Legacy attribute migration

Overview: The Grouper 2.2 upgrade migrates the legacy attributes and group types to the attribute framework in a transparent way. The old API and WS and UI should still work correctly.

Pre-2.2 functionality

Group types:

- Grouper allows administrators to create group types (typically using GSH or the API)
- There are several internal/built-in group types -- base, naming, attributeDef. There's also requireInGroups, addIncludeExclude, and grouperLoader.
- Group types can have one more more attributes and one or more custom lists.
- There are change log events when group types are created, updated, or deleted. The change log processor doesn't do anything with them.
- Special group type called grouperGroupMembershipSettings that's used to help users skin the lite membership UI.
- Hooks
- User audit

Group type assignments (GroupTypeTuple):

- In general, all users can assign a group type to groups that they admin.
- There are built-in hooks to limit who can assign a specific group type or edit attributes of a group associated with that group type.
- When assigning a group type to a group, a bundle of attributes and lists become available for that group. For example with the grouperLoader type.
- There are change log events when group types are assigned or unassigned. The change log processor does look at these events.
- Hooks
- User audit

Fields:

- Attributes and custom lists are fields. (Just like the fields used for group memberships and group/stem/attributeDef privileges.)
- Fields are associated with a group type.
- Fields have properties for read and write privilege.
  - e.g. You must have read (or opt-in or update or admin, etc) privilege on a group to read an attribute value on a group.
  - e.g. You must have update (or read or opt-in or admin, etc) privilege on a group to update/delete an attribute value on a group.
  - Same with addMember, deleteMember, and getMembers on custom lists. You can have crazy settings like requiring opt-out privilege on a group to be able to add member to a custom list.
- Fields have a nullable property. This prevents deleting an attribute from a group if failOnRequiredAttribute is true in the API call and the nullable property is true on the field.
- Change log
- Hooks
- PIT audit

Attributes:

- Stored in the grouper_attributes table. Has a reference to the group and field.
- Single valued only.
- Can be copied when copying groups.
- Hooks
- User audit

Custom lists:

- Stored in the grouper_memberships table like regular memberships and privileges. Again has a reference to the group and field.
- Can be copied when copying groups.
- Change log
- Hooks
- User audit
- PIT audit

Requirements for migration

- Prevent use of legacy attributes.
- Create old style attributes in the new attribute framework within a configurable folder (e.g. etc:legacy:attribute). By default, everybody can read and assign the attribute.
- Preserve the API methods that allow setting and getting an attribute from a group.
- Web services and UI will continue to work when dealing with getting and setting attributes.
- Hooks will continue to fire when setting and deleting attributes.
- Existing views on the legacy attributes will not be preserved.

Details on Migration

- This is done during update to Grouper 2.2.
- Take note about the section at the bottom regarding privileges.
- During the upgrade, the following tables are backed up to the following tables. The backed up tables are used by the migration script to do the migration.
    - grouper_attributes -> grouper_attributes_legacy (grouper_attributes will be dropped during the upgrade)
    - grouper_types -> grouper_types_legacy (grouper_types will be dropped during the upgrade)
    - grouper_groups_types -> grouper_groups_types_legacy (grouper_groups_types will be dropped during the upgrade)
    - grouper_fields -> grouper_fields_legacy (grouper_fields is still used in Grouper 2.2 but is just no longer used to identify attributes. We're also dropping the columns: grouptype_uuid and is_nullable.)
- For each groupType:
    - Create an attributeDef named etc:legacy:attribute:legacyGroupTypeDef_<groupTypeName>. This is assignable to groups.
    - Create an attributeDefName named etc:legacy:attribute:legacyGroupType_<groupTypeName>. This represents the groupType.
    - Give everyone (EveryEntity) ATTR_READ and ATTR_UPDATE privileges. This allows everyone to be able to assign this group type to their groups, which is the functionality that existed with the old groupTypes. Note that the hooks that limit this functionality still apply.
- For each legacy attribute:
    - (Just once per groupType) Create an attributeDef named etc:legacy:attribute:legacyAttributeDef_<groupTypeName>. This is assignable to groupType assignments on groups.
    - Create an attributeDefName named etc:legacy:attribute:legacyAttribute_<legacyAttributeName>. This represents the legacy attribute.
    - Give everyone (EveryEntity) ATTR_READ and ATTR_UPDATE privileges. This allows everyone to be able to read this attribute when assigned to groups (assuming they have GROUP_ATTR_READ or ADMIN on the group) and update this attribute on groups (assuming they have GROUP_ATTR_UPDATE or ADMIN on the group).
- For each custom list:
    - (Just once per groupType) Create an attributeDef named etc:legacy:attribute:legacyCustomListDef_<groupTypeName>. This is assignable to the groupType attribute definition.
    - (Just once per groupType) Create an attributeDefName named etc:legacy:attribute:legacyCustomList_<groupTypeName>
    - This attribute will be assigned to the group type definition to indicate which custom fields are associated with that group type. So it is multi-valued, with the values being the field UUIDs.
- For each groupType assignment:
    - Assign the groupType (etc:legacy:attribute:legacyGroupType_<groupTypeName>) to the group.
- For each legacy attribute assignment:
    - Assign the attribute (etc:legacy:attribute:legacyAttribute_<legacyAttributeName>) to the groupType assignment. And add the value.

