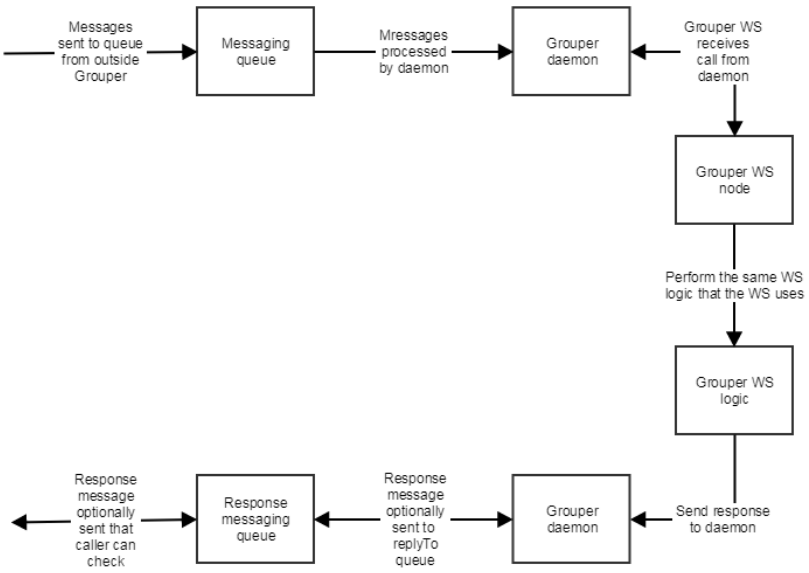
Grouper messaging to web service API

Messages from the Grouper messaging system from a queue can be read by Grouper and treated like a web service call.



Configuration

Enable the daemon in grouper-loader.properties

```
# message to ws daemon job class
# {valueType: "class", mustExtendClass: "edu.internet2.middleware.grouper.app.loader.OtherJobBase",
mustImplementInterface: "org.quartz.Job"}
otherJob.messageConsumerDaemon.class = edu.internet2.middleware.grouper.app.messaging.MessageConsumerDaemon

# message to ws daemon job cron
# {valueType: "cron", required: true}
otherJob.messageConsumerDaemon.quartzCron = 0 * * ? * *
```

Hook up grouper WS to read from a queue:

grouper-loader.properties

```
# there can be multiple entries, "wsMessagingBridge" is the name of this one, change that for each config
section

# the messaging system name must correspond to a messaging system in the grouper.client.properties
grouper.webservice.messaging.wsMessagingBridge.messagingSystemName = grouperBuiltinMessaging

# the queue or topic to check
grouper.webservice.messaging.wsMessagingBridge.queueOrTopicName = sampleWsMessagingQueue
# routingKey is only valid for rabbitmq; for others, it's ignored
grouper.messaging.wsMessagingBridge.routingKey =

# if this is a "queue" or "topic", generally it will be queue
grouper.webservice.messaging.wsMessagingBridge.messageQueueType = queue

# the source id of the source of the user to act as
grouper.webservice.messaging.wsMessagingBridge.actAsSubjectSourceId = g:isa

# the subject id of the user to act as
grouper.webservice.messaging.wsMessagingBridge.actAsSubjectId = GrouperSystem

# the number of seconds between checking the queue or topic
grouper.webservice.messaging.wsMessagingBridge.secondsBetweenChecks = 0

# the long polling seconds, listen to the queue for this many seconds for messages
grouper.webservice.messaging.wsMessagingBridge.longPollingSeconds = 20
```

That will cause a daemon to process messages and reverse proxy the WS

Note, you need to configure the grouper act as too

```
# similar syntax as ws.act.as.group but for the grouper actas (e.g. for grouper messaging to WS bridge)
# Web service users who are in the following group can use the actAs field to act as someone else
# You can put multiple groups separated by commas.  e.g. a:b:c, e:f:g
# You can put a single entry as the group the calling user has to be in, and the grouper the actAs has to be in
# separated by 4 colons
# e.g. if the configured values is:       a:b:c, e:f:d :::: r:e:w, x:e:w
# then if the calling user is in a:b:c or x:e:w, then the actAs can be anyone
# if not, then if the calling user is in e:f:d, then the actAs must be in r:e:w.  If multiple rules, then
# if one passes, then it is a success, if they all fail, then fail.
ws.grouper.act.as.group =
```

