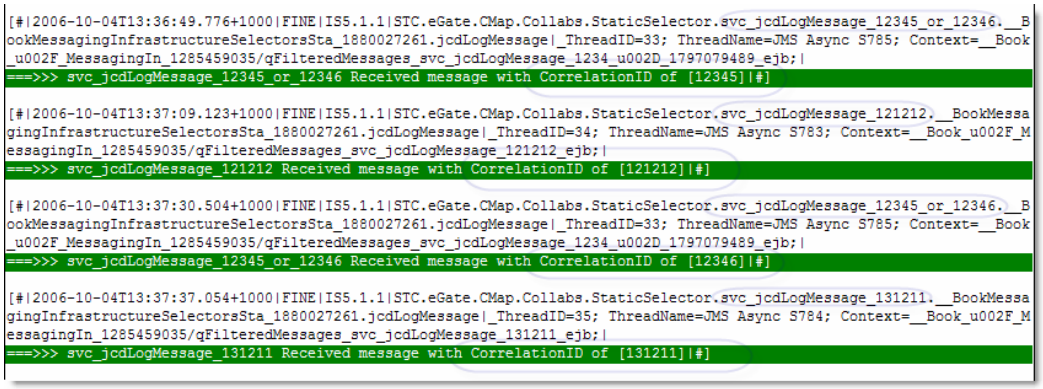
This is not a particularly realistic example as a downstream collaboration does not do anything of any value but the method holds. Let's submit the same series of message as before and observe the server.log indicating that all messages were received by a copy of the logging collaboration and indicating which of the collaboration service processed each message.



There are a copule of obvious drawbacks to implementing a Static Router this way, one of which is common to implementing a Static Router in general.

Firstly, in order to modify hardcoded static router, whether hardcoding is done within a component such as a Java Collaboration or a Business Process, or is done within a Connectivity Map, requires that the component is modified and the deployment profile-bound application is re-deployed. If the static routes are not-so-static this can become a major maintenance issue.

Secondly, and specifically applicable to the JMS Selector-based Static Router, there is no way to specify a "catch-all" selector except by inversing the conjunction of all other defined selectors. As soon as one selector needs to be added or modified the "catch-all" selector would need to be modified as well. Since multiple recipients from a single JMS Queue are Competing Consumers there is no way to pre-determine which will receive a particular message. Selector expressions must be carefully constructed to ensure that no overlaps occur otherwise unpredictable routing behaviour will result.

Building a dynamic selector requires a bit more work but once the method is established it can be applied as needed. Neither the JMS object, that is obtained by selecting the JMS OTD as one of the manipulation/output OTDs, nor the JMS input object, which is in fact a JMS Message object, provide access to the JMS Session object necessary to configure a selector, nor do they provide a way to find out what the selector expression is or to set one for the session. Since a JMS Session object is necessary we will use the Java

Naming and Directory Interface (JNDI) to lookup first the JMS ConnectionFactory object then the JMS Destination from which to receive messages using the dynamically cnstructed selector.

There is a fair bit of code involved in getting the infrastructure set up for a selective receiver with a dynamically constructed selector. The Java Collaboration, shown below, is broken into 5 parts to facilitate discussion. The entire collaboration is shown following the discussion.

The Collaboration uses classes from JMS and JNDI packages so it needs to import them immediately following the package line, as in any other Java class. This code will be the same regardless of the specific selector expressioin one will use later.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.Queue;
import javax.jms.QueueReceiver;
```

A number of static, final class fields, constants if you like, are used.

