API Building & Configuration

Building the Grouper API

The Grouper API is provided both as a binary and source distribution. Note the Grouper Installer will install the binary grouper API.

To build the source distribution :

```
cd grouper-api-1.5.0
ant dist
```

Testing the API is performed using GrouperShell :

> Testing will destroy any pre-existing data in the Groups Registry database

```
bin/gsh.sh -test -all
```

Configuring the Grouper API

This section describes all of the Grouper API configuration files and important settings.

> Starting with Grouper 2.2 it is highly recommended that you restructure your configuration files to use Configuration Overlays.

The Grouper API is distributed with example configuration files with ".example" inserted in the middle of their names. These should be renamed or copied to remove the ".example" substring, or doing a build with ant will do this, or it is already copied in the binary distribution. e.g. for grouper.properties, the example file is grouper.example.properties.

Section	Configuration File	Purpose
Database-Related Settings and Procedures	grouper.hibernate.properties	integrating the Grouper API with the database that will house your Groups Registry
Configuration of Source Adapters	sources.xml	integrating the Grouper API with chosen identity sources

Grouper properties	grouper.properties	defaults for Grouper privileges, enabling identified external users to act with elevated root-like privilege, changing the display name for internal subjects
Logging	log4j2.properties	logging
Daemon	grouper-loader.properties	auto-load memberships from external sql sources, register notification consumers, validate Grouper Rules, update enabled/disabled flags, etc

Database-Related Settings and Procedures

Database Driver Location

Place the jar file containing the JDBC driver for your database in the lib/custom/ directory. The Grouper v1.5.0 package includes the JDBC driver for HSQLDB v1.8.0.10. Sample JDBC drivers are located in lib/jdbcSample (e.g. for Oracle, MySQL, and PostgreSQL).

General Property Settings

Grouper uses Hibernate to persist objects in the Groups Registry. Database-specific settings are configured in conf/grouper.hibernate.properties, which has pre-populated examples for HSQLDB, MySQL, Oracle, and Postgresql.

Required properties are:

Property Name	Purpose
hibernate.connection.driver_class	JDBC driver classname
hibernate.connection.url	JDBC URL for the database
hibernate.connection.username	database user
hibernate.connection.password	database user's password Note, you can also put a filename of the encrypted password
hibernate.dialect	classname of a Hibernate dialect, for setting platform specific features. Choices are listed (Hibernate Reference Documentation - Chapter 3. Configuration - 3.3. JDBC connections - 3.4.1. SQL Dialects) here

You may need to refer to your database support person to determine these required properties.

Detailed Hibernate configuration documentation is available (Hibernate Reference Documentation - Chapter 3. Configuration - 3.3. JDBC connections) here.

MySQL Transaction Support

If you want transactions to work (i.e. when doing a unit of work in grouper, it either all completes or none), which is definitely recommended, though not required, your mysql table format needs to be transactional, e.g. innodb, which is not the default (myisam is the default). One way to enable innodb in mysql is with this line in the my.cnf: default-storage-engine=innodb

MySQL/MariaDB Character Set and Performance Settings

For MySQL and MariaDB, the following settings are recommended.

```
character-set-server = utf8
collation-server = utf8_bin
innodb_buffer_pool_size = 4G
```

utf8mb4 is not recommended because the database engines in MySQL/MariaDB that support transactions have limitations on the length of prefixes for indices. For InnoDB and XtraDB, the prefix is limited to 767 bytes. When using utf8mb4, where 4 bytes are used instead of 2, the prefixes used for some Grouper tables are too long.

You can also consider setting innodb_flush_log_at_trx_commit = 2. The default setting of 1 is required for full ACID compliance and logs are written and flushed to disk at each transaction commit. However, this can be expensive in terms of disk I/O. With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash. The tradeoff here may be acceptable in your environment.

For those running MariaDB you should read this knowledge base article about OPTIMIZE and defragmenting. Some have found MariaDB 10.3+ to be a good, fast variant of MySQL for use by Grouper.

