Grouper Messaging System

The recommended approach for messaging in Grouper v2.6+ is:

1. Use a provisioner instead of messaging so that full and incremental syncs can occur and all the provisioning framework features can be used
2. If you still want to use messaging, consider using a messaging provisioner (such as Amazon AWS SNS/SQS, ActiveMQ, RabbitMQ) instead of the messaging change log. There is more granular control of which objects are eligible for the messaging, and information is kept/displayed about when messages are sent
3. If you still want to use messaging change log consumer, it is still supported

Grouper messaging system is a Java implementation of the Interface GrouperMessagingSystem. It allows messages to be sent and received from a messaging system.

The built in implementation is

- Grouper database (default):

Newer options, (found in Grouper 2.5+) leveraging external messaging systems are:

- Amazon AWS SNS/SQS
- ActiveMQ
- RabbitMQ

The messages:

- must support 100kB in size
- must support ordered messaging (unless ordered messaging consumers are not used)
- there are bulk methods, but the implementation can do them one at a time (if there is an error, block until all are successful)

- Grouper messaging built in
- Grouper messaging send receive change log adapter example
- Grouper messaging send receive example
- Grouper Messaging System development guide
- Grouper Messaging System - full sync messages
- Grouper Messaging with ActiveMQ
- Grouper Messaging with AWS SQS
- Grouper Messaging with RabbitMQ

Configure an ESB change log consumer in grouper-loader.properties

```
#####################################
## Messaging integration with ESB, send change log entries to a messaging system
#####################################

# note, change "messagingEsb" in key to be the name of the consumer.  e.g. changeLog.consumer.myAzureConsumer.
class
# note, routingKey property is valid only for rabbitmq. For other messaging systems, it is ignored.
#changeLog.consumer.messagingEsb.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.EsbConsumer

# quartz cron
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.quartzCron$"}
#changeLog.consumer.messagingEsb.quartzCron = 0 * * * * ?

# el filter
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.elfilter$"}
#changeLog.consumer.messagingEsb.elfilter = event.eventType eq 'GROUP_DELETE' || event.eventType eq 'GROUP_ADD'
|| event.eventType eq 'MEMBERSHIP_DELETE' || event.eventType eq 'MEMBERSHIP_ADD'

# publishing class
# {valueType: "class", mustExtendClass: "edu.internet2.middleware.grouper.changeLog.esb.consumer.
EsbMessagingPublisher", regex: "^changeLog\\.consumer\\.([^.]+)\\.publisher\\.class$"}
#changeLog.consumer.messagingEsb.publisher.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.
EsbMessagingPublisher

# messaging system name
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.messagingSystemName$"}
#changeLog.consumer.messagingEsb.publisher.messagingSystemName = grouperBuiltinMessaging

# routing key
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.routingKey$"}
#changeLog.consumer.messagingEsb.publisher.routingKey =

# EL replacement definition. groupName is the variable for the name of the group. grouperUtil is the class
GrouperUtilElSafe can be used for utility methods.
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.regexRoutingKeyReplacementDefinition$"}
#changeLog.consumer.messagingEsb.regexRoutingKeyReplacementDefinition = ${groupName.replaceFirst('hawaii.edu',
'group.modify').replace(':enrolled', '').replace(':waitlisted', '').replace(':withdrawn', '')}

# replace routing key with periods
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.replaceRoutingKeyColonsWithPeriods$"}
#changeLog.consumer.messagingEsb.replaceRoutingKeyColonsWithPeriods = true

# queue or topic
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.publisher\\.messageQueueType$"}
#changeLog.consumer.messagingEsb.publisher.messageQueueType = queue

# queue or topic name
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.publisher\\.queueOrTopicName$"}
#changeLog.consumer.messagingEsb.publisher.queueOrTopicName = abc

# exchange type for rabbitmq. valid options are DIRECT, TOPIC, HEADERS, FANOUT
# {valueType: "string", regex: "^changeLog\\.consumer\\.([^.]+)\\.publisher\\.exchangeType$"}
#changeLog.consumer.messagingEsb.publisher.exchangeType =
```

The GrouperMessagingSystem interface is located in the GrouperClient:

```
/**
 *
 * @author mchyzer
 * $Id$
 */
package edu.internet2.middleware.grouperClient.messaging;

/**
 * Represents the methods that a messaging system
 * needs to support
 */
public interface GrouperMessagingSystem {


  /**
   * send a message to a queue name.  Note, the recipient could be a
   * queue or a topic (generally always one or the other) based on the
   * implementation of the messaging system.  Messages must be delievered
   * in the order that collection iterator designates.  If there is a problem
   * delivering the messages, the implementation should log, wait (back off)
   * and retry until it is successful.
   * @param grouperMessageSendParam has the queue or topic, and the message(s) and perhaps args
   * @return result
   */
  public GrouperMessageSendResult send(GrouperMessageSendParam grouperMessageSendParam);

  /**
   * this will generally block until there are messages to process.  These messages
   * are ordered in the order that they were sent.
   * @param grouperMessageReceiveParam grouper messaging receive param
   * @return a message or multiple messages.  It will block until there are messages
   * available for this recipient to process
   */
  public GrouperMessageReceiveResult receive(GrouperMessageReceiveParam grouperMessageReceiveParam);


  /**
   * tell the messaging system that these messages are processed
   * generally the message system will use the message id.  Note, the objects
   * sent to this method must be the same that were received in the
   * receiveMessages method.  If there is a problem
   * delivering the messages, the implementation should wait (back off)
   * and retry until it is successful.  Alternatively the message should be
   * returned to queue, returned to end of queue, or sent to another queue
   * @param grouperMessageAcknowledgeParam
   * @return result
   */
  public GrouperMessageAcknowledgeResult acknowledge(GrouperMessageAcknowledgeParam
grouperMessageAcknowledgeParam);

}
```

This is the interface for GrouperMessage (located in the GrouperClient), which has a default implementation that can be used. Note the message contents will be encrypted, have metadata, etc.

```
/**
 * @author mchyzer
 * $Id$
 */
package edu.internet2.middleware.grouperClient.messaging;

/**
 * grouper message sent to/from grouper messaging systems
 */
public interface GrouperMessage {


  /**
   * member id of a subjcet that sent the message
   * @return the from member id
   */
  public String getFromMemberId();


  /**
   * @param fromMemberId1 the from to set
   */
  public void setFromMemberId(String fromMemberId1);


  /**
   * @return the id
   */
  public String getId();


  /**
   * @param id1 the id to set
   */
  public void setId(String id1);


  /**
   * @return the message
   */
  public String getMessageBody();

  /**
   * @param message1 the message to set
   */
  public void setMessageBody(String message1);
}
```

See Also:

GSH to manage built in messaging

GSH to send / receive messages

Grouper Built In Messaging

Message Format Detail

Message Format Config Example

Grouper Messaging System Development Guide

Grouper Messaging to Web Service API

Change Log Consumers