```
static final long lTimeout = 5 * 1000;
static final String cJMS_HOST = "localhost";
static final String cJMS_PORT = "20007";
static final String cRECEIVE_FROM = "qReceiveFrom";
static final String cPROVIDER_URL
       = "stcms://" + cJMS_HOST + ":" + cJMS_PORT;
static final String cCONNECTION_FACTORY
       = "connectionfactories/queueconnectionfactory";
static final String cINITIAL_CONTEXT_FACTORY
       = "com.stc.jms.jndispi.InitialContextFactory";
static final boolean cTRANSACTED = true;
```

Note that it would be more appropriate to obtain values for lTimeout, cJMS_HOST, cJMS_PORT and cRECEIVE_FROM fields at runtime rather than hardcoding them. This is not done in order to not obfuscate the essential JNDI and JMS code with matters that are tangential. Code required to read Java properties, and use their values, which is one way to get modifiable values into the runtime environment, is shown elsewhere in the book.

Obtain the value for the JMSCorrelationID to be used in constructing the selector expression. In this case the value is the entire content of the JMS Message. In a more realistic implementation it could be some part of the input message, whether acquired through JMS or using some other connector. It is likely that this code will change to accommodate specific application requirements. The application

may require using other Message Header fields and User Defined Properties in constructing the selector expression.

```
public void receive
    (com.stc.connectors.jms.Message input
    ,ud1.udtCorrelationIDInput1752469071.Udtcorrelationidinput vCorrIDIn
    ,com.stc.connectors.jms.JMS W_toJMS )
        throws Throwable
{
    // construct dynamic selector expression
    ;
    vCorrIDIn.unmarshalFromString( input.getTextMessage() );
    String sCorrelationID = vCorrIDIn.getCorrelationid();
    ;
    String sSelector = "JMSCorrelationID = '" + sCorrelationID + "'";
    ;
```

Create a timed JMS Receiver with the dynamically constructed selector expression. It is unlikely this code will change. This is pretty much how one would address the JNDI and JMS to obtain the selective receiver required for dynamic selector use.

```
// get a receiver for the specific Queue object
;
QueueConnection myConnection = null;
QueueSession mySession = null;
QueueReceiver myReceiver = null;
;
try {
    java.util.Hashtable env = new java.util.Hashtable();
    env.put( Context.INITIAL_CONTEXT_FACTORY, cINITIAL_CONTEXT_FACTORY );
    env.put( Context.PROVIDER_URL, cPROVIDER_URL );
    InitialContext jndiContext = new InitialContext( env );
    QueueConnectionFactory QCFactory =
        (QueueConnectionFactory) jndiContext.lookup( cCONNECTION_FACTORY );
    myConnection = QCFactory.createQueueConnection();
    Queue myQueue = (Queue) jndiContext.lookup( "queues/" + cRECEIVE_FROM );
    mySession = myConnection.createQueueSession
            ( cTRANSACTED, Session.AUTO_ACKNOWLEDGE );
    myReceiver = mySession.createReceiver( myQueue, sSelector );
    myConnection.start();
} catch ( Exception e ) {
    e.printStackTrace();
    throw new Exception( "\n===>>> Exception from jndi processing", e );
}
;
```

Once the selective receiver is available we can attempt to read one or more messages from the JMS Queue. This part of the collaboration will likely change as required by the implementation. In this sample we read as many messages, that satisfy the selector expression, as there are and append their bodies to a string. This is not particularly realistic but is simple enough for illustration of the concepts. In the example we send the string, assembled by reading selected messages, to a 'regular' JMS Destination configured through the Connectivity Map.

```
// prepare canned text to preprend to messages
// sent out
```

```
;
String sOutText = "";
sOutText += "Using Selector Expression [" + sSelector + "]";
sOutText += "\nReceived input message with Correlation ID ";
sOutText += vCorrIDIn.getCorrelationid();
;
// is there at least one message in the correlation queue?
;
javax.jms.TextMessage m = null;
m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
if (m == null) {
    sOutText = "";
    sOutText += "\nNo candidate messages for the selector [";
    sOutText += myReceiver.getMessageSelector();
    sOutText += "] from queue [";
    sOutText += myReceiver.getQueue().getQueueName() + "]";
    mySession.rollback();
}
;
// process all related message in the correlation queue
;
while (m != null) {
    sOutText += "\n Message Body: [";
    sOutText += m.getText();
    sOutText += "] for Message: ";
    sOutText += m.getJMSMessageID();
    m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
}
;
com.stc.connectors.jms.Message msgOut = W_toJMS.createTextMessage();
msgOut.getMessageProperties().setCorrelationID( sCorrelationID );
msgOut.setTextMessage( sOutText );
W_toJMS.send( msgOut );
```

Note that in addition to setting the body of the outgoing message we are also setting the CorrelationID JMS Header Property just in case the downstream component has a need or use for it.

Finally, we dismantle the selective receiver infrastructure and finish.

