Grouper Web Services for developers

This page is intended for developers on the Grouper team only. These are instructions on how to edit the Grouper code itself, not for users of the Grouper software. Thanks!

Standards

- All options in client must be listed in grouper-misc/grouperClient/conf/grouper.client.usage.example.txt
  - Brackets mean optional
  - Pipes describe the options
- All client options should be tested in: /grouper-ws/src/test/edu/internet2/middleware/grouperClient/poc/GrouperClientWsTest.java
- All client options have the same name as the JSON/XML which have the same name as java variables in the methods
- All fields and methods should have javadoc with the description of the element
- All data setup for the sample capture should be in /grouper-ws-test/src/test/edu/internet2/middleware/grouper/ws/samples/SampleCapture.setupData()
  - The data setup should not require a database reset, and should not need a cache clear (since run from java and WS from different JVMs)
- In the grouper client, if a multi-valued argument can have commas, then use indexing in the param names, e.g. messageBody0=whatever messageBody1=somethingElse
- All classes which generate SOAP should just have String, Array, or Javabean types. Avoid other primitives. e.g. /grouper-ws/src/grouper-ws_v2_4/edu/internet2/middleware/grouper/ws/soap_v2_4/GrouperService.java
- All classes used to marshal to/from REST (JSON/XML/etc) should just have String, Array, or Javabean types. Avoid other primitives. e.g. grouper-misc\grouperClient\src\java\edu\internet2\middleware\grouperClient\ws\beans\WsRestReceiveMessageRequest.java
- GrouperServiceLogic is where primitives are used: /grouper-ws/src/grouper-ws/edu/internet2/middleware/grouper/ws/GrouperServiceLogic.java

To augment a web service

This is an example for adding point in time to getMembers

- In GrouperService, add two params to getMembers and getMembersLite: String pointInTimeFrom, String pointInTimeTo
  - If they are equal, then do it at one point in time, if both filled in, then a range, if one filled in, then assume the other is the minimum point in time or right now
  - Note, this class is what our WSDL is based on, so dont add another extra methods there
  - Note, this is what the SOAP messages will look like, so the only inputs or outputs should be valid fields: strings, beans (with valid fields), bean arrays or string arrays
  - Note, the names of the fields are the names in the soap messages, so dont abbreviate, and use something descriptive. Since this is not just a string, but in the WS it is bassed as a string, make sure it is in the javadoc that the format is yyyy/MM/dd HH:mm:ss.SSS
  - Make sure there is good javadoc about these params
  - Convert them to timestamps (or whatever), and pass to grouperServiceLogic. Note, do this like it is done in other places so everything is consistent (i.e. numbers, booleans, timestamps, enums, etc)

```
Timestamp pointInTimeFromTimestamp = GrouperServiceUtils.stringToTimestamp(pointInTimeFrom);
```

- In GrouperServiceLogic, add the two params to getMembers and getMembersLite
  - Dont add extra methods to this class either
  - Make the params the real type (i.e. Timestamp), but use the same param name as GrouperService
  - Add something to theSummary (note, if it could be huge, i.e. arrays, abbreviate somehow), e.g. + "\n, pointInTimeFrom: " + pointInTimeFrom
    + ", pointInTimeTo: " + pointInTimeTo
  - Use the same Javadoc
  - Note that these two new params might not be compatible with all existing params (i.e. if you send point in time timestamps, maybe you arent allowed to use other params like immediate/non-immediate), not sure. If that is the case, do validation in the try catch after the "summary" is created, and throw a WsInvalidQueryException
  - On that method (getMembers) is where you do the logic. Either you can overload the API so that you dont have to do any logic in GrouperServicLogic (i.e. group.getMembers(field, sources, null, pointInTimeFrom, pointInTimeTo); Adding to the API might be nice, but maybe it is somewhere else and you can do an if statement...
- At this point, there are a bunch of things to do, you can do them in any order... (as long as things compile etc)
- Unit test this in GrouperServiceLogicTest
- Edit the beans: WsRestGetMembersRequest, WsRestGetMembersLiteRequest. Add the fields, getters and setters, must be strings, not timestamps, and make sure the javadoc (copied from above) is on the fields, getters, and setters. Also, the name of the field should be what is in the WS spec, i.e. pointInTimeFrom, pointInTimeTo
- Edit GrouperServiceRest.getMembers and GrouperServiceRest.getMembersLite
- Edit GrouperClient.getMembers(), get the arg in the standard way for that datatype, e.g.

```
    {
      Timestamp pointInTimeFrom = GrouperClientUtils.argMapTimestamp(argMap, argMapNotUsed, "pointInTimeFrom");
      gcGetMembers.assignPointInTimeFrom(pointInTimeFrom);
    }
```

