Grouper subject picker

Starting with Grouper v2.4, the Lite UI is no longer available in the standard setup, and the subject picker UI is not available.

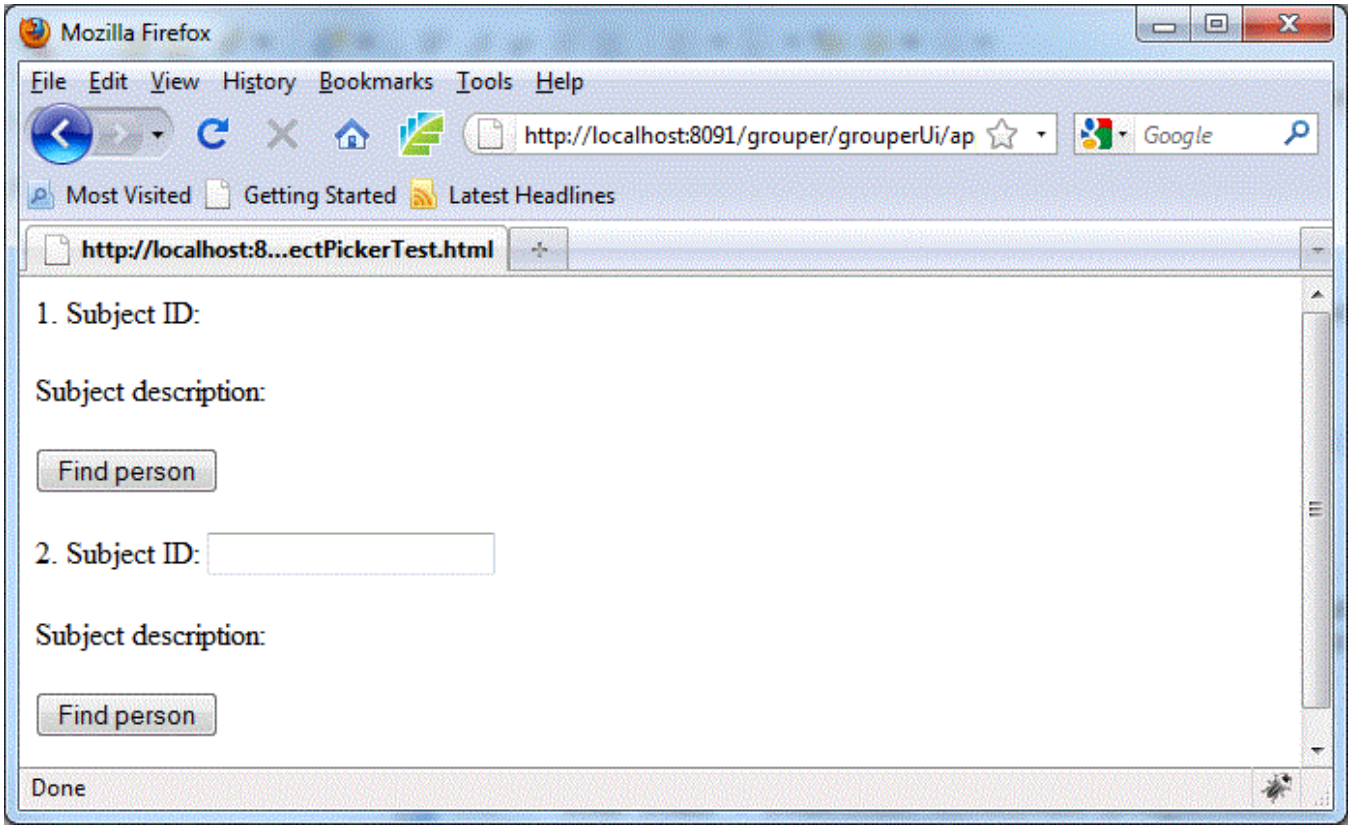
Grouper Subject Picker

Grouper LITE UIs

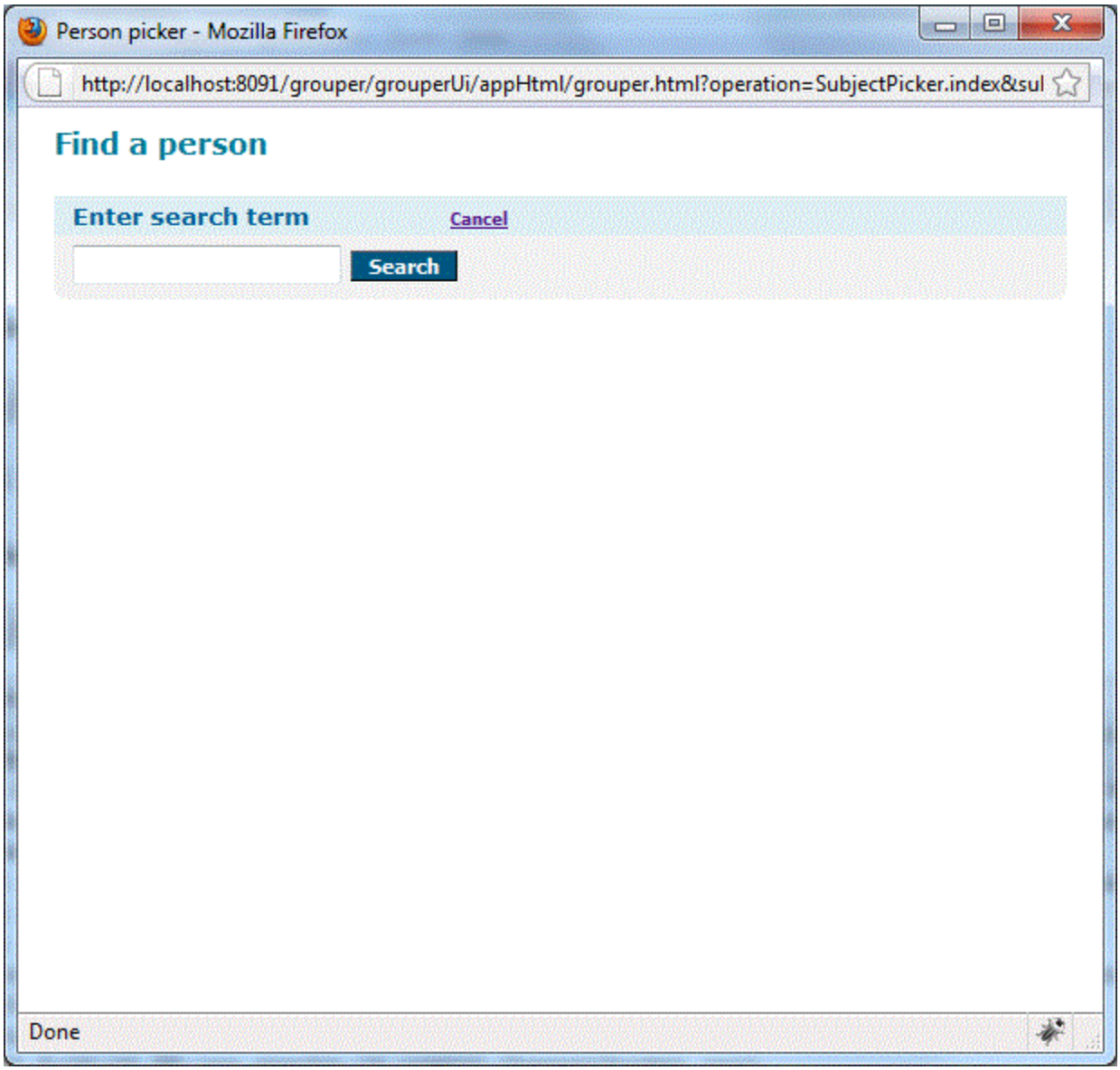
The Grouper subject picker picker is a lightweight embeddable UI component which can help external applications (or Grouper itself) to find subjects to put into textfields or hidden fields or drop downs or whatever. This is available for Grouper v.1.6+.

Example

Here is a simple app screen. The URL says grouper, but it could be any URL.