```
mySession.commit();
mySession.close();
myConnection.stop();
myConnection.close();
;
logger.debug( "\n===>>> sent message: " + sOutText );
```

Here is the collaboration code in its entirety. Further discussion of issues and consideration relating to this particular implementation follows the exhibit.

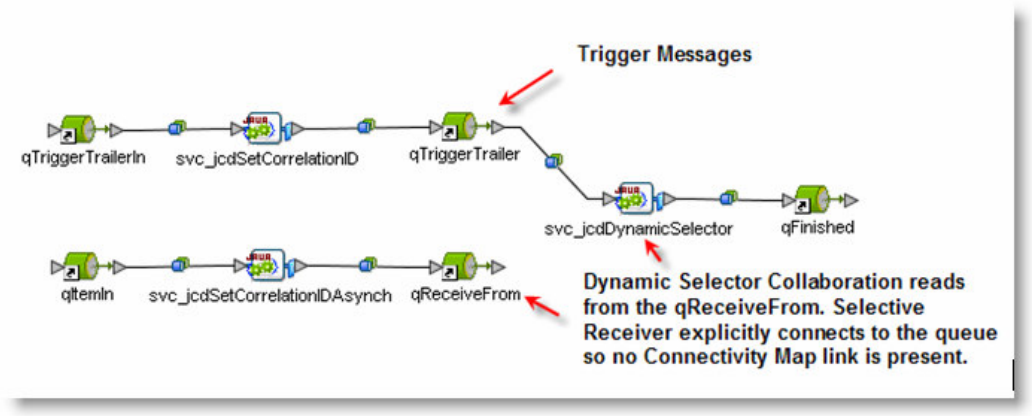
```
package __BookMessagingInfrastructureSelectorsDyn_1147489048;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.Queue;
import javax.jms.QueueReceiver;


public class jcdDynamicSelector
```

```
{

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    static final long lTimeout = 5 * 1000;
    static final String cJMS_HOST = "localhost";
    static final String cJMS_PORT = "20007";
    static final String cRECEIVE_FROM = "qReceiveFrom";
    static final String cPROVIDER_URL = "stcms://" + cJMS_HOST + ":" + cJMS_PORT;
    static final String cCONNECTION_FACTORY =
            "connectionfactories/queueconnectionfactory";
    static final String cINITIAL_CONTEXT_FACTORY =
            "com.stc.jms.jndispi.InitialContextFactory";
    static final boolean cTRANSACTED = true;

    public void receive
        (com.stc.connectors.jms.Message input
        ,ud1.udtCorrelationIDInput1752469071.Udtcorrelationidinput vCorrIDIn
        ,com.stc.connectors.jms.JMS W_toJMS )
            throws Throwable
    {
        // construct dynamic selector expression
        ;
        vCorrIDIn.unmarshalFromString( input.getTextMessage() );
        String sCorrelationID = vCorrIDIn.getCorrelationid();
        ;
        String sSelector = "JMSCorrelationID = '" + sCorrelationID + "'";
        ;
        // get a receiver for the specific Queue object
        ;
        QueueConnection myConnection = null;
        QueueSession mySession = null;
        QueueReceiver myReceiver = null;
        ;
        try {
            java.util.Hashtable env = new java.util.Hashtable();
            env.put( Context.INITIAL_CONTEXT_FACTORY, cINITIAL_CONTEXT_FACTORY );
            env.put( Context.PROVIDER_URL, cPROVIDER_URL );
            InitialContext jndiContext = new InitialContext( env );
            QueueConnectionFactory QCFactory =
                (QueueConnectionFactory) jndiContext.lookup( cCONNECTION_FACTORY );
            myConnection = QCFactory.createQueueConnection();
            Queue myQueue = (Queue) jndiContext.lookup( "queues/" + cRECEIVE_FROM );
            mySession = myConnection.createQueueSession
                ( cTRANSACTED, Session.AUTO_ACKNOWLEDGE );
            myReceiver = mySession.createReceiver( myQueue, sSelector );
            myConnection.start();
        } catch ( Exception e ) {
            e.printStackTrace();
            throw new Exception( "\n===>>> Exception from jndi processing", e );
        }
        ;
        // prepare canned text to preprend to messages
        // sent out
        ;
        String sOutText = "";
        sOutText += "Using Selector Expression [" + sSelector + "]";
        sOutText += "\nReceived input message with Correlation ID ";
        sOutText += vCorrIDIn.getCorrelationid();
        ;
        // is there at least one message in the correlation queue?
        ;
        javax.jms.TextMessage m = null;
        m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
        if (m == null) {
            sOutText = "";
            sOutText += "\nNo candidate messages for the selector [";
            sOutText += myReceiver.getMessageSelector();
            sOutText += "] from queue [";
            sOutText += myReceiver.getQueue().getQueueName() + "]";
            mySession.rollback();
        }
        ;
        // process all related message in the correlation queue
```

