-06-337.hl7
000098_CTLID_20080911054242_02A___-on-gfesb03_20100105-17-20-20-363.hl7
000099_CTLID_20080915055511_01MSG_-on-gfesb03_20100105-17-20-30-269.hl7
000099_CTLID_20080915055511_02A___-on-gfesb03_20100105-17-20-44-380.hl7
000100_CTLID_20080912111854_01MSG_-on-gfesb02_20100105-17-20-59-556.hl7
000100_CTLID_20080912111854_02A___-on-gfesb02_20100105-17-21-11-266.hl7
```

The test is finished.

We are, hopefully, convinced that load balancing and fail-over without message loss, work for a solution using the HL7 BC.

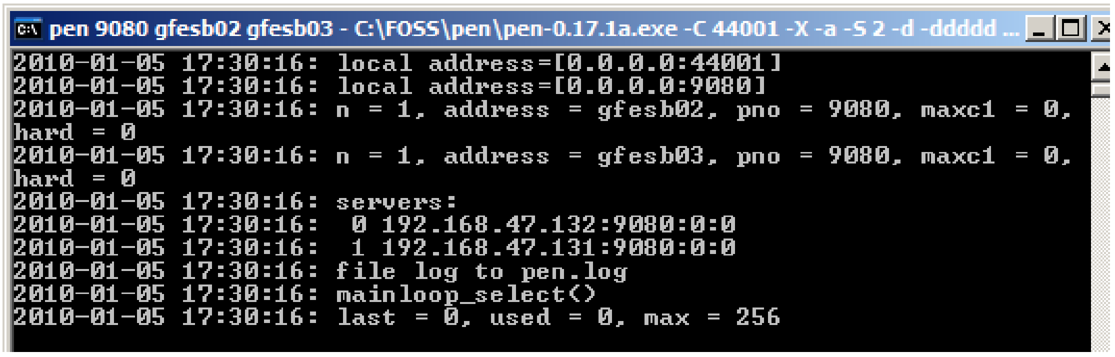
We are also convinced that this configuration provides high availability and facilitates horizontal scaling (addition of more receivers). Receiver hosts can be added and removed without interrupting message flow.

We know that while messages will not be lost, in certain circumstances they can be duplicated. This must be must be accounted for in solutions that use this kind of technology for high availability.

Testing resilience of the HTTP BC-based Solution (Web Service)

We will start with all machines and all components started and processing messages. We expect messages to be processed in a round robin manner, alternating between gfesb02 and gfesb03. Once 15 or so messages are processed, we will "crash" gfesb03 by closing the VMware Player window in which it runs. This will cause retry functionality, after timeout period, to be invoked. We expect message flow to continue without message loss and we expect messages to now be processed by gfesb02 only. This should be reflected in names of the output files. Once we reach 30 or so messages, we will boot gfesb03 again and wait for it to start the GlassFish Application Server and deploy all the receivers. As soon as the WSRcvr has been deployed and the most recent load balancer blacklist interval for the gfesb03 expired, we expect to see gfesb03 to start picking up messages again. We expect this to also be reflected in the names of the output files.

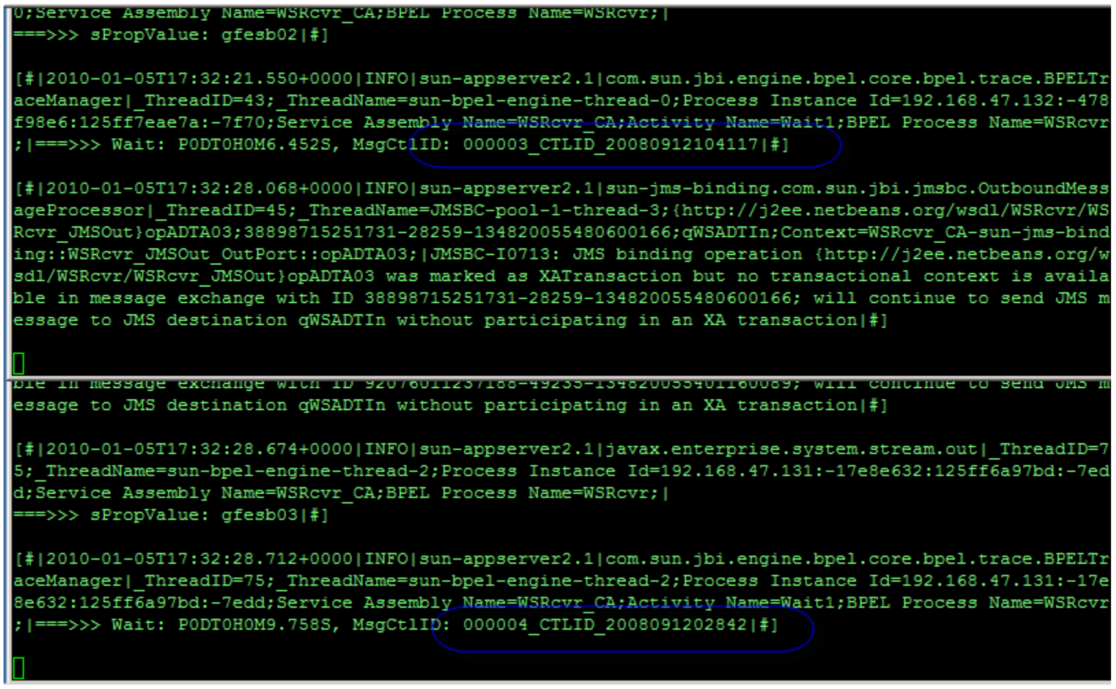
The pen load balancer considers gfesb02 to be server 0 and gfesb03 to be server 1.


pen 9080 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44001 -X -a -S 2 -d -ddddd ...

```
2010-01-05 17:30:16: local address=[0.0.0.0:44001]
2010-01-05 17:30:16: local address=[0.0.0.0:9080]
2010-01-05 17:30:16: n = 1, address = gfesb02, pno = 9080, maxc1 = 0,
hard = 0
2010-01-05 17:30:16: n = 1, address = gfesb03, pno = 9080, maxc1 = 0,
hard = 0
2010-01-05 17:30:16: servers:
2010-01-05 17:30:16:  0 192.168.47.132:9080:0:0
2010-01-05 17:30:16:  1 192.168.47.131:9080:0:0
2010-01-05 17:30:16: file log to pen.log
2010-01-05 17:30:16: mainloop_select()
2010-01-05 17:30:16: last = 0, used = 0, max = 256
```

Let's begin by copying the file ADT_A03_output_101.hl7, containing 101 ADT A03 messages, to the data_ws directory, and observing the console windows and the output directory until around 15 messages are processed.

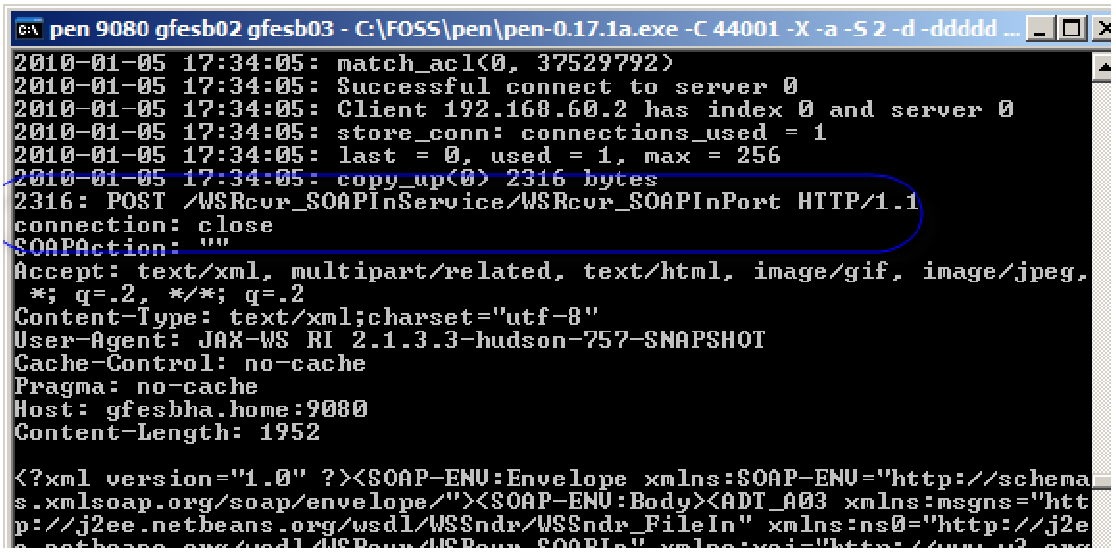
Message processing commences and alternates between gfesb02 and gfesb03.