Database Allow changes and Deny

Some database operations (such as dropping tables or recreating data during tests) require confirmation of a prompt asking whether or not to continue. It is possible to automatically allow or deny these database operations in conf/grouper.properties :

```
# allow and deny for db data or object deletes.
# if a listing is in the allow, it will be allowed to delete db
# if a listing is in the deny, it will be denied from deleting db
# multiple inputs can be entered with .0, .1, .2, etc. These numbers must be sequential, starting with 0
db.change.allow.user.0=grouper3
db.change.allow.url.0=jdbc:mysql://localhost:3306/grouper3?useSSL=false
db.change.allow.user.1=grouper1
db.change.allow.url.1=jdbc:mysql://localhost:3306/grouper1?useSSL=false

db.change.deny.user.0=grouper2
db.change.deny.url.0=jdbc:mysql://localhost:3306/grouper2?useSSL=false
```

Database Initialization Procedure

Database initialization is performed using the GrouperShell.

> Initializing the database will destory any pre-existing data

To initialize the Groups Registry and install tables, populate default group types and fields, and create the root naming stem :

```
bin/gsh.sh -registry -check -runscript
```

To re-initialize the Groups Registry (e.g. after running junit tests) :

```
bin/gsh.sh -registry -reset
```

To see all options :

```
bin/gsh.sh -registry
```

Database Tuning

Analyzing Tables to Improve Query Performance

Whenever a lot of changes are made to the data in the Groups Registry database (including upgrades of Grouper), you should analyze your database tables to improve query performance.

MySQL Syntax: ANALYZE TABLE table_name
http://dev.mysql.com/doc/refman/5.5/en/analyze-table.html

PostgreSQL Syntax: ANALYZE table_name
http://www.postgresql.org/docs/8.4/static/sql-analyze.html

Oracle Syntax: exec dbms_stats.gather_table_stats('schema', 'table_name', cascade => TRUE);
Or this: EXEC DBMS_STATS.gather_schema_stats('schema');
http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14258/d_stats.htm

Note, you can analyze a subset of the rows in oracle if you have a lot of data

```
select 'ANALYZE TABLE ' || table_name || ' estimate STATISTICS sample 100000 rows;' as script from user_Tables
where table_name like 'GROUPER%'

e.g. ANALYZE TABLE GROUPER_ATTRIBUTES estimate STATISTICS sample 100000 rows;
or, e.g. ANALYZE TABLE GROUPER_ATTRIBUTES compute STATISTICS;
```

In all cases, substitute "table_name" with each table that you want to have analyzed. For Oracle, also substitue "schema" with the database schema for your Groups Registry.

MySQL example...

```
ANALYZE TABLE grouper_groups;
ANALYZE TABLE grouper_stems;
ANALYZE TABLE grouper_memberships;
ANALYZE TABLE grouper_group_set;
....
```

Improving queries using histogram statistics

Even with a full set of statistics on tables, columns, and indexes, this is sometimes not enough information for some queries. For example, in a database with 100,000 groups and 100,000 users, a query plan based on memberships may think that there will likely be at most one group per member. So the query plan may be built on the assumption that it can safely do a Nested Loop iteration through the few rows returned. But it is a plausible example that the GrouperAll subject is granted read access to a large number of these groups. This could have an effect on queries for non-wheel users when checking whether the logged in user can read a group. Instead of looping through a few rows, it could be looping through thousands.

With database histograms, values are put into a fixed number of bins. If the column data is heavily skewed toward one value, that value will occupy one or more bins by itself, and the query analysis can use that information to get a rough estimate on the cardinality of a filter on that column.

With Oracle, a first step toward improving these queries is to add a histogram for a single column, e.g.:

```
BEGIN
DBMS_STATS.GATHER_TABLE_STATS (
ownname => 'GROUPER'
, tabname => 'GROUPER_MEMBERSHIPS'
, method_opt => 'FOR COLUMNS MEMBER_ID'
);
END;
```

Histograms on more than one column require an extended version of this.

```
select dbms_stats.create_extended_stats(null, 'GROUPER_GROUP_SET', '(OWNER_GROUP_ID, FIELD_ID)') from dual;
-- (will return a generated rowid such as SYS_STU$V77C8_NRNA1MJMG#1SPOH$)

exec dbms_stats.gather_table_stats(null, 'GROUPER_GROUP_SET');

select * from user_tab_col_statistics where table_name = 'GROUPER_GROUP_SET';
```

MySQL starting from version 8 has histograms, probably similar to Oracle. https://mysqlserverteam.com/histogram-statistics-in-mysql/ .

