Grouper configuration in the database and UI

Grouper allows configuration, which is normally in config files, to be stored in the database. This is the preferred approach because:

- Configuration in the database makes the configuration consistent in an environment (otherwise the config files need to be kept in sync for the UI servers, WS servers, daemons, and GSH).
- Editing the configuration in the UI prevents configuration problems and helps users to configure Grouper easier and more reliably.
- It allows the administrator to revert the configuration if needed

How it works

In the hierarchy section of the config file will be an option to use the database

e.g. in grouper.base.properties

OLD

```
grouper.config.hierarchy = classpath:grouper.base.properties, classpath:grouper.properties
```

NEW

```
grouper.config.hierarchy = classpath:grouper.base.properties, classpath:grouper.properties, database:grouper
```

This "database:grouper" will check the grouper database table: grouper_config for overlays. This should be last so that when you edit something in the UI, it will take effect.

Column	Description
ID	UUID
CONFIG_FILE_NAME	Config file name without the "base", e.g. grouper.properties or grouper-loader.properties
CONFIG_KEY	key of the config, e.g. grouper.env.name
CONFIG_VALUE	value of the config, on the right of the equals sign in the properties file. Note, currently there is a 4k limit to size of value
Note: there are more columns in this table which are currently unused	

Note: if you have configs you want to differ in different components (e.g. ws/ui/daemon), you need to set those in the config files, and not in the database, which will span all components (e.g. UI/WS/Daemon) that use that database

Config files

1. grouper.properties
2. grouper-loader.properties
3. subject.properties
4. grouper.client.properties (if there is a grouper.hibernate.properties in the classpath)
5. grouper.cache.properties
6. grouper-ui.properties
7. grouper-ws.properties
8. grouper.text.en.us.properties (2.5.30+)

Edit the configuration in the UI

There is an editor in the UI

- The source IP address will need to configured or disabled in grouper-ui.properties

```
# The configuration editor in grouper is very sensitive.  If you can only allow certain source IP
addresses
# it will add another layer of security.  Otherwise allow 0.0.0.0/0 and all will be allowed
# If this configuration item is not filled in, then none are allowed
# you can configure multiple CIDR addresses or networks comma separated, e.g. 1.2.3.4/32, 2.3.4.5/24
grouperUi.configurationEditor.sourceIpAddresses =
```

Or put this SQL in the database

```
INSERT INTO grouper_config (id,config_file_name,config_key,config_value,config_comment,config_file_hierarchy,
config_encrypted,config_sequence,config_version_index,last_updated,hibernate_version_number,config_value_clob,
config_value_bytes) VALUES
('b0fd9db204ae4d07af881ee7b178f45c','grouper-ui.properties','grouperUi.configurationEditor.
sourceIpAddresses','0.0.0.0/0',NULL,'INSTITUTION','F',0,0,1604150796687,1,NULL,9);
commit;
```

The UI uses comments in the config file to describe the configuration, and will also use configuration metadata (described below)

Import the config

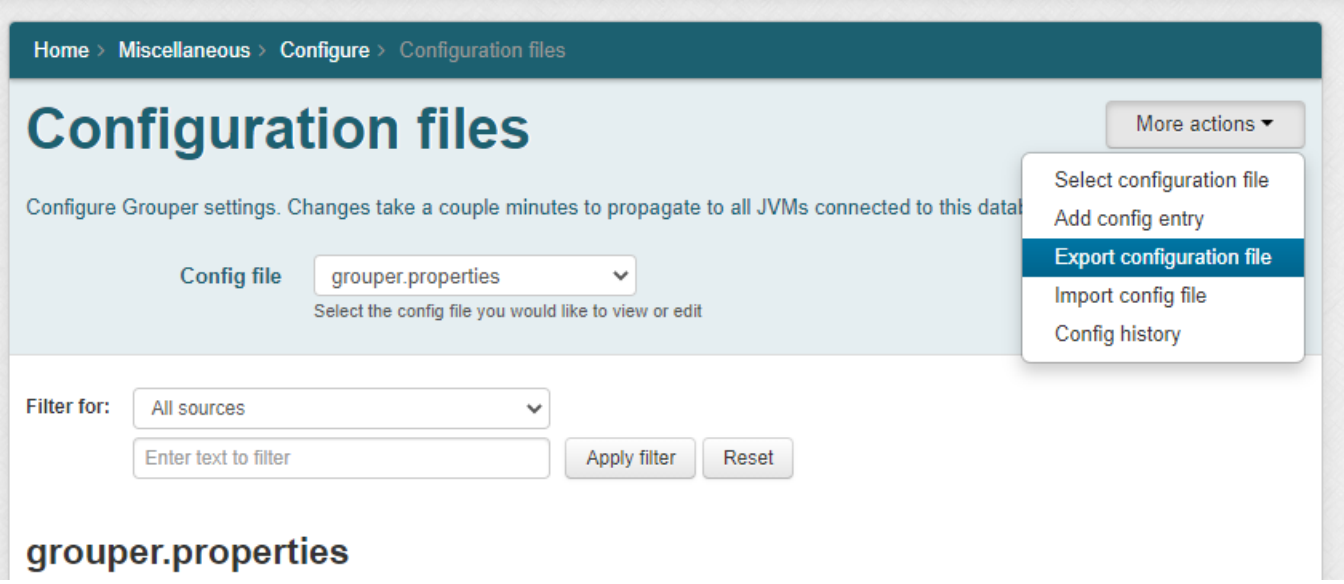
You can import the configuration in the database. You cannot import a "base" config file.