```
0;Service Assembly Name=WSRcvr_CA;BPEL Process Name=WSRcvr;|
===>>> sPropValue: gfesb02|#]

[#|2010-01-05T17:32:21.550+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=43;_ThreadName=sun-bpel-engine-thread-0;Process Instance Id=192.168.47.132:-478
f98e6:125ff7eae7a:-7f70;Service Assembly Name=WSRcvr_CA;Activity Name=Wait1;BPEL Process Name=WSRcvr
;|===>>> Wait: P0DT0H0M6.452S, MsgCtlID: 000003_CTLID_20080912104117|#]

[#|2010-01-05T17:32:28.068+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.OutboundMess
ageProcessor|_ThreadID=45;_ThreadName=JMSBC-pool-1-thread-3;{http://j2ee.netbeans.org/wsdl/WSRcvr/WS
Rcvr_JMSOut}opADTA03;38898715251731-28259-134820055480600166;qWSADTIn;Context=WSRcvr_CA-sun-jms-bind
ing::WSRcvr_JMSOut_OutPort::opADTA03;|JMSBC-I0713: JMS binding operation {http://j2ee.netbeans.org/w
sdl/WSRcvr/WSRcvr_JMSOut}opADTA03 was marked as XATransaction but no transactional context is availa
ble in message exchange with ID 38898715251731-28259-134820055480600166; will continue to send JMS m
essage to JMS destination qWSADTIn without participating in an XA transaction|#]
```

```
ble in message exchange with ID 92076011237188-49235-134820055401160089; will continue to send JMS m
essage to JMS destination qWSADTIn without participating in an XA transaction|#]

[#|2010-01-05T17:32:28.674+0000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=7
5;_ThreadName=sun-bpel-engine-thread-2;Process Instance Id=192.168.47.131:-17e8e632:125ff6a97bd:-7ed
d;Service Assembly Name=WSRcvr_CA;BPEL Process Name=WSRcvr;|
===>>> sPropValue: gfesb03|#]

[#|2010-01-05T17:32:28.712+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=75;_ThreadName=sun-bpel-engine-thread-2;Process Instance Id=192.168.47.131:-17e
8e632:125ff6a97bd:-7edd;Service Assembly Name=WSRcvr_CA;Activity Name=Wait1;BPEL Process Name=WSRcvr
;|===>>> Wait: P0DT0H0M9.758S, MsgCtlID: 000004_CTLID_2008091202842|#]
```

The load balancer sees "connection: close" HTTP header, which was set in the WSSndr BPEL process, and closes the connection after each reply. This is what facilitates round robin processing.

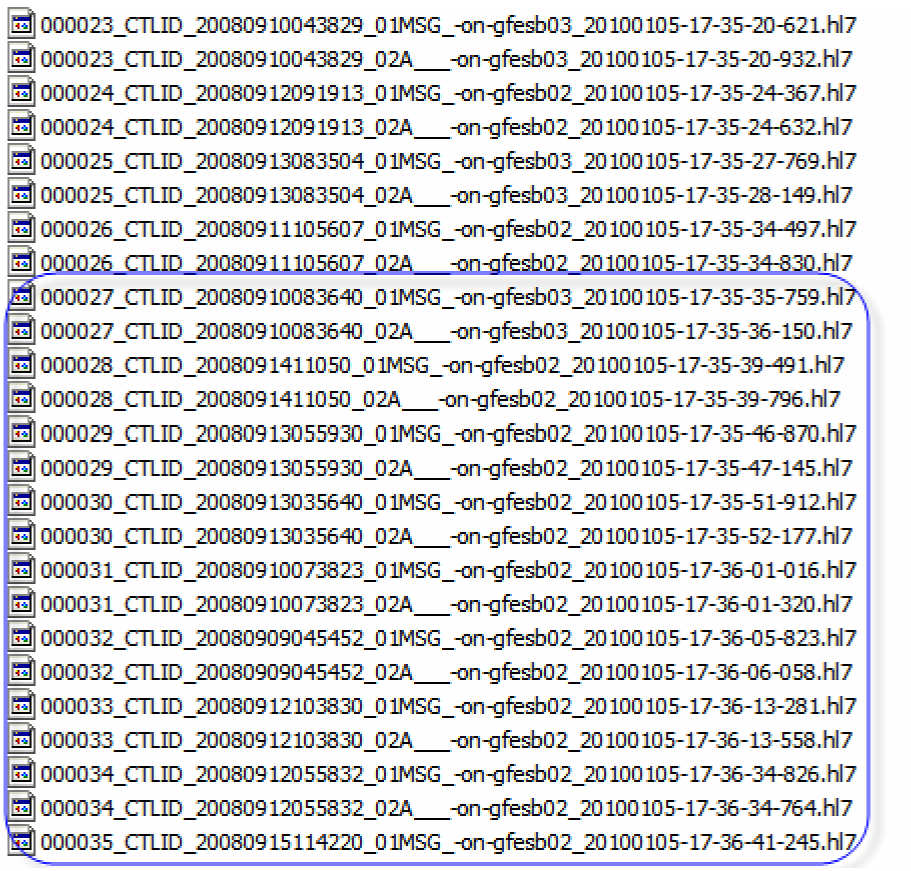


pen 9080 gfesb02 gfesb03 - C:\FOSS\pen\pen-0.17.1a.exe -C 44001 -X -a -S 2 -d -ddddd ...

```
2010-01-05 17:34:05: match_acl(0, 37529792)
2010-01-05 17:34:05: Successful connect to server 0
2010-01-05 17:34:05: Client 192.168.60.2 has index 0 and server 0
2010-01-05 17:34:05: store_conn: connections_used = 1
2010-01-05 17:34:05: last = 0, used = 1, max = 256
2010-01-05 17:34:05: copy_up(0) 2316 bytes
2316: POST /WSRcvr_SOAPInService/WSRcvr_SOAPInPort HTTP/1.1
connection: close
SOAPAction: ""
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
 *; q=.2, */*; q=.2
Content-Type: text/xml;charset="utf-8"
User-Agent: JAX-WS RI 2.1.3.3-hudson-757-SNAPSHOT
Cache-Control: no-cache
Pragma: no-cache
Host: gfesbha.home:9080
Content-Length: 1952