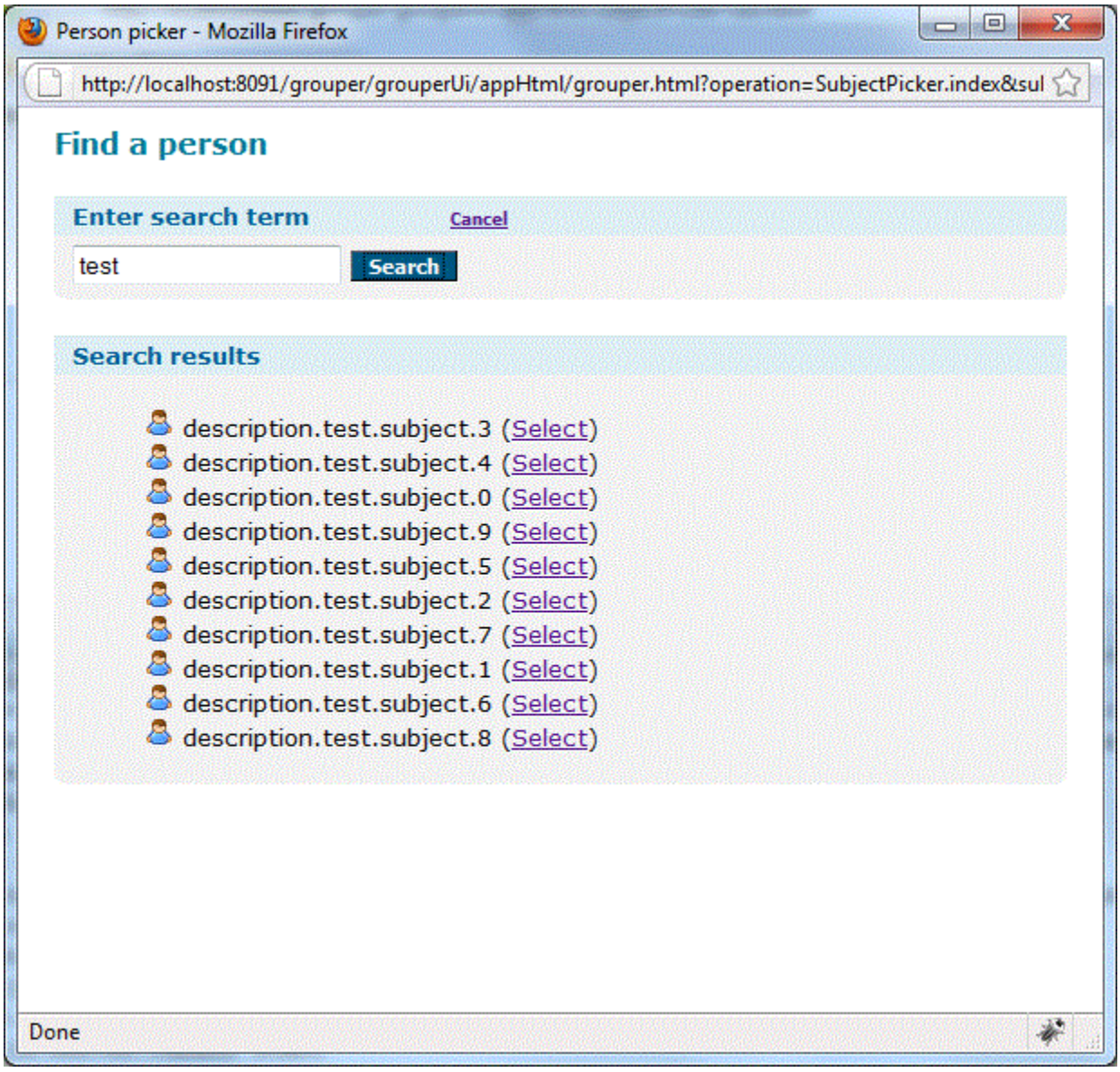


There are two fields which need subjects selected, one just goes to a label on the screen, one to a textfield. When clicking on find person, this screen appears as a popup



This screen above is hosted in the Grouper UI. It is assumed there is singlesign on between the two applications for good usability. If there isnt, then the user would need to login to Grouper (I dont think the subject picker can be used anonymously). Since it is a popup there should be fewer problems with cross site browser plugin blockers (e.g. for ajax)

