# Grouper update memberships lite UI



I will put together a UI for lite group membership management

Here is a screen movie of the app, here is one with integration with another app. To view, you need the free xvid codec

### Features

- Link directly to a group id or name (cant search for a group)
- If the user has read and update, the user can use the screen
- It will show the group information (name, description, etc), and the members (paged)
- You can add a user with a combobox to search for subjects
- You can delete members with checkboxes (if immediate members)
- Will not (at least initially) have information about immediate/effective, will be simple
- Externalized text (same as current UI)
- Modal dialog component: SimpleModal
- "Throbber" component: BlockUI
- You can bulk add or replace members based on a textfield of newline delimited subjectIds or subjectIdentifiers
- You can filter the user list based on subject (to see if a subject is in the list)
- Support back/forward/bookmark
- There are menus: dhtmlx menu
- Comboboxes: dhtmlx combo
- Advanced paging

### Architecture

- Ajax / json / ws / JSP (loose rest)
  - Note: I will use an ajax framework which facilitates manipulating the screen from Java (not GWT)
  - There will be no (or very little) business logic in Javascript
  - The WS will return a framework JSON object model which is not Grouper specific
  - A noteworthy aspect is that sections of the screen are replaced by JSP's which are evaluated as part of web service requests (not web requests). Multiple sections can be replaced with one request
- Will use json-lib.jar (note, different than the current WS json library [better])
- This will be a zip file you can unzip on top of Grouper web services, configure, run.
  - Note: if you want to run separately, you can configure WS to not run this UI, and you can configure the UI to not accept non-UI WS
- Will use the dhtmlx ajax gui tool library, jquery, walter zorn tooltips
  - Note: there will be rich GUI components on the screen for improved usability / performance
- Will prevent CSRF attacks
- Not sure if there will be cookies
- Will support up to date browsers (IE 6+, FF, Safari, Chrome)
- Will be straighforward, easy to customize, easy to develop/debug
- Will be compatible with Grouper 1.4.2+
- No-build development environment. Used with JavaRebel, you do not need to restart or deploy the webapp very often. The CVS structure is a webapp itself. The grouper API is not included (like the current UI), there is an ant target to build and copy Grouper, and an ant target to get the libs in WEB-INF/lib, other than that, not much needed for development. There will be a "dist" target to make jars and wars.

Note: if we are ok with this architecture, maybe some of the new features of Grouper could have screens here... it will be a lot easier for me to quickly make screens...

### Coding samples

To make ajax work, there is a common method:

```
function ajax(theUrl, options) {
```

Note, there is no callback after the ajax is done (usually there is with business logic in Javascript), instead that is all handled on the server side. So a button that calls ajax looks like this

```
    <a href="#" onclick="ajax('SimpleMembershipUpdate.deleteSingle?memberId=${theMember.uuid}'); return false;"
>The link</a>
```

Note that the class and method are listed there to reduce the levels of indirection (normally you will see a one-to-one mapping anyways). The security restriction however is that the class must be in the edu.internet2.middleware.grouper.grouperUi.serviceLogic package, and the method must have this signature: public void methodName(HttpServletRequest, HttpServletResponse) Then in the Java side, you fill out an object model which consists of ordered actions to occur on screen. This will be things like, replace this div with this JSP, change this form element value, hide this, show that, etc.

```
GuiResponseJs guiResponseJs = GuiResponseJs.retrieveGuiResponseJs();
guiResponseJs.addAction(GuiScreenAction.newAlert("The member was deleted: " + guiMember.getSubject().
getDescription()));
guiResponseJs.addAction(GuiScreenAction.newInnerHtmlFromJsp("#simpleMembershipResultsList",
  "/WEB-INF/grouperUi/templates/simpleMembershipUpdate/simpleMembershipMembershipList.jsp"));
```

Note in that case the JSP is parsed during the web service call, and contents are shipped back in JSON in the guiResponseJs object model...

The rich client controls are easy to do as well, just have helper methods to set things up, e.g. a combobox:

```
<%-- note, this div will be the combobox --%>
<div id="simpleMembershipUpdateAddMember" style="width:400px;"></div>
<script>
guiRegisterDhtmlxCombo('simpleMembershipUpdateAddMember', 400,
true, "../app/SimpleMembershipUpdate.filterUsers" );
</script>
```

Then on the serverside, do a method, call helper methods to build the XML, and tell the controller not to send the GuiResponseJs JSON object:

```
public void filterUsers(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) {

  final Subject loggedInSubject = GrouperUiJ2ee.retrieveSubjectLoggedIn();


  GrouperSession grouperSession = null;

  String searchTerm = httpServletRequest.getParameter("mask");

  try {
    grouperSession = GrouperSession.start(loggedInSubject);


    Set<Subject> subjects = null;

    StringBuilder xmlBuilder = new StringBuilder(GuiUtils.DHTMLX_OPTIONS_START);

    if (StringUtils.defaultString(searchTerm).length() < 2) {
      GuiUtils.dhtmlxOptionAppend(xmlBuilder, "", "Enter 2 or more characters", null);
    } else {
      subjects = SubjectFinder.findAll(searchTerm);
    }

    for (Subject subject : GrouperUtil.nonNull(subjects)) {
      String value = GuiUtils.convertSubjectToValue(subject);

      String imageName = GuiUtils.imageFromSubjectSource(subject.getSource().getId());
      String label = GuiUtils.convertSubjectToLabel(subject);

      GuiUtils.dhtmlxOptionAppend(xmlBuilder, value, label, imageName);
    }

    xmlBuilder.append(GuiUtils.DHTMLX_OPTIONS_END);

    GuiUtils.printToScreen(xmlBuilder.toString(), "text/xml", false, false);

  } catch (Exception se) {
    throw new RuntimeException("Error searching for members: '" + searchTerm + "', " + se.getMessage(), se);
  } finally {
    GrouperSession.stopQuietly(grouperSession);
  }

  //dont print the regular JSON
  throw new ControllerDone();

}
```

sda