<?xml version="1.0" ?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema
s.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body><ADT_A03 xmlns:msgns="htt
p://j2ee.netbeans.org/wsdl/WSSndr/WSSndr_FileIn" xmlns:ns0="http://j2e
```

We now crash gfesb03 and see what files are being written to the output directory – this is our primary indication of correctness of message processing.

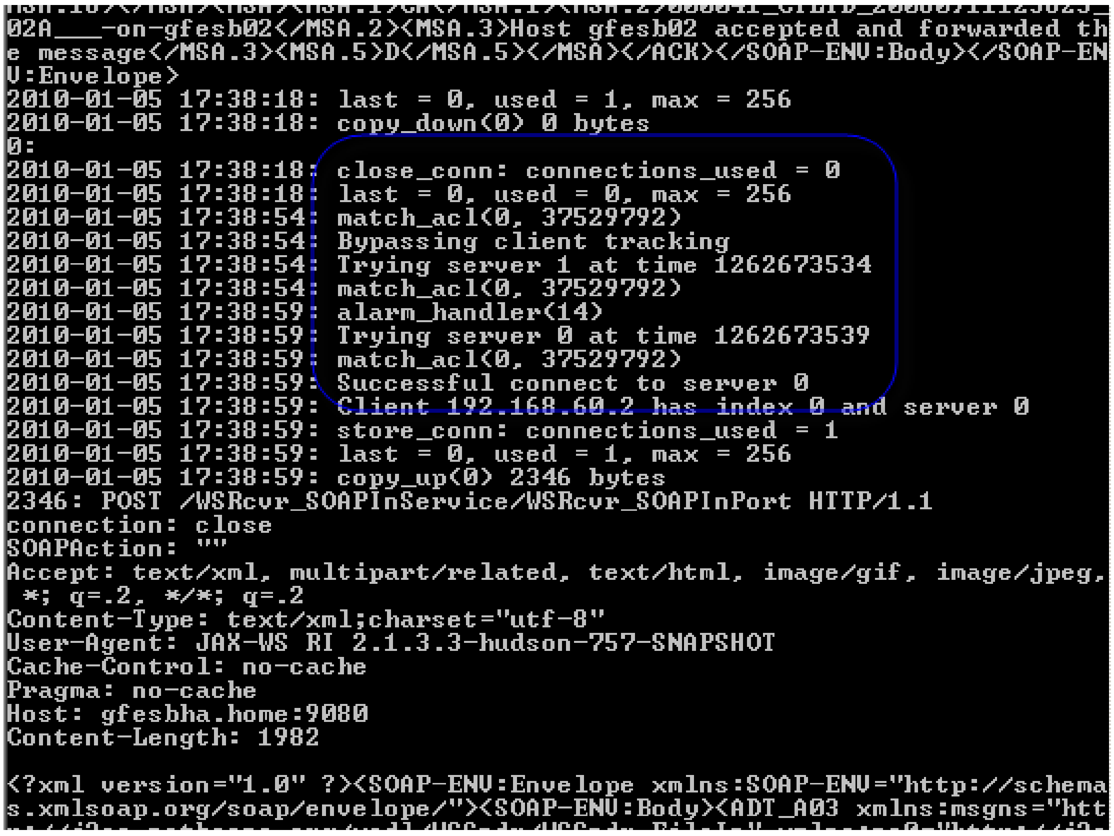
For me all messages following message 000027 were processed by the surviving gfesb02.



```
000023_CTLID_20080910043829_01MSG_-on-gfesb03_20100105-17-35-20-621.hl7
000023_CTLID_20080910043829_02A___-on-gfesb03_20100105-17-35-20-932.hl7
000024_CTLID_20080912091913_01MSG_-on-gfesb02_20100105-17-35-24-367.hl7
000024_CTLID_20080912091913_02A___-on-gfesb02_20100105-17-35-24-632.hl7
000025_CTLID_20080913083504_01MSG_-on-gfesb03_20100105-17-35-27-769.hl7
000025_CTLID_20080913083504_02A___-on-gfesb03_20100105-17-35-28-149.hl7
000026_CTLID_20080911105607_01MSG_-on-gfesb02_20100105-17-35-34-497.hl7
000026_CTLID_20080911105607_02A___-on-gfesb02_20100105-17-35-34-830.hl7
000027_CTLID_20080910083640_01MSG_-on-gfesb03_20100105-17-35-35-759.hl7
000027_CTLID_20080910083640_02A___-on-gfesb03_20100105-17-35-36-150.hl7
000028_CTLID_200809141105O_01MSG_-on-gfesb02_20100105-17-35-39-491.hl7
000028_CTLID_200809141105O_02A___-on-gfesb02_20100105-17-35-39-796.hl7
000029_CTLID_20080913055930_01MSG_-on-gfesb02_20100105-17-35-46-870.hl7
000029_CTLID_20080913055930_02A___-on-gfesb02_20100105-17-35-47-145.hl7
000030_CTLID_20080913035640_01MSG_-on-gfesb02_20100105-17-35-51-912.hl7
000030_CTLID_20080913035640_02A___-on-gfesb02_20100105-17-35-52-177.hl7
000031_CTLID_20080910073823_01MSG_-on-gfesb02_20100105-17-36-01-016.hl7
000031_CTLID_20080910073823_02A___-on-gfesb02_20100105-17-36-01-320.hl7
000032_CTLID_20080909045452_01MSG_-on-gfesb02_20100105-17-36-05-823.hl7
000032_CTLID_20080909045452_02A___-on-gfesb02_20100105-17-36-06-058.hl7
000033_CTLID_20080912103830_01MSG_-on-gfesb02_20100105-17-36-13-281.hl7
000033_CTLID_20080912103830_02A___-on-gfesb02_20100105-17-36-13-558.hl7
000034_CTLID_20080912055832_01MSG_-on-gfesb02_20100105-17-36-34-826.hl7
000034_CTLID_20080912055832_02A___-on-gfesb02_20100105-17-36-34-764.hl7
000035_CTLID_20080915114220_01MSG_-on-gfesb02_20100105-17-36-41-245.hl7
```

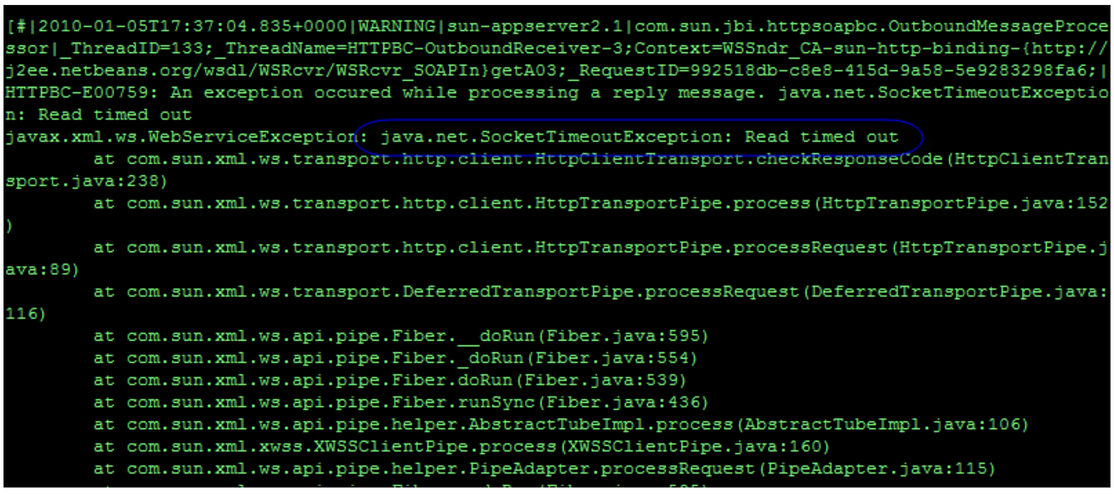
Let’s boot gfesb03 and see it picks up the workload.

While gfesb03 is not available the load balancer tries to connect to it every 60 seconds to see if it becomes available.



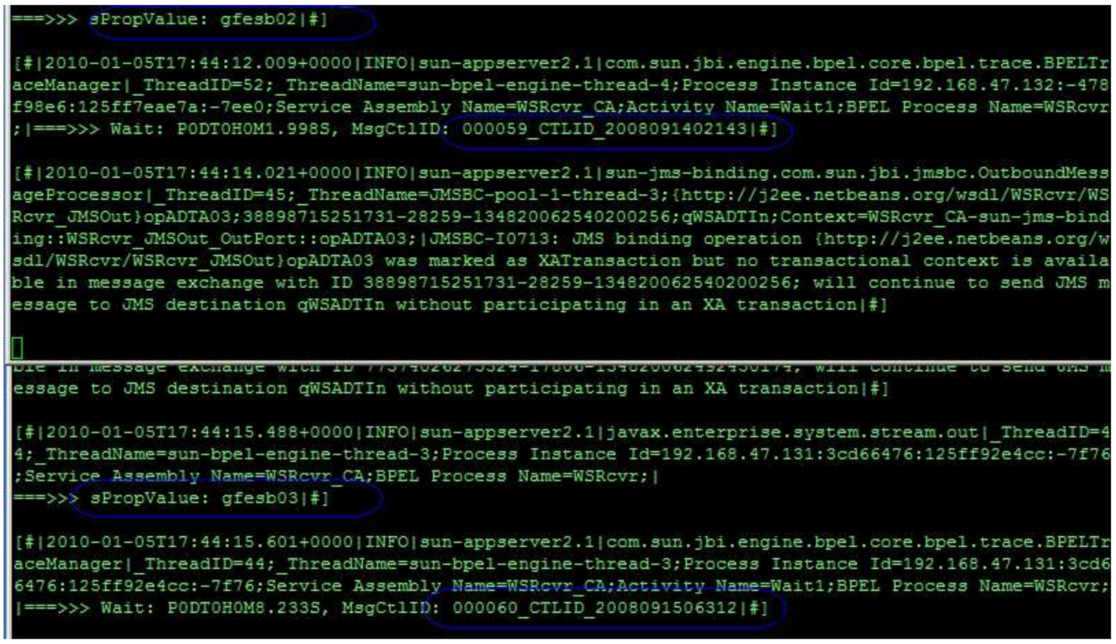
```
02A___-on-gfesb02</MSA.2><MSA.3>Host gfesb02 accepted and forwarded th
e message</MSA.3><MSA.5>D</MSA.5></MSA></ACK></SOAP-ENV:Body></SOAP-EN
V:Envelope>
2010-01-05 17:38:18: last = 0, used = 1, max = 256
2010-01-05 17:38:18: copy_down(0) 0 bytes
0:
2010-01-05 17:38:18: close_conn: connections_used = 0
2010-01-05 17:38:18: last = 0, used = 0, max = 256
2010-01-05 17:38:54: match_acl(0, 37529792)
2010-01-05 17:38:54: Bypassing client tracking
2010-01-05 17:38:54: Trying server 1 at time 1262673534
2010-01-05 17:38:54: match_acl(0, 37529792)
2010-01-05 17:38:59: alarm_handler(14)
2010-01-05 17:38:59: Trying server 0 at time 1262673539
2010-01-05 17:38:59: match_acl(0, 37529792)
2010-01-05 17:38:59: Successful connect to server 0
2010-01-05 17:38:59: Client 192.168.60.2 has index 0 and server 0
2010-01-05 17:38:59: store_conn: connections_used = 1
2010-01-05 17:38:59: last = 0, used = 1, max = 256
2010-01-05 17:38:59: copy_up(0) 2346 bytes
2346: POST /WSRcvr_SOAPInService/WSRcvr_SOAPInPort HTTP/1.1
connection: close
SOAPAction: ""
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
 *; q=.2, */*; q=.2
Content-Type: text/xml;charset="utf-8"
User-Agent: JAX-WS RI 2.1.3.3-hudson-757-SNAPSHOT
Cache-Control: no-cache
Pragma: no-cache
Host: gfesbha.home:9080
Content-Length: 1982

