Grouper provisioning incremental workflow

Grouper provisioners in Grouper 2.5 and above will be an "esb listener" and process EsbEvent objects. However, unlike Grouper 2.4 listeners, the provisioner change log consumer will process multiple events at once, and not from a JSON representation. This occurs in the provisioning framework.

Configure an incremental job

grouper-loader.properties

```
changeLog.consumer.<jobName>.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.EsbConsumer
changeLog.consumer.<jobName>.publisher.class= edu.internet2.middleware.grouper.app.provisioning.
ProvisioningConsumer
changeLog.consumer.<jobName>.quartzCron = 0 * * * * ?
changeLog.consumer.<jobName>.provisionerTarget = <provisionerConfigId>
changeLog.consumer.<jobName>.provisionerJobSyncType = incrementalProvisionChangeLog
changeLog.consumer.<jobName>.publisher.debug = false
```

Recalculate operations (stateful vs stateless)

When events are processed by the provisioning framework, they are one of two types:

1. Without recalculate (stateful)
    a. Take an "add member 123 to group a:b:c" and send that operation to the target
    b. This will check the state in grouper (is the user a member), and the cached target state (does grouper think the user is a member in the target group?)
        i. If the Grouper state doesnt match the event, e.g. an add member and the user is not a member of the group), this will be converted into a recalc event
        ii. If the cached state of the target doesnt match the event, e.g. if grouper thinks the user is already a member of the group in the target, and its a "add member" event, convert the event to a recalc event
    c. This is used by default for change log events
    d. Best performance, since there are fewer target operations
    e. Edge cases could produce temporary incorrect results (the target could be blissfully out of sync until the next full sync)
2. Recalculate (stateless)
    a. There is no "add member" event, there is "check group a:b:c for subject 123".
    b. Check the state on the target, compare with grouper, and issue the correct operation
    c. This is used for:
        i. Full sync, Group sync, User sync
        ii. UI provision buttons pressed on full, group, user, or membership sync'ing
        iii. Error correction. If an operation produces an error, do a recalc on that object
        iv. If state doesnt match in a stateful operation (described above)
    d. Performance not as good
    e. Target will be in the correct state
    f. Events can be replayed without problems. Idempotent

If you want all operations to be stateless, you can change this provisioning setting:

grouper-loader.properties

```
provisioner.<provisionerConfigId>.recalculateAllOperations = true
```

Workflow

Here is the workflow of a provisioner

Step	Name	Description
0	ESB change log consumer generates events	the esb events from change log are handed to provisioner, and it kicks off the incremental provisioner. all provisioning jobs make sure no other jobs are running, unless they are asynchronous then it is ok