Configuration of Source Adapters

Grouper uses Subject API compliant "source adapters" to integrate with external identity stores. "Subjects" are the objects housed there that are presented to Grouper for management vis-à-vis group membership and Grouper privileges. These may represent people, other groups, computers, applications, services, most anything for which you manage identity. With the exception of Grouper groups, Grouper treats all subjects opaquely. See the Subject API documentation for further background and details concerning subjects, source adapters, and other aspects of the Subject API.

Each source adapter connects with a single back-end store using JDBC or JNDI. Grouper makes no specific assumptions about the schema of any subject types. Instead, sections of the configuration file, grouper/conf/sources.xml, declare the details of how to connect with each back-end store, the identifier(s) to be used for the subjects it contains, how to select and search for subjects, and which subject attributes should be made available to Grouper.

Three types of source adapters are included in the Grouper API v1.5.0 package. JDBCSourceAdapter and JNDISourceAdapter classes are included in subject.jar, and GrouperSourceAdapter is built along with the Grouper API. Every Grouper API deployment MUST include a *source* element in grouper /conf/sources.xml for the GrouperSourceAdapter so that Grouper can refer to its own groups in the same manner as other subjects.

JDBC and JNDI sources have two options each. For JDBC, if you can make a table/view where each subject is represented as one row of the view, then the more powerful GrouperJdbcSourceAdapter2. One of the major advantages is that if you enter in a phrase in the subject search, e.g. "John Smith", then it will search for records which have John and Smith in them (case insensitive), whereas the GrouperJdbcSourceAdapter will look for the whole string "John Smith" and will not return a record for "John L Smith". For JNDI, UW contributed a source adapter which should give better performance.

As of Grouper 2.0, Grouper stores additional data about subjects that are used by Grouper to search and sort a list of members. Each source must be configured for at least one search attribute and one sort attribute.

See the sources.example.xml for example usages of the sources.xml

Note that in 1.5.0 the subject API changed, so if you have custom subject sources you will need to tweak and recompile them.

Choosing Identifiers for Subjects

Identifiers and their management can get complicated. They can be revoked or not, re-assigned or not, lucent or opaque, etc. Depending on such characteristics, a given identifier might be a good or bad choice to use in the context of managing the identified subject's group memberships.

For example, a username is often lucent - easily remembered by the person to whom it is associated. But it may also be revokable, meaning that it no longer refers to that person (perhaps they have a new one), or even re-assignable, meaning that it might refer to some other person at a later time. If a username is used to record membership, username changes must trigger corresponding membership changes. A username is better suited to authentication than it is to indicating membership.

On the other hand, an opaque registryID (machine, not human, readable) that never changes is great for membership, but lousy for authentication - it might not even be known by the person to whom it is associated. How would I identify myself to Grouper if I wished to opt-in to a list or manage a group?

Grouper accommodates subject identifier issues in two ways. First, it maintains UUIDs for every subject and group within the Groups Registry. These are never exposed by the API, but are associated with externally supplied subject identifiers within the Groups Registry. This approach allows the identifier associated with a given subject to be changed without any need to change actual memberships.

Second, by relying on the Subject API, Grouper is able to lookup subjects that are presented with an identifier in one namespace and obtain identifiers in other namespaces for that subject. That means that it can translate a username into a registryID, for example. So, when a user authenticates to an application using the Grouper API, that application can use the Subject API to fetch an identifier for the person chosen by the site for use in memberships. Similarly, when a membership in the Groups Registry is to be expressed elsewhere, the identifier used for group members can be translated by a provisioning connector by use of the Subject API into one that is suitable in the provisioned context.

Subject ID: should be unchangeable, unrevokable. Usually this an opaque id (number or uuid etc). The source that a subject is associated with also should not change.

Subject Identifier: anything that can refer to a subject uniquely. Usually these are netIds, eppns, etc.

It would be nice if subject id's and identifiers are unique across sources, though this is not required.

You should not have the same subject in more than one source.

Subjects should be resolvable for as long as you want users to be able to search for them or view them on the UI. It is possible for subjects to not be active in which case they are not searchable, but still be resolvable so they can be shown in the UI in auditing.

Grouper Properties

All configuration of Grouper properties detailed in this section occur in the grouper/conf/grouper.properties file. Look in the grouper.example.properties file for the more obscure settings. Common settings are listed below.

Note that in Grouper 2.2 and above, overlay Configuration files can be used (and are recommended).