<?xml version="1.0" ?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema
s.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body><ADT_A03 xmlns:msgns="htt
```

While gfesb03 boots and starts deploying applications it may be in a state which convinces the WSSndr (HTTP BC) that it is available for processing messages, where in fact it is not. The solution we are exercising copes with this situation in the same way as it does with the server not being there at all – it times out and retries.



```
[#|2010-01-05T17:37:04.835+0000|WARNING|sun-appserver2.1|com.sun.jbi.httpsoapbc.OutboundMessageProce
ssor|_ThreadID=133;_ThreadName=HTTPBC-OutboundReceiver-3;Context=WSSndr_CA-sun-http-binding-{http://
j2ee.netbeans.org/wsdl/WSRcvr/WSRcvr_SOAPIn}getA03;_RequestID=992518db-c8e8-415d-9a58-5e9283298fa6;|
HTTPBC-E00759: An exception occured while processing a reply message. java.net.SocketTimeoutExceptio
n: Read timed out
javax.xml.ws.WebServiceException: java.net.SocketTimeoutException: Read timed out
        at com.sun.xml.ws.transport.http.client.HttpClientTransport.checkResponseCode(HttpClientTran
sport.java:238)
        at com.sun.xml.ws.transport.http.client.HttpTransportPipe.process(HttpTransportPipe.java:152
)
        at com.sun.xml.ws.transport.http.client.HttpTransportPipe.processRequest(HttpTransportPipe.j
ava:89)
        at com.sun.xml.ws.transport.DeferredTransportPipe.processRequest(DeferredTransportPipe.java:
116)
        at com.sun.xml.ws.api.pipe.Fiber.__doRun(Fiber.java:595)
        at com.sun.xml.ws.api.pipe.Fiber._doRun(Fiber.java:554)
        at com.sun.xml.ws.api.pipe.Fiber.doRun(Fiber.java:539)
        at com.sun.xml.ws.api.pipe.Fiber.runSync(Fiber.java:436)
        at com.sun.xml.ws.api.pipe.helper.AbstractTubeImpl.process(AbstractTubeImpl.java:106)
        at com.sun.xml.xwss.XWSSClientPipe.process(XWSSClientPipe.java:160)
        at com.sun.xml.ws.api.pipe.helper.PipeAdapter.processRequest(PipeAdapter.java:115)
```

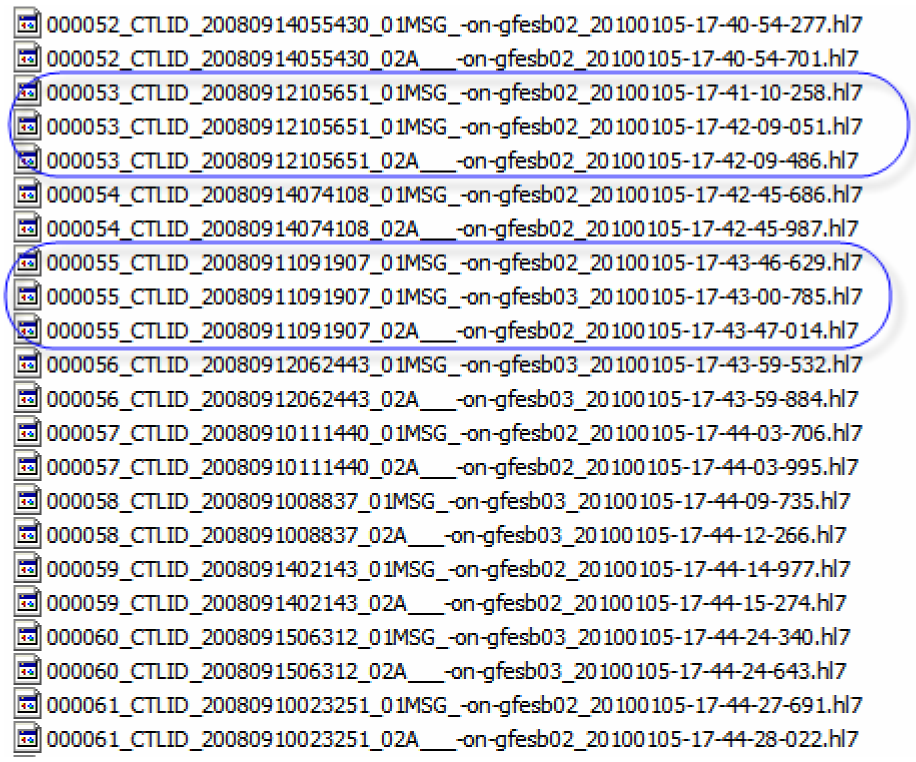
Once gfesb03 is ready and the load balancer connects to it, it re-joins the fold and starts processing its share of the workload.