- Edit GcGetMembers
    - Note, this is not a javabean since the setter needs to return "this" for chaining. So we have assigners and retrievers, so we dont break the javabean contract. We also have adders for things that are multivalued (in this case we dont). So, add timestamp fields and assignPointInTimeFrom(timestamp) methods. Make sure the javadoc is consistent with the WS javadoc
    - If there are things to validate (i.e. if that param is not compatible with other params, then use the validate() method. Note that the timestamp format validation is already done, no need to do that anywhere
    - In the execute() method, convert to String in standard way, and pass to the REST bean:

```
getMembers.setPointInTimeFrom(GrouperClientUtils.dateToString(this.pointInTimeFrom));
```

- Merge your changes (in this case fields, getters, setters), into the client version of WsRestGetMembersRequest. Make sure the Javadoc is intact
- Add a test case (or normally I just augment an existing test case, either way) to GrouperClientWsTest
- Change the doc in grouper.client.usage.example.txt. Probably dont need to change the example call, but document the param,

```
[--disabledTime=yyyy/mm/dd hh:mi:ss] [--enabledTime=yyyy/mm/dd hh:mi:ss]
```

- Edit the change log for the release that the grouper.client.usage.example.txt file was updated. e.g. https://spaces.at.internet2.edu/display/Grouper/Grouper+changes+v1.6here is the one for 2.0
- Edit the doc for that operation, e.g. for getMembers, and say it is for 2.0+ or whatever, e.g. a bullet under features which says something like

```
- You can pass in pointInTimeFrom and pointInTimeTo to get the memberlist at a certain
  point in time in the past, or in a date range.  This should be formatted: yyyy/mm/dd hh:mi:ss
```

- Make sure that there is no coresoap (package) in use in src/grouper_ws_v1_6, src/grouper_ws_v2_0, etc.
- Email Chris when done and committed so he can review the changes.
- Should probably file a separate Jira report for changes to a web service operation so it gets communicated explicitly
- If anything is missing, please edit this doc. Thanks!

To add a new web service

This is an example of adding getAuditEntries web service

- In GrouperService, add getAuditEntries method.
    - Note, this class is what our WSDL is based on
    - Note, this is what the SOAP messages will look like, so the only inputs or outputs should be valid fields: strings, beans (with valid fields), bean arrays or string arrays
    - Note, the names of the fields are the names in the soap messages, so dont abbreviate, and use something descriptive.
    - Make sure there is good javadoc about the parameters
- In GrouperServiceLogic, add the implementation
- Add unit tests in GrouperServiceLogicTest
- Add Rest Request classes: WsRestGetAuditEntriesLiteRequest and WsRestGetAuditEntriesRequest
- Add getAuditEntriesLite and getAuditEntries in GrouperServiceRest class
- Add new enum audits in GrouperWsRestGet. The word audits here means the path will end with audits
- Rest samples are within the same module: grouper-ws. Add Rest tests WsSampleGetAuditEntriesRest and WsSampleGetAuditEntriesRestLite
- Add the new classes to WsRestClassLookup
- For Soap web services, we will need to generate new wsdl
    - Run java2wsdl ant goal from grouper-ws module
    - Refresh grouper-ws-java-generated-client module and the latest wsdl file should have been changed
    - Run wsdl2java ant goal from grouper-ws module. This will generate java client for the new endpoint
- To test the new soap endpoint, write tests in grouper-ws-java-generated-client module. WsSampleGetAuditEntries and WsSampleGetAuditEntriesLite
- Add tests in GrouperClientWsTest
- Add operation in GrouperClient.java in grouperClient module
- Add GcGetAuditEntries in grouperClient module
- Add WsRestGetAuditEntriesRequest and WsGetAuditEntriesResults in grouperClient module
- Make sure that there is no coresoap (package) in use in src/grouper_ws_v1_6, src/grouper_ws_v2_0, etc.
- Email Chris when done and committed so he can review the changes.
- Should probably file a separate Jira report for changes to a web service operation so it gets communicated explicitly
- If anything is missing, please edit this doc. Thanks!