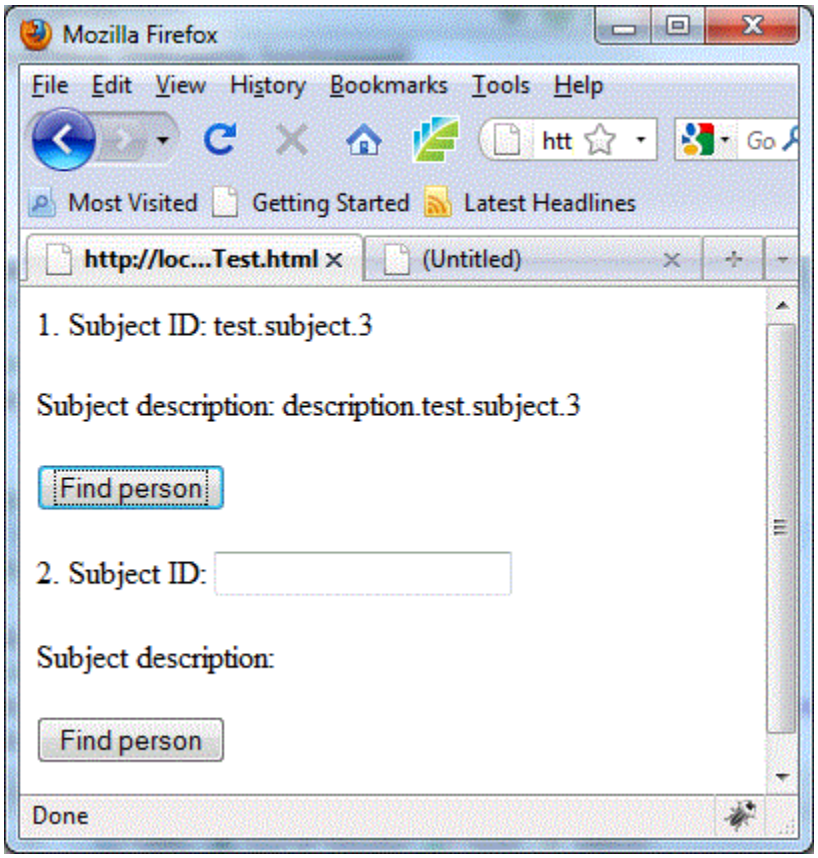
Now a subject can be searched for. Its results will populate with ajax. Note: there is a subject picker "name" that links the picker with a config file. That config file specifies which sources are to be used for searching, and if there is a group the results need to be in (e.g. only people who are active employees should be returned).



Now another search can take place, or one of the subjects can be selected. If the subject is selected, then the window will close, and the calling window will get a javascript call with the subject chosen (the label, the id, and the subject object (if configured to send that). The calling page can easily take that and put it in a hidden field, a textfield, a label, etc with a little javascript. We cna give examples of any javascript people will need.

So the bottom line is that with some Javascript, and some configuring of properties files in the Grouper UI, any web application can have a customized subject picker. Here is the calling page with the subject selected

---



Default settings

Subject pickers (you can have multiple at your site, customized for each application), have a lot of settings which are available. So to avoid having to set common settings in each subject picker instance, there are default settings. In the media.properties you can configure defaults for any of the subject picker settings. Note, since this is the media.properties, Grouper has defaults built in, and you can override those defaults:

```
###################################
## Subject picker


## subject picker config defaults

# if the subject should be sent back to the calling page in javascript object
subjectPicker.defaultSettings.sendSubjectJsonToCallback = true

#comma separated css urls (relative or absolute) for skinning this subject picker
subjectPicker.defaultSettings.extraCss =

# when the subject object is sent in Javascript to the caller, which fields or attributes should be sent
subjectPicker.defaultSettings.subjectObject.include.subjectId = true
subjectPicker.defaultSettings.subjectObject.include.sourceId = true
subjectPicker.defaultSettings.subjectObject.include.name = true
subjectPicker.defaultSettings.subjectObject.include.typeName = true
subjectPicker.defaultSettings.subjectObject.include.description = true
#comma separated list of subject attirbutes or INCLUDE_ALL_ATTRIBUTES for all
#subjectObject.include.attributes = loginid,lfname
#subjectObject.include.attributes = INCLUDE_ALL_ATTRIBUTES
subjectPicker.defaultSettings.subjectObject.include.attributes =

# put sourceIds to search in, or leave blank for all
subjectPicker.defaultSettings.searchInSourceIds =

## You can configure per source how the subjects appear on screen, and customize per subject picker instance as
well
# Increment the index (0, 1, 2, etc) to configure multiple sources
#source id we are configuring
subjectPicker.defaultSettings.sourceProperties.sourceId.0 =
#this is the expression language of how the subject result should appear on screen
subjectPicker.defaultSettings.sourceProperties.subjectElForSource.0 =

# max results that can be retrieved before the group filter resultsMustBeInGroup is applied
subjectPicker.defaultSettings.maxSubjectsResultsBeforeGroupSearch = 800

# max results that can be retrieved
subjectPicker.defaultSettings.maxSubjectsResults = 200

#if results must be in group, or blank for no check.  e.g. put your active employee group here
subjectPicker.defaultSettings.resultsMustBeInGroup =

# is an actas should be applied for group operations.  Generally this is GrouperSystem, though could be anyone,
or blank
# to act as the logged in user
subjectPicker.defaultSettings.actAsSourceId = g:isa
subjectPicker.defaultSettings.actAsSubjectId = GrouperSystem
```