```
===>>> sPropValue: gfesb02|#]

[#|2010-01-05T17:44:12.009+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=52;_ThreadName=sun-bpel-engine-thread-4;Process Instance Id=192.168.47.132:-478
f98e6:125ff7eae7a:-7ee0;Service Assembly Name=WSRcvr_CA;Activity Name=Wait1;BPEL Process Name=WSRcvr
;|===>>> Wait: P0DT0H0M1.998S, MsgCtlID: 000059_CTLID_2008091402143|#]

[#|2010-01-05T17:44:14.021+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.OutboundMess
ageProcessor|_ThreadID=45;_ThreadName=JMSBC-pool-1-thread-3;{http://j2ee.netbeans.org/wsdl/WSRcvr/WS
Rcvr_JMSOut}opADTA03;38898715251731-28259-134820062540200256;qWSADTIn;Context=WSRcvr_CA-sun-jms-bind
ing::WSRcvr_JMSOut_OutPort::opADTA03;|JMSBC-I0713: JMS binding operation {http://j2ee.netbeans.org/w
sdl/WSRcvr/WSRcvr_JMSOut}opADTA03 was marked as XATransaction but no transactional context is availa
ble in message exchange with ID 38898715251731-28259-134820062540200256; will continue to send JMS m
essage to JMS destination qWSADTIn without participating in an XA transaction|#]

```

```
essage to JMS destination qWSADTIn without participating in an XA transaction|#]

[#|2010-01-05T17:44:15.488+0000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=4
4;_ThreadName=sun-bpel-engine-thread-3;Process Instance Id=192.168.47.131:3cd66476:125ff92e4cc:-7f76
;Service Assembly Name=WSRcvr_CA;BPEL Process Name=WSRcvr;|
===>>> sPropValue: gfesb03|#]

[#|2010-01-05T17:44:15.601+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=44;_ThreadName=sun-bpel-engine-thread-3;Process Instance Id=192.168.47.131:3cd6
6476:125ff92e4cc:-7f76;Service Assembly Name=WSRcvr_CA;Activity Name=Wait1;BPEL Process Name=WSRcvr;
|===>>> Wait: P0DT0H0M8.233S, MsgCtlID: 000060_CTLID_2008091506312|#]
```

Here too we see some message duplication. The first highlighted set of messages, 000053, was both submitted by gfesb02. The first time message acknowledgment did not came soon enough and caused redelivery. Since gfesb03 was not available the retry used gfesb02, the only available host. In the second case, message 000055, gfesb03 was already available and was asked to process the message again. As before, messages were duplicated but none was lost.



```
000052_CTLID_20080914055430_01MSG_-on-gfesb02_20100105-17-40-54-277.hl7
000052_CTLID_20080914055430_02A___-on-gfesb02_20100105-17-40-54-701.hl7
000053_CTLID_20080912105651_01MSG_-on-gfesb02_20100105-17-41-10-258.hl7
000053_CTLID_20080912105651_01MSG_-on-gfesb02_20100105-17-42-09-051.hl7
000053_CTLID_20080912105651_02A___-on-gfesb02_20100105-17-42-09-486.hl7
000054_CTLID_20080914074108_01MSG_-on-gfesb02_20100105-17-42-45-686.hl7
000054_CTLID_20080914074108_02A___-on-gfesb02_20100105-17-42-45-987.hl7
000055_CTLID_20080911091907_01MSG_-on-gfesb02_20100105-17-43-46-629.hl7
000055_CTLID_20080911091907_01MSG_-on-gfesb03_20100105-17-43-00-785.hl7
000055_CTLID_20080911091907_02A___-on-gfesb02_20100105-17-43-47-014.hl7
000056_CTLID_20080912062443_01MSG_-on-gfesb03_20100105-17-43-59-532.hl7
000056_CTLID_20080912062443_02A___-on-gfesb03_20100105-17-43-59-884.hl7
000057_CTLID_20080910111440_01MSG_-on-gfesb02_20100105-17-44-03-706.hl7
000057_CTLID_20080910111440_02A___-on-gfesb02_20100105-17-44-03-995.hl7
000058_CTLID_2008091008837_01MSG_-on-gfesb03_20100105-17-44-09-735.hl7
000058_CTLID_2008091008837_02A___-on-gfesb03_20100105-17-44-12-266.hl7
000059_CTLID_2008091402143_01MSG_-on-gfesb02_20100105-17-44-14-977.hl7
000059_CTLID_2008091402143_02A___-on-gfesb02_20100105-17-44-15-274.hl7
000060_CTLID_2008091506312_01MSG_-on-gfesb03_20100105-17-44-24-340.hl7
000060_CTLID_2008091506312_02A___-on-gfesb03_20100105-17-44-24-643.hl7
000061_CTLID_20080910023251_01MSG_-on-gfesb02_20100105-17-44-27-691.hl7
000061_CTLID_20080910023251_02A___-on-gfesb02_20100105-17-44-28-022.hl7
```

Once things settled down, message processing proceeds to completion, with messages being processed alternating between gfesb02 and gfesb03.

Load balancing works. Fail-over works. The HTTP BC-based receiver solution is highly available. Additional hosts can be dynamically introduced to horizontally scale the solution for extra processing capacity.

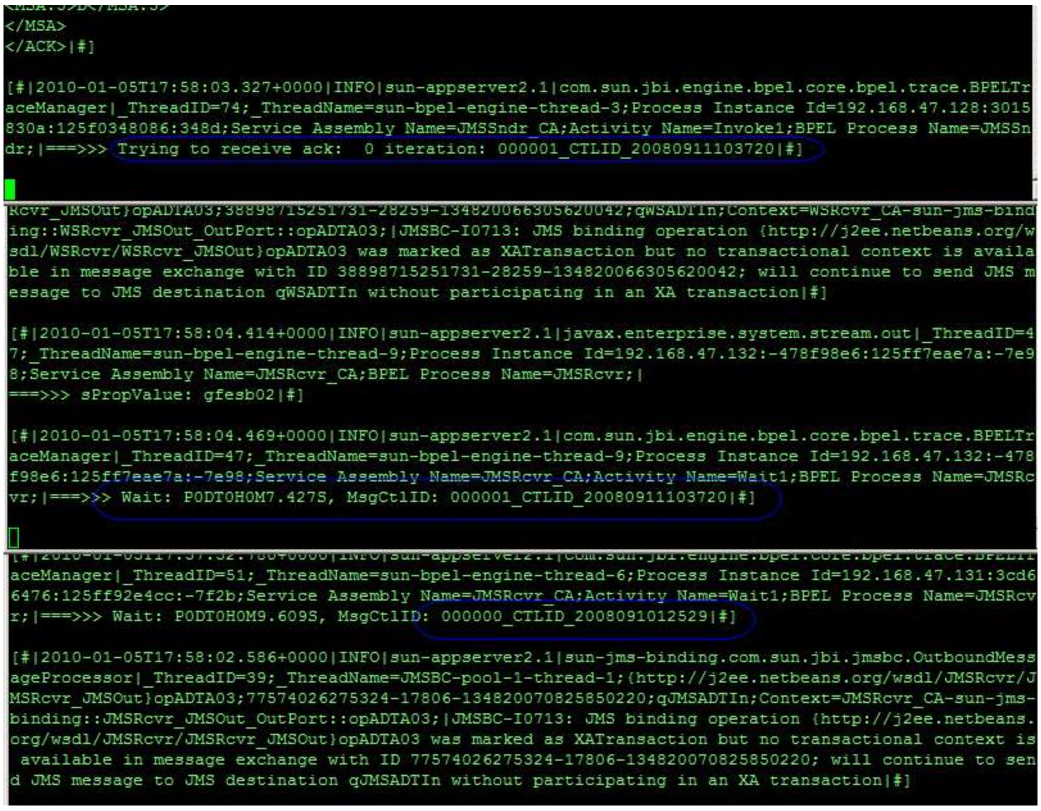
Testing resilience of the JMS BC-based Solution

We will start with all machines and all components started and processing messages. We expect messages to be processed in a round robin manner, alternating between gfesb02 and gfesb03. Once 15 or so messages are processed, we will “crash” gfesb03 by closing the VMware Player window in which it runs. This will cause retry functionality, after timeout period, to be invoked. We expect message flow to continue without message loss and we expect messages to now be processed by gfesb02 only. This should be reflected in names of the output files. Once we have seen enough messages, we will boot gfesb02 again and wait for it to start the GlassFish Application Server and deploy all the receivers. As soon as the JMSRcvr has been deployed we expect to see gfesb02 to start picking up messages again. We expect this to also be reflected in the names of the output files.

Recall that this solution implements the “Competing Consumers” EIP pattern, using JMS as the shared infrastructure. Since there is no load balancer we will not see server selection.

Let’s begin by copying the file ADT_A03_output_101.hl7, containing 101 ADT A03 messages, to the data_jms directory, and observing the console windows and the output directory until around 15 messages are processed.

Message processing commences and alternates between gfesb02 and gfesb03. Console windows provide feedback on messaging activity.



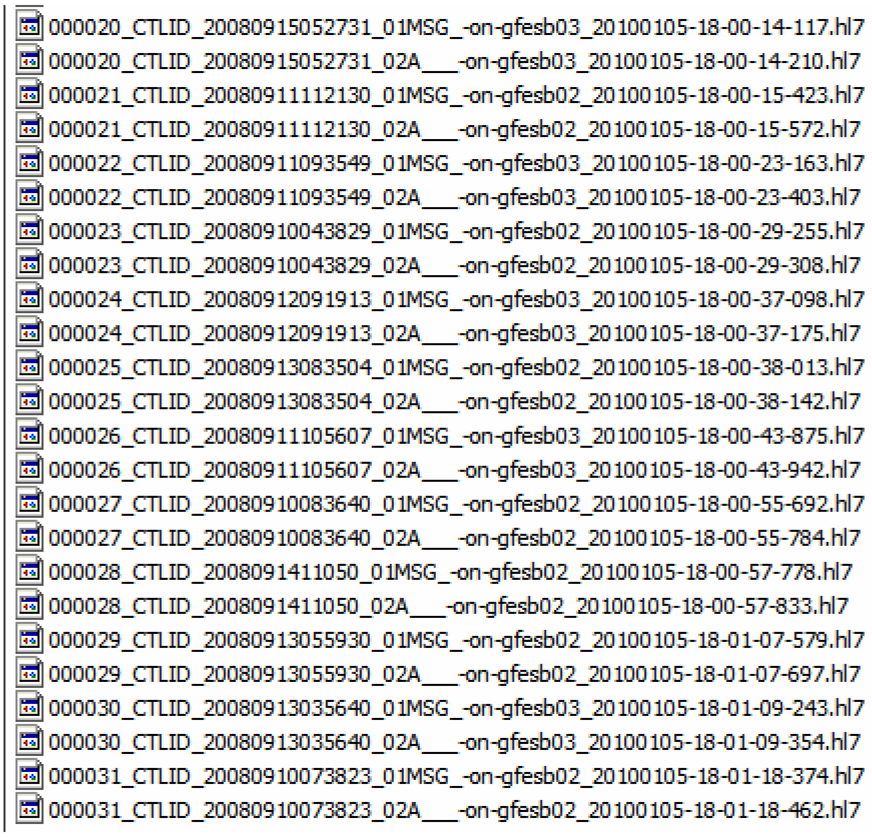
```
<MSA.3>D</MSA.3>
</MSA>
</ACK>|#]