Design and API Changes

GroupType

- Get rid of grouper_types table. The default, base, naming, and attributeDef types will no longer be represented in the API. So if you query for those types, you'll get an exception.

GroupType.createType(session, name)
 - deprecated
GroupType.createType(session, name, exceptionIfExists)
 - deprecated
GroupType.internal_createType(session, name, isAssignable, isInternal, exceptionIfExists, changed, uuid)
 - rewrite
 - Remove isAssignable and isInternal
 - Create new attributeDef and attributeDefName in etc:legacy:attribute to represent the groupType.
   - The attributeDef is assignable to groups. Calling it etc:legacy:attribute:legacyGroupTypeDef_<name>.
   - The attributeDefName is a marker attribute for the group type. Call it etc:legacy:attribute:legacyGroupType_<name>.
 - The groupType uuid should become the uuid of the attributeDefName legacyGroupType_<name>.
GroupType.addAttribute(session, name, read, write, required)
 - deprecate
GroupType.addAttribute(session, name, read, write, required, exceptionIfExists)
 - deprecate
 - rewrite - the current code creates a field for this, but we're not doing that.
 - Ignore required.
 - Ignore read and write privileges. Privileges should be set using the new attribute framework.
 - Create attributeDef for attributes for this groupType if it has not been created yet. It will be called etc:legacy:attribute:legacyAttributeDef_<name of groupType>. As part of this, a scope would be set too: legacyAttributeDef_<name>.getAttributeDefScopeDelegate().assignScope(AttributeDefScopeType.idEquals, legacyGroupType_<name>.getId(), null); Finally, create attribute in new framework. Calling it etc:legacy:attribute:legacyAttribute_<name>.
GroupType.addOrUpdateAttribute(session, name, read, write, required)
 - same as above
GroupType.addList(session, name, read, write)
 - We're still supporting custom lists. Still deprecate in favor of separate groups.
GroupType.delete(session)
 - deprecate
 - Delete the two attribute definitions.
GroupType.deleteField(session, name)
 - deprecate
 - If the field is for an attribute, if there aren't any assignments, delete the legacyAttribute_<name>.
GroupType.getFields()
 - deprecate
 - Note that attributes are no longer fields.

GroupTypeFinder

GroupTypeFinder.find(name, exceptionIfNotFound)
 - deprecate
 - Create GroupType objects based on attributeDefNames in etc:legacy:attribute that start with legacyGroupType_.
 - If you have code that searches for the "base" type, you'll get an error now.
GroupTypeFinder.findByUuid(uuid, exceptionIfNotFound)
 - deprecate
 - Same as above
GroupTypeFinder.findAll()
 - deprecate
 - Just return all
GroupTypeFinder.findAllAssignable()
 - deprecate
 - Just return all

GroupTypeTuple

 - Get rid of the grouper_group_types table.
 - Everything is essentially deprecated.
 - The uuid of a GroupTypeTuple will be the uuid of the attribute assignment of legacyGroupType_<name>.