```
        ;
        while (m != null) {
            sOutText += "\n Message Body: [";
            sOutText += m.getText();
            sOutText += "] for Message: ";
            sOutText += m.getJMSMessageID();
            m = (javax.jms.TextMessage) myReceiver.receive( lTimeout );
        }
        ;
        com.stc.connectors.jms.Message msgOut = W_toJMS.createTextMessage();
        msgOut.getMessageProperties().setCorrelationID( sCorrelationID );
        msgOut.setTextMessage( sOutText );
        W_toJMS.send( msgOut );
        ;
        mySession.commit();
        mySession.close();
        myConnection.stop();
        myConnection.close();
        ;
        logger.debug( "\n===>>> sent message: " + sOutText );
    }
}
```

__Book/MessagingInfrastructure/Selectors/DynamicSelector/jcdDynamicSelector

The collaboration shown above is triggered by a JMS Message that conveys the Correlation ID to be used by the selective receiver. The JMS Queue used could be the same Queue as the one in which all other messages exist or it could be different. In the former case one would expect a static selector expression to pick just the message that starts the ball rolling and to ignore messages that the collaboration is to explicitly receive. In the latter case one would have to ensure that the JMS Queue that triggers the selective receiver collaboration only receives trigger messages and no others. In the sample two distinct JMS Queues are used.



__Book/MessagingInfrastructure/Selectors/DynamicSelector/cm_DynamicSelector

Let's buld and deploy the project and exercise the implementation to convince ourselves that the dynamic selector infrastructure works. To qItemIn submit three messages, one with the value of 12345, one with the value of 121212 and another one with the value of 12345. To qTriggerTraileIn submit two messages, one with

the value of 12345 and one with the value of 121212, in any order. In qFinished observe two messages, one relating to messages with the Correlation ID of 12345 and one with the Correlation ID of 121212.

The server.log shows the correlated messages.

```
[#|2006-10-
05T22:04:18.617+1000|FINE|IS5.1.1|STC.eGate.CMap.Collabs.DynamicSelector.
svc_jcdDynamicSelector.__BookMessagingInfrastructureSelectorsDyn_114748904
8.jcdDynamicSelector|_ThreadID=32; ThreadName=JMS Async S293;
Context=__Book_u002F_MessagingIn_152549869/qTriggerTrailer_svc_jcdDynam
icSelector_ejb;|
===>>> sent message: Using Selector Expression [JMSCorrelationID = '12345']
Received input message with Correlation ID 12345
 Message Body: [12345] for Message:
ID:a2124:10e17e05f07:840:c0a83c02:10e185ba808:7fbcb25110e17e01b777c04
 Message Body: [12345] for Message:
ID:f8d5a:10e17e05f09:840:c0a83c02:10e185be2c0:7fbcb25110e17e01b777c00|
#]