[#|2010-01-05T17:58:03.327+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=74;_ThreadName=sun-bpel-engine-thread-3;Process Instance Id=192.168.47.128:3015
830a:125f0348086:348d;Service Assembly Name=JMSSndr_CA;Activity Name=Invoke1;BPEL Process Name=JMSSn
dr;|===>>> Trying to receive ack:  0 iteration: 000001_CTLID_20080911103720|#]
```

```
Rcvr_JMSOut}opADTA03;38898715251731-28259-13482006630562004 2;qWSADTIn;Context=WSRcvr_CA-sun-jms-bind
ing::WSRcvr_JMSOut_OutPort::opADTA03;|JMSBC-I0713: JMS binding operation {http://j2ee.netbeans.org/w
sdl/WSRcvr/WSRcvr_JMSOut}opADTA03 was marked as XATransaction but no transactional context is availa
ble in message exchange with ID 38898715251731-28259-13482006630562004 2; will continue to send JMS m
essage to JMS destination qWSADTIn without participating in an XA transaction|#]

[#|2010-01-05T17:58:04.414+0000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=4
7;_ThreadName=sun-bpel-engine-thread-9;Process Instance Id=192.168.47.132:-478f98e6:125ff7eae7a:-7e9
8;Service Assembly Name=JMSRcvr_CA;BPEL Process Name=JMSRcvr;|
===>>> sPropValue: gfesb02|#]

[#|2010-01-05T17:58:04.469+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=47;_ThreadName=sun-bpel-engine-thread-9;Process Instance Id=192.168.47.132:-478
f98e6:125ff7eae7a:-7e98;Service Assembly Name=JMSRcvr_CA;Activity Name=Wait1;BPEL Process Name=JMSRc
vr;|===>>> Wait: P0DT0H0M7.427S, MsgCtlID: 000001_CTLID_20080911103720|#]
```

```
[#|2010-01-05T17:57:52.780+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=51;_ThreadName=sun-bpel-engine-thread-6;Process Instance Id=192.168.47.131:3cd6
6476:125ff92e4cc:-7f2b;Service Assembly Name=JMSRcvr_CA;Activity Name=Wait1;BPEL Process Name=JMSRcv
r;|===>>> Wait: P0DT0H0M9.609S, MsgCtlID: 000000_CTLID_2008091012529|#]

[#|2010-01-05T17:58:02.586+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.OutboundMess
ageProcessor|_ThreadID=39;_ThreadName=JMSBC-pool-1-thread-1;{http://j2ee.netbeans.org/wsdl/JMSRcvr/J
MSRcvr_JMSOut}opADTA03;77574026275324-17806-13482007082585022 0;qJMSADTIn;Context=JMSRcvr_CA-sun-jms-
binding::JMSRcvr_JMSOut_OutPort::opADTA03;|JMSBC-I0713: JMS binding operation {http://j2ee.netbeans.
org/wsdl/JMSRcvr/JMSRcvr_JMSOut}opADTA03 was marked as XATransaction but no transactional context is
 available in message exchange with ID 77574026275324-17806-13482007082585022 0; will continue to sen
d JMS message to JMS destination qJMSADTIn without participating in an XA transaction|#]
```

JMSSndr logs the wait for acknowledgement attempt, which is synonymous with an attempt to deliver a message to the receiver. “0 iteration” means the first time delivery is attempted. This will be successful until receiver fails, at which point “1 iteration:” will be shown, meaning that the retry functionality in the JMSSndr was invoked.

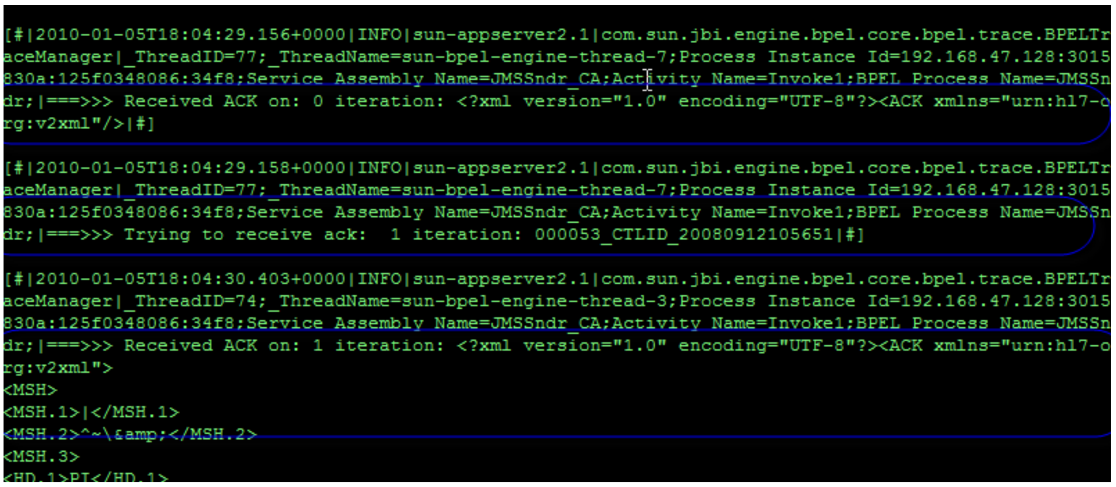
So far messages were processed mostly by alternating receivers. The round robin sequence is not strictly followed since there is no way to configure it. If the receiver processes a message fast enough it may get another message.