This setting describes the env that grouper is running, e.g. used in the daily report from the loader which

```
grouper.env.name = production
```

If Grouper should auto init the registry if not initted (i.e. insert the root stem, built in fields, etc)

```
registry.autoinit = true
```

If Grouper should try and detect and log configuration errors on startup, in general this should be true, unless the output is too annoying or if it is causing a problem

```
configuration.detect.errors = true
```

If groups like the wheel group should be auto-created for convenience (note: check config needs to be on)

```
configuration.autocreate.system.groups = false
```

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). The next group would end in 1, then 2, etc

```
configuration.autocreate.group.name.0 = etc:uiUsers
configuration.autocreate.group.description.0 = users allowed to log in to the UI
configuration.autocreate.group.subjects.0 = johnsmith
```

By default, anyone with admin rights on a group can edit the types or attributes. Specify types (and related attributes) which are wheel only, or restricted to a certain group

```
security.types.typeName.wheelOnly = true
security.types.grouperLoader.wheelOnly = true

#security.types.typeName.allowOnlyGroup = etc:someAdminGroup
```

If you don't want to be prompted for DDL changes in certain databases (e.g. dev), list them here:
Allow and deny for db data or object deletes, without prompting the user to confirm
If a listing is in the allow, it will be allowed to delete db
If a listing is in the deny, it will be denied from deleting db
Multiple inputs can be entered with .0, .1, .2, etc. These numbers must be sequential, starting with 0

```
db.change.allow.user.0=grouper3
db.change.allow.url.0=jdbc:mysql://localhost:3306/grouper3?useSSL=false
db.change.allow.user.1=grouper1
db.change.allow.url.1=jdbc:mysql://localhost:3306/grouper1?useSSL=false

db.change.deny.user.0=grouper2
db.change.deny.url.0=jdbc:mysql://localhost:3306/grouper2?useSSL=false
```

There is a substantial section for include/exclude and requireGroups. These are group types which help you create composite groups to manage include /exclude lists for groups (especially useful for grouper loader privisioned groups), or groups which require memberships in other groups (e.g. activeStaff). See the grouper.example.properties file if you want to customize things, but to enable, set these:

```
grouperIncludeExclude.use = false
grouperIncludeExclude.requireGroups.use = false
```

Here are some requireGroups (increment the 0 to add more):

```
#grouperIncludeExclude.requireGroup.name.0 = requireActiveStudent
#grouperIncludeExclude.requireGroup.attributeOrType.0 = attribute
#grouperIncludeExclude.requireGroup.group.0 = school:community:activeStudent
#grouperIncludeExclude.requireGroup.description.0 = If value is true, members of the overall group must be an
active student (in the school:community:activeStudent group). Otherwise leave this value not filled in.
```

Hooks are ways to plugin in your own java code to affect how Grouper does its logic. You can register multiple classes for one hook base class by comma separating the hooks implementations. You can also register hooks at runtime with: GrouperHookType.addHookManual("hooks.group.class", YourSchoolGroupHooks2.class);
See the grouper.example.properties for the full list, here are two examples:

```
#implement a group attribute hook by extending edu.internet2.middleware.grouper.hooks.AttributeHooks
#hooks.attribute.class=edu.yourSchool.it.YourSchoolGroupHooks,edu.yourSchool.it.YourSchoolGroupHooks2

#implement a group hook by extending edu.internet2.middleware.grouper.hooks.GroupHooks
#hooks.group.class=edu.yourSchool.it.YourSchoolGroupHooks,edu.yourSchool.it.YourSchoolGroupHooks2
```

You can validate group attributes via regex (see grouper.example.properties for more info) (increment the 0 to add more)

```
#group.attribute.validator.attributeName.0=extension
#group.attribute.validator.regex.0=^[a-zA-Z0-9]+$
#group.attribute.validator.vetoMessage.0=Group ID '$attributeValue$' is invalid since it must contain only
alpha-numerics
```