Field

 - We're not going to treat legacy attributes as fields anymore or support the 2.1 API that does.
 - Getting rid of the columns grouptype_uuid and is_nullable.

 - Field.isAttributeName()
   - deprecate
   - Would always return false since legacy attributes are no longer fields.

FieldFinder

 - FieldFinder.findFieldIdForAttribute(attributeName, exceptionIfNull)
   - deprecate
   - This won't work anymore since attributes aren't fields.

FieldType

 - "attribute" is no longer a valid field type.

Group

- Group.initGroupAttributes(groups)
  - Seems like this was set up to address a performance issue with getting legacy attributes.
  - Deprecate and allow it to continue to work. That is, efficiently query for legacy attributes in etc:legacy:attribute.
- Group.deleteAttribute(attributeName)
  - deprecate
- Group.deleteAttribute(attributeName, failOnRequiredAttribute)
  - deprecate
  - Throw exception if failOnRequiredAttribute is set.
  - rewrite to delete from new attribute framework.
- Group.getAttributeValue(attributeName, checkSecurity, exceptionIfNotFound)
  - deprecate
  - exceptionIfNotFound is currently based on a field check. Since legacy attributes won't be fields, this needs to change to instead check all the attributeDefNames starting with legacyAttribute_ under etc:legacy:attribute that are applicable for this group.
  - checkSecurity needs to check security based on new attribute framework.
- Group.getAttribute(attributeName)
  - already deprecated. why doesn't this just call getAttributeValue???
- Group.getAttributes()
  - already deprecated. why doesn't this just call getAttributesMap???
- Group.setAttribute(attributeName, value)
  - deprecate
- Group.setAttribute(attributeName, value, checkPrivileges)
  - deprecate
- Group.setAttribute(attributeName, value, checkPrivileges, uuid)
  - deprecate
  - Change to use new framework. The uuid should be used as the uuid of the attribute assignment.
- Group.getAttributesMap(checkSecurity)
  - deprecate
  - Change to use new framework.
- Group.getAttributesDb()
  - already deprecated. why doesn't this just call getAttributesMap???
- Group.internal_setAttributes(attributes)
  - deprecate
- Group.setAttributes(attributes)
  - deprecate
- Group.internal_copy()
  - This continues to only work with legacy attributes for now.
  - Privilege checks on attributes need to use new framework.
- Group.addType(type)
  - deprecate
- Group.addType(type, exceptionIfAlreadyHasType)
  - deprecate
  - Assign legacyGroupType_<typeName> to group.
- Group.canReadField(subj, field) and various other similar methods
  - This will no longer work for legacy attributes.
- Group.deleteType(type)
  - deprecate
  - Change to use new framework (including privilege checks).
- Group.getRemovableTypes()
  - deprecate
  - All types are removable.
- Group.getTypes()
  - deprecate
  - No types are "internal"
- Group.getTypesDb()
  - deprecate
  - Return all assignments within etc:legacy:attribute.
- Group.hasType(type)
  - deprecate
- Group.setTypes(types)
  - deprecate

Attribute

- Basically deprecate everything.
- Get rid of grouper_attributes table.
- The attribute id should be the uuid of the attribute assignment.
- field id is no longer applicable.

Privileges

One of the challenges with the migration to Grouper 2.2 is around dealing with privileges. The legacy and new attribute frameworks have different ways of dealing with read and write privileges on attributes.

With the legacy attribute framework, read access to an attribute assigned to a group is based on a specified level of access on that group. For instance, in order to read an attribute assigned to a group, you may need read or update or admin privilege on the group. The same applies to write access in the legacy attribute framework.

With the new attribute framework, before v2.2, read access on an attribute assigned to a group required view privilege on the group and attr_read privilege on the attribute definition. And update access required admin privilege on the group and attr_update privilege on the attribute definition.

Privileges for attributes have changed for 2.2. More information. The 2.2 upgrade steps do not try to migrate privileges for either the legacy attribute or new attribute frameworks.