Sample message input

```
{
  "grouperHeader": {
    "messageVersion": "1",  //mandatory
    "timestampInput": "2017-07-23T18:25:43.511Z",          //timestamp message sent, mandatory
    "type": "grouperMessagingToWebService",                //says what type of message, mandatory
    "endpoint": "WsRestAddMemberRequest",                  //which endpoint, mandatory, matches the container
name below
    "messageInputUuid": "abc123",                          //for logging, mandatory, make up a uuid
    "replyToQueueOrTopicName": "someQueue",                //if replying, optional
    "replyToQueueOrTopic": "queue",                        //if replying, "queue" or "topic", optional
    "replyToRoutingKey": "x:y",                                                    //if replying,
optional, valid only if replying to rabbitmq
    "replyToExchangeType": "TOPIC",                        //if replying, optional, valid only if replying to
rabbitmq using exchange
    "httpMethod": "PUT",                                   //http method that would be in WS, mandatory
    "httpPath": "/servicesRest/v2_2_000/groups" //http path that would be in the WS, mandatory
  },
  // this is simply the json body of any grouper json web service normal or lite
  "WsRestAddMemberRequest":{
    "subjectLookups":[
      {
        "subjectId":"test.subject.0",
        "subjectSourceId":"jdbc"
      }
    ]
    ,
    "wsGroupLookup":{
      "groupName":"test:testGroup"
    }
  }
}
```

Sample message output

If there is a "reply to", then this is the message that will be sent

```
{
  "grouperHeader": {
    "messageVersion": "1",                          //mandatory
    "timestampInput": "2017-07-23T18:23:43.511Z",  //timestamp message input sent, mandatory
    "timestampOutput": "2012-07-23T18:25:45.511Z", //timestamp message putput sent, mandatory
    "type": "grouperMessagingFromWebService",      //says what type of message, mandatory
    "endpoint": "WsRestAddMemberRequest",          //which endpoint, mandatory, matches input container
    "messageInputUuid": "abc123",                  //for logging, mandatory, matches the input uuid
    "httpResponseCode": 201,                       //http response code that would have been sent over WS,
mandatory
    "httpHeader_X-Grouper-resultCode": "SUCCESS",  //http header that would have been sent over WS, mandatory
    "httpHeader_X-Grouper-success": "T",           //http header that would have been sent over WS, mandatory
    "httpHeader_X-Grouper-resultCode2": "NONE"     //http header that would have been sent over WS, mandatory
  },
  // this is simply the json body of any grouper json web service normal or lite
  "WsAddMemberResults":{
    "responseMetadata":{
      "millis":"120",
      "serverVersion":"2.2.0"
    },
    "resultMetadata":{
      "resultCode":"SUCCESS",
      "resultMessage":"Success for: clientVersion: 2.2.0, wsGroupLookup: WsGroupLookup[pitGroups=[],
groupName=test:testGroup], subjectLookups: Array size: 1: [0]: WsSubjectLookup[subjectId=test.subject.0,
subjectSourceId=jdbc]\n\n, replaceAllExisting: false, actAsSubject: null, fieldName: null, txType: NONE,
includeGroupDetail: false, includeSubjectDetail: false, subjectAttributeNames: null\n, params: null\n,
disabledDate: null, enabledDate: null",
      "success":"T"
    },
    "results":[
      {
        "resultMetadata":{
          "resultCode":"SUCCESS",
          "success":"T"
        },
        "wsSubject":{
          "id":"test.subject.0",
          "name":"my name is test.subject.0",
          "resultCode":"SUCCESS",
          "sourceId":"jdbc",
          "success":"T"
        }
      }
    ]
    ,
    "wsGroupAssigned":{
      "displayExtension":"testGroup",
      "displayName":"test:testGroup",
      "extension":"testGroup",
      "idIndex":"10000",
      "name":"test:testGroup",
      "typeOfGroup":"group",
      "uuid":"81cb5aa761fb477ca152738691a93e8b"
    }
  }
}
```

Grouper client support

The grouper client could be run with a flag that runs as a message instead of WS. This could help with testing, debug, etc.