```
000020_CTLID_20080915052731_01MSG_-on-gfesb03_20100105-18-00-14-117.hl7
000020_CTLID_20080915052731_02A___-on-gfesb03_20100105-18-00-14-210.hl7
000021_CTLID_20080911112130_01MSG_-on-gfesb02_20100105-18-00-15-423.hl7
000021_CTLID_20080911112130_02A___-on-gfesb02_20100105-18-00-15-572.hl7
000022_CTLID_20080911093549_01MSG_-on-gfesb03_20100105-18-00-23-163.hl7
000022_CTLID_20080911093549_02A___-on-gfesb03_20100105-18-00-23-403.hl7
000023_CTLID_20080910043829_01MSG_-on-gfesb02_20100105-18-00-29-255.hl7
000023_CTLID_20080910043829_02A___-on-gfesb02_20100105-18-00-29-308.hl7
000024_CTLID_20080912091913_01MSG_-on-gfesb03_20100105-18-00-37-098.hl7
000024_CTLID_20080912091913_02A___-on-gfesb03_20100105-18-00-37-175.hl7
000025_CTLID_20080913083504_01MSG_-on-gfesb02_20100105-18-00-38-013.hl7
000025_CTLID_20080913083504_02A___-on-gfesb02_20100105-18-00-38-142.hl7
000026_CTLID_20080911105607_01MSG_-on-gfesb03_20100105-18-00-43-875.hl7
000026_CTLID_20080911105607_02A___-on-gfesb03_20100105-18-00-43-942.hl7
000027_CTLID_20080910083640_01MSG_-on-gfesb02_20100105-18-00-55-692.hl7
000027_CTLID_20080910083640_02A___-on-gfesb02_20100105-18-00-55-784.hl7
000028_CTLID_2008091411050_01MSG_-on-gfesb02_20100105-18-00-57-778.hl7
000028_CTLID_2008091411050_02A___-on-gfesb02_20100105-18-00-57-833.hl7
000029_CTLID_20080913055930_01MSG_-on-gfesb02_20100105-18-01-07-579.hl7
000029_CTLID_20080913055930_02A___-on-gfesb02_20100105-18-01-07-697.hl7
000030_CTLID_20080913035640_01MSG_-on-gfesb03_20100105-18-01-09-243.hl7
000030_CTLID_20080913035640_02A___-on-gfesb03_20100105-18-01-09-354.hl7
000031_CTLID_20080910073823_01MSG_-on-gfesb02_20100105-18-01-18-374.hl7
000031_CTLID_20080910073823_02A___-on-gfesb02_20100105-18-01-18-462.hl7
```

Let’s crash gfesb02, for a change, and see retry functionality in action.

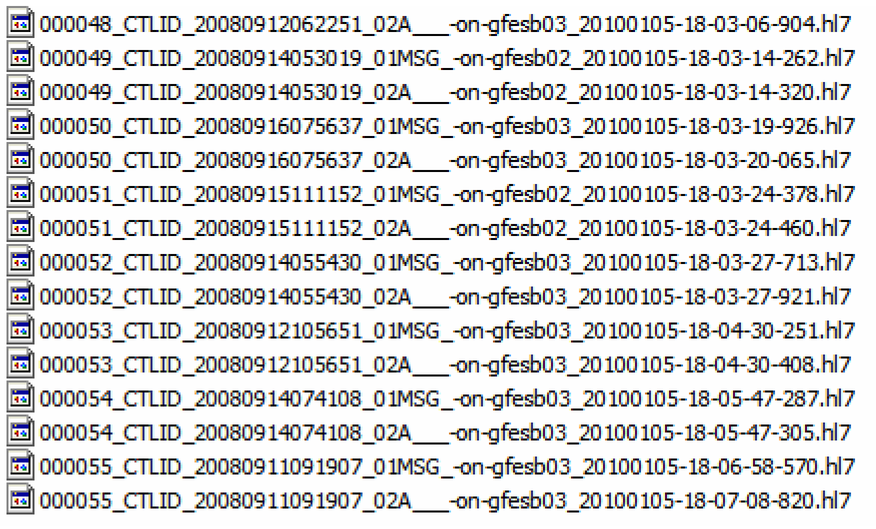
Console window on gfesb01 shows retry activity.



```
[#|2010-01-05T18:04:29.156+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=77;_ThreadName=sun-bpel-engine-thread-7;Process Instance Id=192.168.47.128:3015
830a:125f0348086:34f8;Service Assembly Name=JMSSndr_CA;Activity Name=Invoke1;BPEL Process Name=JMSSn
dr;|===>>> Received ACK on: 0 iteration: <?xml version="1.0" encoding="UTF-8"?><ACK xmlns="urn:hl7-o
rg:v2xml"/>|#]

[#|2010-01-05T18:04:29.158+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=77;_ThreadName=sun-bpel-engine-thread-7;Process Instance Id=192.168.47.128:3015
830a:125f0348086:34f8;Service Assembly Name=JMSSndr_CA;Activity Name=Invoke1;BPEL Process Name=JMSSn
dr;|===>>> Trying to receive ack:  1 iteration: 000053_CTLID_20080912105651|#]

[#|2010-01-05T18:04:30.403+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=74;_ThreadName=sun-bpel-engine-thread-3;Process Instance Id=192.168.47.128:3015
830a:125f0348086:34f8;Service Assembly Name=JMSSndr_CA;Activity Name=Invoke1;BPEL Process Name=JMSSn
dr;|===>>> Received ACK on: 1 iteration: <?xml version="1.0" encoding="UTF-8"?><ACK xmlns="urn:hl7-o
rg:v2xml">
<MSH>
<MSH.1>|</MSH.1>
<MSH.2>^~\&amp;</MSH.2>
<MSH.3>
<HD.1>PI</HD.1>
```

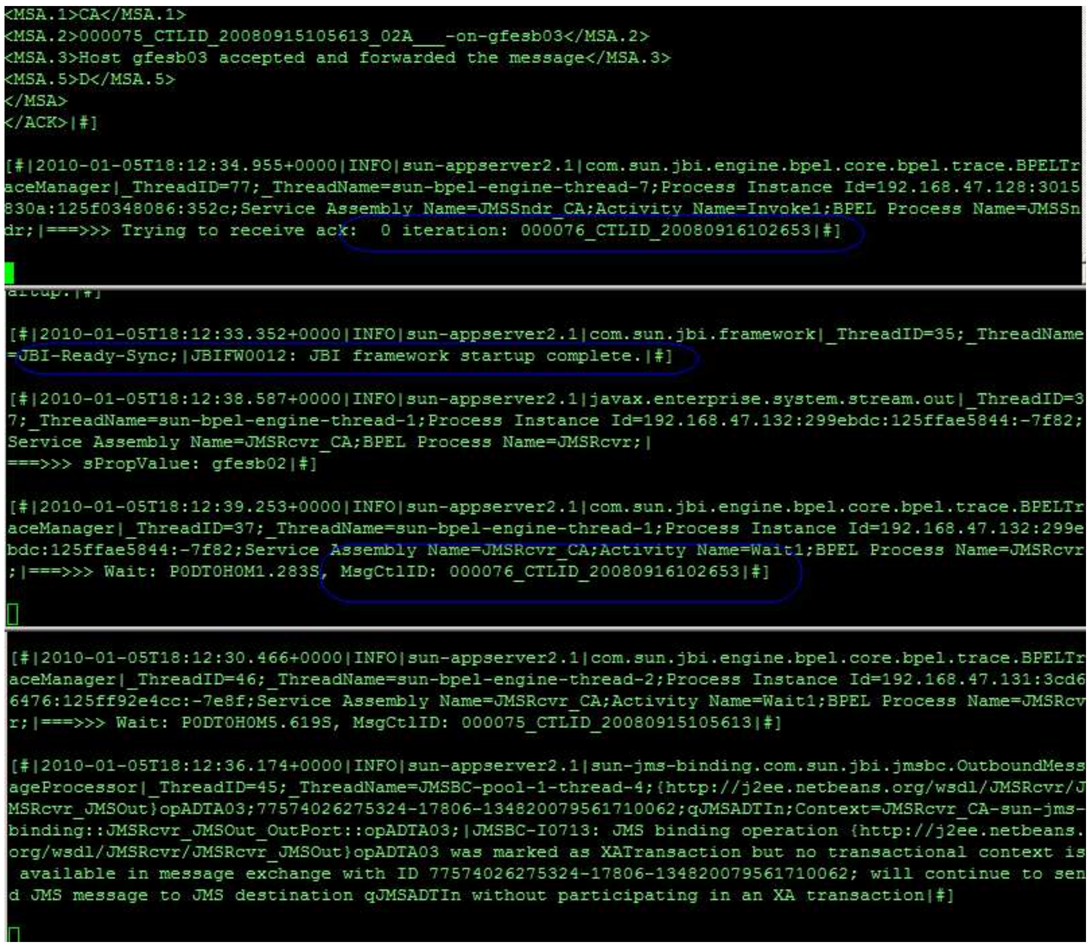
Note, in the first callout, that the ACK was returned by it was empty. The explicit retry logic cause another attempt to be made (second callout), which succeeded and returned a non-empty ACK (third callout).

Message processing then settled down to gfesb03 only.