Database structure data definition language (DDL) settings (see grouper.example.properties for full list)

```
# if you want to not create the subject tables (grouper examples for unit testing),
# then set this to true
ddlutils.exclude.subject.tables = false

# set the path where ddl scripts are generated (they will be uniquely named in this directory).
# if blank, the directory used will be the current directory
ddlutils.directory.for.scripts = ddlScripts

# during schema export, should it install grouper data also or not. e.g. insert the root stem, default true
ddlutils.schemaexport.installGrouperData = true

# when grouper starts, should it shut down if not right version?
ddlutils.failIfNotRightVersion = true

# after you have converted id's, and are happy with the conversion of removing the uuid col,
# this will remove the backup uuid cols when running the gsh command: registryInitializeSchema()
ddlutils.dropBackupUuidCols = false

# after you have converted field id foreign keys, and are happy with the conversion of removing the attribute
name,
# membership list name, and type cols,
# this will remove the backup field name/type cols when running the gsh command: registryInitializeSchema()
ddlutils.dropBackupFieldNameTypeCols = false

# before the group name etc was moved to the grouper_groups table, the attributes table
# was backed up.  If it should not be backed up, or if the upgrade is done and works, then it can
# be removed, set to true, run: gsh -registry -deep
ddlutils.dropAttributeBackupTableFromGroupUpgrade = false

# Since grouper_memberships no longer has effective memberships, that table doesn't need via_id,
# depth and parent_membership.  If they were converted, this will drop the backup of those cols with: gsh -
registry -deep
ddlutils.dropMembershipBackupColsFromOwnerViaUpgrade = false

# this is the schema ddlutils uses to query metadata with jdbc. usually this can be omitted,
# and it defaults to your database loginid, however, in postgres, it can be different, so enter here
#ddlutils.schema = public


#if you are running a DB that supports them, but you dont want them, disable object comments here (defaults to
false)
ddlutils.disableComments = false

#set to true and we wont subsitute varchar 4000 for text in mysql (wont work in innodb utf-8 databases
ddlutils.dontSubstituteVarchar4000forTextMysql = false
```

Mail settings (optional, e.g. for daily report from the loader)

```
#smtp server is a domain name or dns name, must be simple clear text stmp with no authentication
#mail.smtp.server = whatever.school.edu

#leave blank if unauthenticated
#mail.smtp.user =

#leave blank if unauthenticated
#mail.smtp.pass =

#this is the default email address where mail from grouper will come from
#mail.from.address = noreply@school.edu

#this is the subject prefix of emails, which will help differentiate prod vs test vs dev etc
#mail.subject.prefix = TEST:
```

Default privileges

Grouper requires that all subjects must be explicitly granted access or naming privileges (cf. Grouper glossary), with one caveat. There is a special "subject" internal to Grouper called the ALL subject, which is a stand-in for any subject. The ALL subject can be granted a privilege in lieu of assigning that privilege explicitly to each and every subject.

When a new group or naming stem is created, any of its associated privileges can be granted by default to the ALL subject. This is configured by a series of properties in grouper.properties, one per privilege. If a property has the value "true" then ALL is granted that privilege by default when a group or naming stem is created. Otherwise it is not, and hence no subject has that privilege by default. The groups read and view settings below are set to true to make the quickstart easier to run. If you have an deployment where privacy among Grouper users is important, you should consider changing those to false so that access to see or view memberships of groups must be explicitly assigned.

Property Name	Value in Grouper v1.5 Distribution
groups.create.grant.all.admin	false
groups.create.grant.all.optin	false
groups.create.grant.all.optout	false
groups.create.grant.all.update	false
groups.create.grant.all.read	true
groups.create.grant.all.view	true
stems.create.grant.all.create	false
stems.create.grant.all.stem	false
attributeDefs.create.grant.all.attrAdmin	false
attributeDefs.create.grant.all.attrOptin	false
attributeDefs.create.grant.all.attrOptout	false
attributeDefs.create.grant.all.attrRead	false
attributeDefs.create.grant.all.attrUpdate	false
attributeDefs.create.grant.all.attrView	false

Super-user Privileges

Grouper has another special "subject" called GrouperSysAdmin that acts as a super-user. GrouperSysAdmin is permitted to do everything - the privilege system is ignored for that special subject. Grouper can be configured to consider all members of a distinguished group to be able to act as super-users, much as the "wheel" group does in BSD Unix. Two properties control this behavior:

Property Name	Description
groups.wheel.use	"true" or "false" to enable or disable this capability.
groups.wheel.group	The group name of the group whose members are to be considered security-equivalent to GrouperSysAdmin. By default this is: etc:sysadmingroup

The Grouper UI enables users that belong to the wheel group to choose when to act with the privileges of GrouperSystem and when to act as their normal selves.