You can import from a config file. This will only be a config file in the classpath or on the filesystem. This is how to get started with this feature. Note, you will need to remove the config from other config files as well, e.g. web service configs.

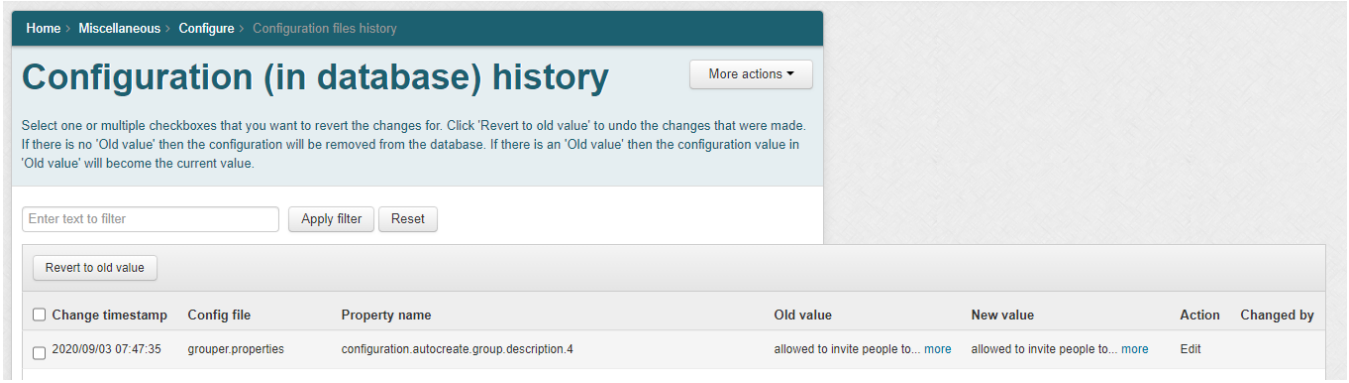
If we need to be able to export a properties file we can do that too.

You can import from a consolidated export from the filesystem.

Export the config (v2.5.34+)



Config history (v2.5.34+)



Configuration metadata

There can be JSON configuration metadata in the config files to help the UI correctly display and validate the configs. These metadata are in the "base" config files only. This is an example:

```
########################################
## inititalization and configuration settings
########################################

#if grouper should auto init the registry if not initted (i.e. insert the root stem, built in fields, etc)
#defaults to true
# {valueType: "boolean", sensitive: false}
registry.autoinit = true

#auto-create groups (increment the integer index), and auto-populate with users
#(comma separated subject ids) to bootstrap the registry on startup
#(note: check config needs to be on)
# {regex: "configuration.autocreate.group.name.[0-9]+", valueType: "group"}
#configuration.autocreate.group.name.0 = $$grouper.rootStemForBuiltinObjects$$:uiUsers
# {regex: "configuration.autocreate.group.description.[0-9]+", valueType: "string"}
#configuration.autocreate.group.description.0 = users allowed to log in to the UI
# {regex: "configuration.autocreate.group.subjects.[0-9]+", valueType: "subject", multiple: true}
#configuration.autocreate.group.subjects.0 = johnsmith
```

Note: just because there is validation doesnt mean a script cant be used.... e.g.

```
# group who can assign id index cols (also, wheel or root is allowed)
# {valueType: group}
grouper.tableIndex.groupWhoCanAssignIdIndex = $$grouper.rootStemForBuiltinObjects$$:canAssignIdIndex
```

Metadata name	JSON type	Default	Example value	Description
multiple	boolean	false	true/false	if comma separated values
mustExtendClass	String		a.b.c.SomeClass	If the value is a class and must extend another class
mustImplementInterface	String		a.b.c.SomeInterface	If the value is a class and must extend an interface
regex	String		^configuration.autocreate.group.description.[0-9]+$	If the key must match a certain regex
required	boolean	false	true/false	If a value must be provided
requiresRestart	boolean	false	true/false	If the JVM needs to be restarted when changing value
sampleValue	String		Something	An example value that shows the user how to configure

sensitive	boolean	false	true/false	If the value can be a password or something sensitive
valueType	String	String	attributeDef, attributeDefName, boolean, class, floating, group, integer, password, stem, string, subject	From enum ConfigItemMetadataType, the type of the value
formElement (new in 2.5 build)	String	text for most things and password for sensitive items	text, textarea, password, dropdown;	From enum ConfigItemFormElement
optionValues	String[]		if this is a dropdown then this is the option values available	