In the media.properties, you need to specify if you are putting your configs for each subject picker in the classpath or in a hardcoded director. Also you can say what group by default can use subject pickers

```
#users must be in this group to be able to login to the subjectPicker UI
require.group.for.subjectPicker.logins=


# if putting config files on file system and not classpath, then put the files here
subjectPicker.confDir = c:/temp/subjectPicker
```

Then for each subject picker, assign a subject picker name, some alphanumeric or underscore that will tie the subject picker in the URL to the property file. Make a file in the directory above or on classpath: /subjectPicker/subjectPickerName.properties

This file can be blank, or could have any of the default settings above overridden:

```
 ###################################
## Subject picker

# if putting config files on file system and not classpath, then put the files here
subjectPicker.confDir = c:/temp/subjectPicker


## subject picker config defaults

# if the subject should be sent back to the calling page in javascript object
sendSubjectJsonToCallback = true

#comma separated css urls (relative or absolute) for skinning this subject picker
extraCss = ../public/assets/css/subjectPickerExample.css
#extraCss = http://localhost:8091/grouper/grouperUi/public/assets/css/subbjectPickerExample.css

# when the subject object is sent in Javascript to the caller, which fields or attributes should be sent
subjectObject.include.subjectId = true
subjectObject.include.sourceId = true
subjectObject.include.name = true
subjectObject.include.typeName = true
subjectObject.include.description = true
#comma separated list of subject attirbutes or INCLUDE_ALL_ATTRIBUTES for all
subjectObject.include.attributes =
#subjectObject.include.attributes = loginid,lfname
#subjectObject.include.attributes = INCLUDE_ALL_ATTRIBUTES

# put sourceIds to search in, or leave blank for all
#searchInSourceIds = jdbc, g:isa

## You can configure per source how the subjects appear on screen, and customize per subject picker instance as
well
# Increment the index (0, 1, 2, etc) to configure multiple sources
#source id we are configuring
sourceProperties.sourceId.0 = g:isa
#this is the expression language of how the subject result should appear on screen
sourceProperties.subjectElForSource.0 = ${subject.id}

# max results that can be retrieved before the group filter resultsMustBeInGroup is applied
maxSubjectsResultsBeforeGroupSearch = 800

# max results that can be retrieved
maxSubjectsResults = 800

#if results must be in group, or blank for no check.  e.g. put your active employee group here
resultsMustBeInGroup =

# is an actas should be applied for group operations.  Generally this is GrouperSystem, though could be anyone,
or blank
# to act as the logged in user
actAsSourceId = g:isa
actAsSubjectId = GrouperSystem
```

Not only the settings can be configured, but also the text. This is in the nav.properties so it can in different languages or locales based on browser preferences. Also this means Grouper has defaults, institutions can override those defaults (globally), and also further customize per subjectPicker. Here are the defaults in the nav.properties:

```
#####################################
## Subject picker defaults
#####################################
subjectPickerDefault.header = Find a person
subjectPickerDefault.title = Person picker
subjectPickerDefault.searchSectionTitle = Enter search term
subjectPickerDefault.searchButtonText = Search

subjectPickerDefault.resultsSectionTitle = Search results

subjectPickerDefault.noSearchTerm = Enter a search term

subjectPickerDefault.noResultsFound = No results found

subjectPickerDefault.tooManyResults = Too many results, please narrow your search.  Note, a partial listing
might still display.

subjectPickerDefault.cancelText = Cancel
```

Based on the subject picker name, you can customize per subject picker in your local nav.properties

```
#####################################
## Subject picker test for subject picker with name subjectPickerName
#####################################

subjectPicker.subjectPickerName.title = Person picker
```

Note that the same rules apply in the nav.properties as other apps, you can use the same syntax to make infodot help icons appear