Changing the display name of GrouperAll and GrouperSystem

Before version 1.3.0 the Grouper UI referred to *EveryEntity* as GrouperAll and *GrouperSysAdmin as* GrouperSystem. As of version 1.3.1 the name attribute of GrouperAll and GrouperSystem can be set through the properties below.

Property Name	Description
subject.internal.grouperall.name	The name to use for GrouperAll instead of EveryEntity
subject.internal.groupersystem.name	The name to use for GrouperSystem instead of GrouperSysAdmin

If you choose not to use the defaults you will have to update the UI nav.properties file to ensure consistency e.g. subject.privileges.from-grouperall=inherits from EveryEntity

Changing default privilege caching

Grouper includes three `PrivilegeCache` implementations:

- NoCachePrivilegeCache - No caching performed
- SimplePrivilegeCache - Caches results but flushes all cached entries upon any update
- SimpleWheelPrivilegeCache - Same as 'SimplePrivilegeCache' but with better support for using a wheel group.

The privileges.access.cache.interface and privileges.naming.cache.interface properties can be set to determine the privilege caching regimen. The default is edu.internet2.middleware.grouper.NoCachePrivilegeCache.

Using a privilege management system external to Grouper

Grouper's internal security implementation relies on two java interfaces, one for Naming Privileges and another for Access Privileges. Grouper ships with classes that implement these interfaces, but 3rd parties are free to supply their own and so manage Grouper privileges using a privilege management system external to Grouper. Two properties declare the java classes that Grouper will use to implement these interfaces:

Property Name	Description
privileges.access.interface	classname of the java class that implements the Access Interface
privileges.naming.interface	classname of the java class that implements the Naming Interface
privileges.attributeDef.interface	classname of the java class that implements the Attribute access Interface

It is not clear that this has been taken advantage of... the internal privilege management is the one usually used. Also, performance of the system will be drastically reduced if external privileges are used, since internal privilege management can join tables in one query to securely select from the registry. If you want to store privileges externally, another option is provisioning the internal access adapter settings and table data from an outside system.

Notifications / change log

To enable the change log, set this:

Property Name	Default Value	Description
changeLog.enabled	true	if we should insert records into grouper_change_log_temp when events happen

If you are using ldappc, then you need to keep updating the last membership time. If not (and not using this column for other custom reasons), you will reduce the number of queries by setting this to false.

Property Name	Default Value	Description
groups.updateLastMembershipTime	true	If true, when a membership is added to a group (either a privilege or a list member), then an update will be made to the lastMembershipChange property for the group.
stems.updateLastMembershipTime	true	If true, when a membership is added to a stem (this would be a naming privilege), then an update will be made to the lastMembershipChange property for the stem.

Logging

Logging is configured in the grouper/conf/log4j.properties configuration file. By default Grouper will write event log information to grouper/grouper_event.log, error logging to grouper/grouper-error.log, and debug logging, if enabled, to grouper/grouper-debug.log. The log4j configuration can be adjusted to control the verbosity, type and output of Grouper's logging.

Daemon

The Grouper - Daemon is a daemon command line process which handles many tasks including running jobs that load/remove memberships of groups based on results from a sql query, executing change log consumers, validating Grouper Rules, and updating enabled/disabled flags. There is a grouper-loader.properties file to configure. The common settings are described below, see the grouper-loader.example.properties for descriptions of all settings.

If you want Grouper to make sure the loader type and attributes exist if not there, set this. Otherwise you need to add the type and attributes yourself with GSH.

```
# auto-add grouper loader types and attributes when grouper starts up if they are not there
loader.autoadd.typesAttributes = false
```

If most of your loader queries come from one subject source, you can set the default subject source here, so your loader queries only need to return SUBJECT_ID and not the source id also:

```
default.subject.source.id =
```

If all of your queries are run against your grouper db credentials (e.g. if you have other schemas on the same DB to query), you dont have to configure DB connections. If you have other databases to query (e.g. an external data warehouse, etc), you can configure the db credentials here. The name here is "warehouse", use different names for different connections

```
db.warehouse.user = mylogin
#note the password can be stored encrypted in an external file
db.warehouse.pass = secret
db.warehouse.url = jdbc:mysql://localhost:3306/grouper?useSSL=false
db.warehouse.driver = com.mysql.jdbc.Driver
```