[#|2006-10-
05T22:04:25.968+1000|FINE|IS5.1.1|STC.eGate.CMap.Collabs.DynamicSelector.
svc_jcdDynamicSelector.__BookMessagingInfrastructureSelectorsDyn_114748904
8.jcdDynamicSelector|_ThreadID=32; ThreadName=JMS Async S293;
Context=__Book_u002F_MessagingIn_152549869/qTriggerTrailer_svc_jcdDynam
icSelector_ejb;|
===>>> sent message: Using Selector Expression [JMSCorrelationID =
'121212']
Received input message with Correlation ID 121212
 Message Body: [121212] for Message:
ID:eac9e:10e17e05f08:840:c0a83c02:10e185bc23e:7fbcb25110e17e01b777c02|
#]
```

Whilst this is a practical way of implementing a selective receiver with the selector expression created dynamically at runtime, it is a fairly inefficient one as the entire process of looking up and creating appropriate JMS objects and destroying them after use is performed each time through the collaboration. A knowledgeable reader can likely improve efficiency by using statics to hold various objects and not destroying objects on exit from the collaboration.

The Connectivity Map may or may not imply that a selective receiver is used. This depends on the site naming conventions and designer's willingness to provide hints to others that there exist implicit interrelationships that the Connectivity Map does not show.

Since the JMS Message Server URL is constructed using hardcoded values the collaboration is not portable between Message Servers. A better way would be to configure the URL using values acquired at runtime so that changes can be externalised and made without

affecting the collaboration or requiring projects that use it to be re-built and re-deployed.

Not that this method can be used to implement eGate-only-based correlations. Indeed, this mthod is used in the section 10.10, eGate Correlation with Dynamic Selectors to implement a number of Message Relationship Patterns.

11.5.8 FIFO Modes (???)

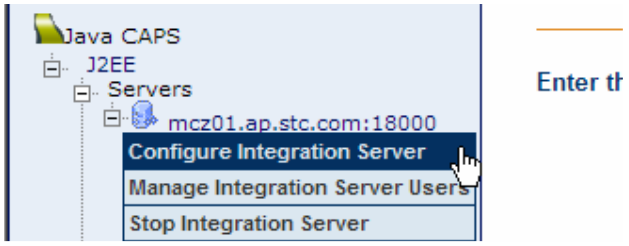
@@@@@@@@@@Research time order groups @@@@@@@@@@
[TBD]

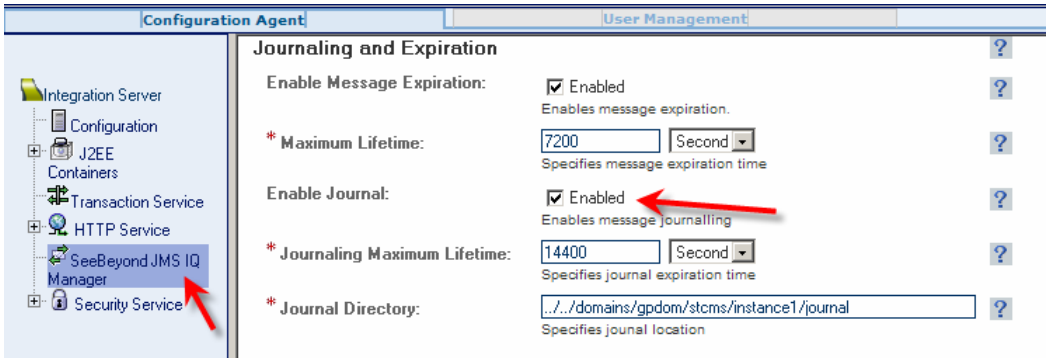
11.5.9 Throttling (???)

[TBD]

11.5.10 Message Journaling

The Sun SeeBeyond JMS IQ Manager supports journaling of delivered messages. Journaling is enabled globally through the IS Administrator interface or through the Enterprise Manager "Configure Integration Server" interface.





Since Journaling is enabled globally, and journalled messages, even after expiration, are not removed from the journal storage, procedures must be put in place to remove messages from the Journal storage to prevent it from growing indefinitely.