Configuration file layout

1. Config file starts with license, which has comments, none of which start with 10 hashes

```
#
# Copyright 2014 Internet2
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# Grouper loader uses Grouper Configuration Overlays (documented on wiki)
# By default the configuration is read from grouper-loader.base.properties
# (which should not be edited), and the grouper-loader.properties overlays
# the base settings.  See the grouper-loader.base.properties for the possible
# settings that can be applied to the grouper.properties
```

2. Section header starts with a comment with at least 10 hashes, has documentation that starts with two hashes, and ends with a line with at least 10 hashes. Note the first line is the header in the config UI, and the next lines are the description (more info).

```
########################################
## Config chaining hierarchy
## configure how configs are read and in what order
########################################
```

3. Blank line separates items and sections
4. Config file comment and item has documentation. Comments start with a single hash. Comments can contain json metadata. Metadata can span several comment lines. Item keys and values are separated by an equals and optional whitespace. Metadata must follow comments, not the other way around. After the comment or metadata is the property or whitespace

```
# seconds between checking to see if the config files are updated
# {valueType: "integer", required: true}
grouper.config.secondsBetweenUpdateChecks = 60
```

5. grouper.properties config for DB config (across all configs)

```
# seconds between checking to see if the config files are updated in the database.  If anything edited,
then refresh all.
# Note that the last edited is stored in a config property for deletes.  -1 means dont check for
incrementals.
# Note if *.config.secondsBetweenUpdateChecks is greater than this number
# for this config, then it wont update until that amount has passed.
# {valueType: "integer", required: true}
grouper.config.secondsBetweenUpdateChecksToDb = 60

# seconds between full refreshes of the database config
# {valueType: "integer", required: true}
grouper.config.secondsBetweenFullRefresh = 3600

# millis since that config inserted, edited or deleted.  Grouper will update this config.  Do not edit it
# Note change this with jdbc and dont audit.
# if grouper.config.secondsBetweenUpdateChecks is -1, then dont use this property
# {valueType: "integer", required: true}
grouper.config.millisSinceLastDbConfigChanged = 0
```

6. You could comment out one sample value after the metadata which will be used an a sample value. A sample value shows up on the screen but has no value

```
#auto-create groups (increment the integer index), and auto-populate with users
#(comma separated subject ids) to bootstrap the registry on startup
#(note: check config needs to be on)
# {regex: "configuration.autocreate.group.name.[0-9]+", valueType: "group", required: true}
#configuration.autocreate.group.name.0 = $$grouper.rootStemForBuiltinObjects$$:uiUsers
```

7. df

Technical design of retrieving the configuration from the database

Log with this in log4j.properties

```
log4j.logger.edu.internet2.middleware.grouperClient.config.db.ConfigDatabaseLogic = DEBUG
```

In the grouper client (since that is where the hierarchical config code is), have the logic to retrieve the configuration from the database.

It should use pooling so that it is efficient.

If should **NOT** use anything from the grouper API or anything that uses grouper client config (basically dont use anything except maybe Morph encryption in the grouperClient, and external libraries e.g. c3p0 pooling). This is because the database framework in the API uses configuration. So the configuration cannot use the API or there is a circular logic problem in looping and bootstrapping.

There are some configs for this, but these are set after the first grouper.properties is retrieved. These are not in the grouper-hibernate.properties so they can be edited at runtime (grouper-hibernate.properties is not stored in the database of course).

Algorithm

1. A configuration is retrieved from the API (from any of the config files)

```
e.g.GrouperConfig.retrieveConfig().propertyValueString("grouper.rootStemForBuiltinObjects", "etc")
```

2. If it has not been longer than the *.config.secondsBetweenUpdateChecks for that config (e.g. grouper.config.secondsBetweenUpdateChecks), then just return the cached (in memory) config
3. If it has been longer, then see if the last full refresh has been longer than grouper.config.secondsBetweenFullRefresh. If so, then do a full refresh of all configs in DB
4. If it has been longer, then see if has not been longer than grouper.config.secondsBetweenUpdateChecksToDb. If it has not, then get the DB config for that config file from memory cache

5. If it has been longer, then query the millis since last refresh (from config table, get this value): grouper.config.millisSinceLastDbConfigChanged If the last refresh is before that value, then do a full refresh
6. Note, Grouper can clear the cache when any property (besides grouper.config.millisSinceLastDbConfigChanged) is changed
7. Note: grouper.config.millisSinceLastDbConfigChanged is updated by grouper (and not audited), when there is any insert/update/delete to config