If you want to use the Grouper daily report, configure in the grouper-loader.properties (and the mail settings described above in grouper.properties). This is a daily email that gets sent to you about your grouper health, including status about all the loader jobs in the last day.

```
#quartz cron-like schedule for daily grouper report, the default is 7am every day: 0 0 7 * * ?
#leave blank to disable this
daily.report.quartz.cron =

#comma separated email addresses to email the daily report, e.g. a@b.c, b@c.d
daily.report.emailTo =
```

Schedule when to run the enabled/disabled cron.

```
###################################
## enabled / disabled cron
###################################

#quartz cron-like schedule for enabled/disabled daemon.  Note, this has nothing to do with the changelog
#leave blank to disable this, the default is 12:01am, 11:01am, 3:01pm every day: 0 1 0,11,15 * * ?
changeLog.enabledDisabled.quartz.cron = 0 1 0,11,15 * * ?
```

Manage change log consumers and also specify whether some events are written to the change log.

```
###################################
## Change log
###################################

# should the change log temp to change log daemon run?  Note, this should be true
changeLog.changeLogTempToChangeLog.enable = true

#quartz cron-like schedule for change log temp to change log daemon, the default is 50 seconds after every
minute: 50 * * * * ?
#leave blank to disable this
changeLog.changeLogTempToChangeLog.quartz.cron =

# Should the change log include flattened memberships?
changeLog.includeFlattenedMemberships = true

# Should the change log include flattened privileges?
changeLog.includeFlattenedPrivileges = true

# Should the change log include flattened permissions?
changeLog.includeFlattenedPermissions = true

# Should the change log include non-flattened (immediate and composite only) memberships?
changeLog.includeNonFlattenedMemberships = false

# Should the change log include non-flattened (immediate only) privileges?
changeLog.includeNonFlattenedPrivileges = false

#changeLog.consumer.printTest.class = edu.internet2.middleware.grouper.changeLog.consumer.PrintTest
#changeLog.consumer.printTest.quartzCron =

#rules consumer, needed for some of the Grouper rule types to run (e.g. flattenedMembershipRemove,
flattenedMembershipAdd)
changeLog.consumer.grouperRules.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.RuleConsumer
changeLog.consumer.grouperRules.quartzCron =

#consumer for syncing groups to other groupers
changeLog.consumer.syncGroups.class = edu.internet2.middleware.grouper.client.GroupSyncConsumer
changeLog.consumer.syncGroups.quartzCron =
```

XMPP notifications

```
###################################
## XMPP notifications
## (note, uncomment the consumer class and cron above)
## this will get grouper ws getMembers rest lite xmp:
## http://anonsvn.internet2.edu/cgi-bin/viewvc.cgi/i2mi/trunk/grouper-ws/grouper-ws/doc/samples/getMembers
/WsSampleGetMembersRestLite_xml.txt?view=log
###################################

## general xmpp configuration
xmpp.server.host = jabber.school.edu
xmpp.server.port = 5222
xmpp.user = username
# note, pass can be in an external file with morphstring
xmpp.pass =
xmpp.resource = grouperServer
```

Schedule when to run daemon to validate rules.

```
###################################
## Rules config
###################################

# when the rules validations and daemons run.  Leave blank to not run
rules.quartz.cron = 0 0 7 * * ?
```

ESB Integration

```
####################################
## ESB integration
####################################

#changeLog.consumer.xmppTest.quartzCron =
#changeLog.consumer.xmppTest.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.EsbConsumer
#changeLog.consumer.xmppTest.elfilter = event.eventType eq 'GROUP_DELETE' || event.eventType eq 'GROUP_ADD' ||
event.eventType eq 'MEMBERSHIP_DELETE' || event.eventType eq 'MEMBERSHIP_ADD'
#changeLog.consumer.xmppTest.publisher.class = edu.internet2.middleware.grouper.changeLog.esb.consumer.
EsbXmppPublisher
#changeLog.consumer.xmppTest.publisher.server = jabber.school.edu
#changeLog.consumer.xmppTest.publisher.port = 5222
#changeLog.consumer.xmppTest.publisher.username = jabberuser
#changeLog.consumer.xmppTest.publisher.password = /home/whatever/pass/jabberuserEncrypted.pass
#changeLog.consumer.xmppTest.publisher.recipient = system1@school.edu
#changeLog.consumer.xmppTest.publisher.addSubjectAttributes = NETID
```

See Also

Subject API

Member Search and Sort

Configuration Overlays