Using

Once you have the subject picker configured in the Grouper UI, you can use it from an application. Note, with this setup with Grouper as an external application, there is no way to tell that the user is coming from the application you think it is coming from. This is one reason why authentication is required, and you can clamp down on who can use the service (e.g. active members of the community). If we want more security we could pass tokens with web services or something. In the HTML of the web app that needs a subject picker, make a button with an onclick that opens the picker. You also need to pass in the element name on the screen which had its button pressed (since multiple elements could need a subject picker).

```
 <button onclick="theWindow = window.open('http://localhost:8091/grouper/grouperUi/appHtml/grouper.html?
operation=SubjectPicker.index&subjectPickerName=whatever&subjectPickerElementName=subject1','newWindow',
'scrollbars,resizable'); theWindow.focus(); return false;"
    >Find person</button>
```

This will popup the subject picker. Note this is not a modal popup, it is assumed the user will use it or close it. If they forget it and use it later, that is another reason the element name is returned (so it doesnt mangle another screen). Anyways, have a javascript in the calling page for the callback for when a subject in the popup is selected. Here is an example that handles two subject pickers on one page, and escapes HTML on the screen

```
<script>

 function grouperSubjectSelected(elementName,subjectId,subjectDescription, pickerResultSubject) {

   subjectDescription = escapeHtml(subjectDescription);
   subjectId = escapeHtml(subjectId);
   if (pickerResultSubject.subject.attributes.loginid) {
     alert(pickerResultSubject.subject.attributes.loginid[0]);
   }
   if (pickerResultSubject.subject.attributes.lfname) {
     alert(pickerResultSubject.subject.attributes.lfname[0]);
   }
   if (pickerResultSubject.subject.name) {
     alert(pickerResultSubject.subject.name);
   }

   //alert(pickerResultSubject);
   document.getElementById(elementName + "DescriptionSpanId").innerHTML = subjectDescription;

   if (elementName == 'subject1') {
     document.getElementById(elementName + "IdSpanId").innerHTML = subjectId;
   } else if (elementName == 'subject2') {
     document.getElementById(elementName + "IdSpanId").value = subjectId;
   } else {
     alert("ERROR: Cant find elementName: " + elementName);
   }

 }

 /** convert input into a non-null string */
 function escapeHtml(input) {
   input = input.replace(/&/g,  "&amp;");
   input = input.replace(/</g,  "&lt;");
   input = input.replace(/>/g,  "&gt;");
   return input;
 }
</script>
```

Note, you could use a library like Jquery to make this easier. Also, it is nice if the elements you are putting the subjects into have id's (not just names), since it is more browser neutral (IE has issues with names).

Algorithm

There is a max number of subjects to display, which is configurable. Sources also have a max. Regardless if you search for someone's subject id or net id, that result will be at the top (by subjectId and subjectIdentifier). So if the netId is "a", and that throws an exception in the source for too many results, the subject with netId "a" will still be in the results to choose. the subjectId or sourceId match will always be at the top. The results are sorted by display label.

If the number of results is less than the max results by source (i.e. no exception thrown), but more than the max for the subject picker, then an alert will display to the user, and a partial listing of results will show on the screen.

Since subjects can be limited by group, the max number of results should be less than 1000 (I think 200). This is because finding a subject who is a member of a group is an expensive operation since the matching subjects need to be returned from the source, then the subejcts need to be queried (in batches) to the grouper registry to see which are in the group. You can test it out for yourself to see what a reasonable limit is.

Security

Note, there are browser security restrictions that prohibit applications with different domain names or ports from accessing other windows of the browser. In this case the subject picker is a popup, and it needs to call a javascript function in its "opener". So if Grouper and the application have different domain names or ports, then you need to workaround this. Set this in the settings to an HTML file which resides in the application:

```
# put a URL here where the result (subjectId, sourceId, name, description) will be submitted back
# blank if same domain and just call opener directly
submitResultToUrl =
```

That HTML page (or servlet or whatever) will be submitted to with an HTTP GET with the results of the picker. An example of the HTML page is attached, it just gets the args from the URL, and calls the opener function which will work since it is the same domain and port, and closes itself.

To do

- Note that there is no sorting or paging here. This is because it is hard to sort and page across multiple sources, and even with a single source the subject API doesnt support it. The admin UI handles this by retrieving a lot of results and sorts and pages in Java. For the screen the initial hope is that if there are 200 results on the screen, hopefully the person can be found. Also, there are other tricks of bringing the netId and subjectId to the top