```
000048_CTLID_20080912062251_02A___-on-gfesb03_20100105-18-03-06-904.hl7
000049_CTLID_20080914053019_01MSG_-on-gfesb02_20100105-18-03-14-262.hl7
000049_CTLID_20080914053019_02A___-on-gfesb02_20100105-18-03-14-320.hl7
000050_CTLID_20080916075637_01MSG_-on-gfesb03_20100105-18-03-19-926.hl7
000050_CTLID_20080916075637_02A___-on-gfesb03_20100105-18-03-20-065.hl7
000051_CTLID_20080915111152_01MSG_-on-gfesb02_20100105-18-03-24-378.hl7
000051_CTLID_20080915111152_02A___-on-gfesb02_20100105-18-03-24-460.hl7
000052_CTLID_20080914055430_01MSG_-on-gfesb03_20100105-18-03-27-713.hl7
000052_CTLID_20080914055430_02A___-on-gfesb03_20100105-18-03-27-921.hl7
000053_CTLID_20080912105651_01MSG_-on-gfesb03_20100105-18-04-30-251.hl7
000053_CTLID_20080912105651_02A___-on-gfesb03_20100105-18-04-30-408.hl7
000054_CTLID_20080914074108_01MSG_-on-gfesb03_20100105-18-05-47-287.hl7
000054_CTLID_20080914074108_02A___-on-gfesb03_20100105-18-05-47-305.hl7
000055_CTLID_20080911091907_01MSG_-on-gfesb03_20100105-18-06-58-570.hl7
000055_CTLID_20080911091907_02A___-on-gfesb03_20100105-18-07-08-820.hl7
```

Let’s start gfesb02. Once started, it rejoined the fold and started processing messages as before.



```
<MSA.1>CA</MSA.1>
<MSA.2>000075_CTLID_20080915105613_02A___-on-gfesb03</MSA.2>
<MSA.3>Host gfesb03 accepted and forwarded the message</MSA.3>
<MSA.5>D</MSA.5>
</MSA>
</ACK>|#]

[#|2010-01-05T18:12:34.955+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=77;_ThreadName=sun-bpel-engine-thread-7;Process Instance Id=192.168.47.128:3015
830a:125f0348086:352c;Service Assembly Name=JMSSndr_CA;Activity Name=Invoke1;BPEL Process Name=JMSSn
dr;|===>>> Trying to receive ack:  0 iteration: 000076_CTLID_20080916102653|#]
```

```
artup.|#]

[#|2010-01-05T18:12:33.352+0000|INFO|sun-appserver2.1|com.sun.jbi.framework|_ThreadID=35;_ThreadName
=JBI-Ready-Sync;|JBIFW0012: JBI framework startup complete.|#]

[#|2010-01-05T18:12:38.587+0000|INFO|sun-appserver2.1|javax.enterprise.system.stream.out|_ThreadID=3
7;_ThreadName=sun-bpel-engine-thread-1;Process Instance Id=192.168.47.132:299ebdc:125ffae5844:-7f82;
Service Assembly Name=JMSRcvr_CA;BPEL Process Name=JMSRcvr;|
===>>> sPropValue: gfesb02|#]

[#|2010-01-05T18:12:39.253+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=37;_ThreadName=sun-bpel-engine-thread-1;Process Instance Id=192.168.47.132:299e
bdc:125ffae5844:-7f82;Service Assembly Name=JMSRcvr_CA;Activity Name=Wait1;BPEL Process Name=JMSRcvr
;|===>>> Wait: P0DT0H0M1.283S, MsgCtlID: 000076_CTLID_20080916102653|#]
```

```
[#|2010-01-05T18:12:30.466+0000|INFO|sun-appserver2.1|com.sun.jbi.engine.bpel.core.bpel.trace.BPELTr
aceManager|_ThreadID=46;_ThreadName=sun-bpel-engine-thread-2;Process Instance Id=192.168.47.131:3cd6
6476:125ff92e4cc:-7e8f;Service Assembly Name=JMSRcvr_CA;Activity Name=Wait1;BPEL Process Name=JMSRcv
r;|===>>> Wait: P0DT0H0M5.619S, MsgCtlID: 000075_CTLID_20080915105613|#]

[#|2010-01-05T18:12:36.174+0000|INFO|sun-appserver2.1|sun-jms-binding.com.sun.jbi.jmsbc.OutboundMess
ageProcessor|_ThreadID=45;_ThreadName=JMSBC-pool-1-thread-4;{http://j2ee.netbeans.org/wsdl/JMSRcvr/J
MSRcvr_JMSOut}opADTA03;77574026275324-17806-134820079561710062;qJMSADTIn;Context=JMSRcvr_CA-sun-jms-
binding::JMSRcvr_JMSOut_OutPort::opADTA03;|JMSBC-I0713: JMS binding operation {http://j2ee.netbeans.
org/wsdl/JMSRcvr/JMSRcvr_JMSOut}opADTA03 was marked as XATransaction but no transactional context is
 available in message exchange with ID 77574026275324-17806-134820079561710062; will continue to sen
d JMS message to JMS destination qJMSADTIn without participating in an XA transaction|#]
```

Once things settled down processing continued in a round robin fashion until the run was completed.

Load balancing works. Fail-over works. The JMS BC-based receiver solution is highly available. Additional hosts can be dynamically introduced to horizontally scale the solution for extra processing capacity.

Summary

This note walked through the preparation of the GlassFish ESB v2.2 VMware Virtual Appliances for Load Balancing and High Availability exercise and deploying ready-made GlassFish ESB solutions. The exercise for HL7 BC-based, Web Service-based and JMS-based highly available, load balanced, and horizontally scalable receivers, processing HL7 v2.3.1 messages, was conducted and discussed.

We now have three GlassFish ESB VMware Appliances with GlassFish ESB v2.2 Runtime infrastructure, ready to use for further GlassFish ESB Load Balancing and High Availability exercise.

We are now convinced that for the applicable class of GlassFish ESB-based solutions load balancing and dynamic failover without message loss work. For these classes of applications this provides for high availability and horizontal scalability without resorting to Application Server or Operating System clustering.