1	process incoming messages, add actions	check the messaging queue for this provisioner queue (every provisioner will have one by default). process messages (e.g. from UI: full sync, group syncs, user syncs, or membership syncs). Or non blocking sync results. Or when groups/folders are marked as provisionable. Or other reasons. Note, dont trust the provisioning sync state for these. Recalc things from target
2	propagate provisioning attributes	make sure that provisionable settings (attributes under the covers) are propagated to all descendent objects in folders
3	convert esb change log events to provisioning actions	convert esb events from change log into actions in the provisioner. By default these will translate to actions without recalc. i.e. if a membership is added in grouper, then try to add it to the target
4	filter by full sync	if a full sync started after this event happened, filter it
5	recalc events during full sync	if events happen between when a (a)synchronous full sync started, and before it finished, do a full recalc
6	look for sync errors and add events	if there are sync errors that happened a (configurable randomness), find them and add esb events to list
7	filter non recalc actions that are captures by recalc	if there are non recalc group/entity/membership actions that are being recalced, then filter them
8	retrieve sync group objects	retrieve all the group sync objects for all events
9	query recalc group provisioning attributes	retrieve provisioning attributes for recalc groups and adjust sync objects
10	filter if not provisionable	if the sync group object says not provisionable for a non-recalc event, then filter it
11	filter by group sync	if a group has had a group sync after this event took place, and the event is related to the group, then filter it (dont need to do this action anymore)
12	convert events to group sync	if there are enough events over a configurable threshold, convert events to group sync
13	convert events to full sync	if there are enough events over a configurable threshold, and weighted by type of event, convert events to full sync
14	recalc events during group sync	if events happen between when a (a)synchronous group sync started, and before it finished, do a full recalc action
15	retrieve sync membership objects	retrieve sync membership objects for all memberships in events
16	retrieve sync member objects	retrieve sync member objects for all members in the queue
17	retrieve grouper incremental data	retrieve grouper incremental data for groups, entities, and memberships, process the data into wrappers
18	filter by unneeded actions	if the state of the action is not expected to change the target based on sync objects, filter the action
19	convert inconsistent events to recalc	e.g. if a delete of membership is issued, but the membership exists, then just recalc
20	copy incremental state to wrappers	take the incremental state and copy to the wrapper objects
21	retrieve subject link	resolve subjects for subject link if recalc or for subjects missing data
22	translate grouper groups/entities data to target format	take the grouper groups, entities and translate to the target format
23	manipulate grouper groups/entities attributes	based on configs manipulate the defaults, types, etc for grouper target translated group/entity attributes and fields
24	matching id of grouper groups/entities	calculate the matching id of grouper translated group/entity data
25	index matching id of grouper groups/entities	take all the matching ids of grouper groups/entities and index those for quick lookups

26	validate objects and filter invalid	look at objects and validate them and filter invalid objects
27	retrieve incremental data from target	for recalc, get data from target
28	target object attribute manipulation	process the target objects for attribute manipulation
29	target id incremental objects	assign target ids to target objects
30	create missing groups / entities	create missing groups / entities
31	retrieve target group and entity link	based on data retrieved, update the group and entity link
32	translate grouper memberships to target	translate grouper memberships to target after the link data is resolved
33	manipulate grouper memberships attributes	based on configs manipulate the defaults, types, etc for grouper target translated membership attributes and fields
34	matching id of grouper memberships	calculate the matching id of grouper translated memberships data
35	index matching ID of grouper and target objects	take the matching IDs of the grouper side and the target side and index objects in the data index
36	compare target objects	look at the grouper side and the target side and compare and generate the target actions
37	send changes to target	send inserts updates and deletes to target
38	mark messages as read	the provisioner should take the messages and process them and acknowledge the message...

Messaging to control provisioning externally (e.g. from UI or WS)

You can send messages to provisioners to control provisioning, though there is no feedback about what happened. Well, you could poll the messaging system to see if the message has been processed or deleted. Generally this should take effect in a minute or two. The default implementation will select all messages at once for all provisioners with the assumption that there arent that many at any given time.

Queue for provisioning: grouperProvisioningControl_provisionerName (e.g. grouperProvisioningControl_myPspngProvisioner)

Java to send a message:

```
    ProvisioningMessage provisioningMessage = new ProvisioningMessage();
    provisioningMessage.setFullSync(true);
    String message = provisioningMessage.toJson();

    GrouperMessagingEngine.send(
        new GrouperMessageSendParam().assignGrouperMessageSystemName(GrouperBuiltinMessagingSystem.BUILTIN_NAME)
          .assignQueueType(GrouperMessageQueueType.queue)
          .assignQueueOrTopicName("grouperProvisioningControl_myPspngProvisioner")
          .assignAutocreateObjects(true)
          .addMessageBody(message));
```

Message to do a full sync:

```
{"fullSync":true,"fullSyncType":"optionalFullSyncType"}
```

Message to sync some groups:

```
{"groupIdsForSync":["abc123","def456"]}
```

Message to sync some users:

```
{"memberIdsForSync":["abc123","def456"]}
```

Message to sync from memberships:

```
{"membershipsForSync":[{"groupId":"abc123","memberId":"jkl789"},{"groupId":"def456","memberId":"qwe543"}]}
```