Grouper attribute based access control with JEXL script loaded groups

In Grouper 2.6.6 there is a first pass at JEXL loaded groups. It is basic and can be built on. Note: this is subject to change as we see a working solution and discuss the optimal path forward.

See the blog!

For more info, see the blog on Attribute Based Access Control with Grouper, from February 2022

JEXL scripts facilitate implementing the part of ABAC that defines who is included in a policy based on attributes of those users. Other parts of ABAC such as resource attributes or environment attributes can be taken into consideration with Grouper permissions or by the service which has protected resources.

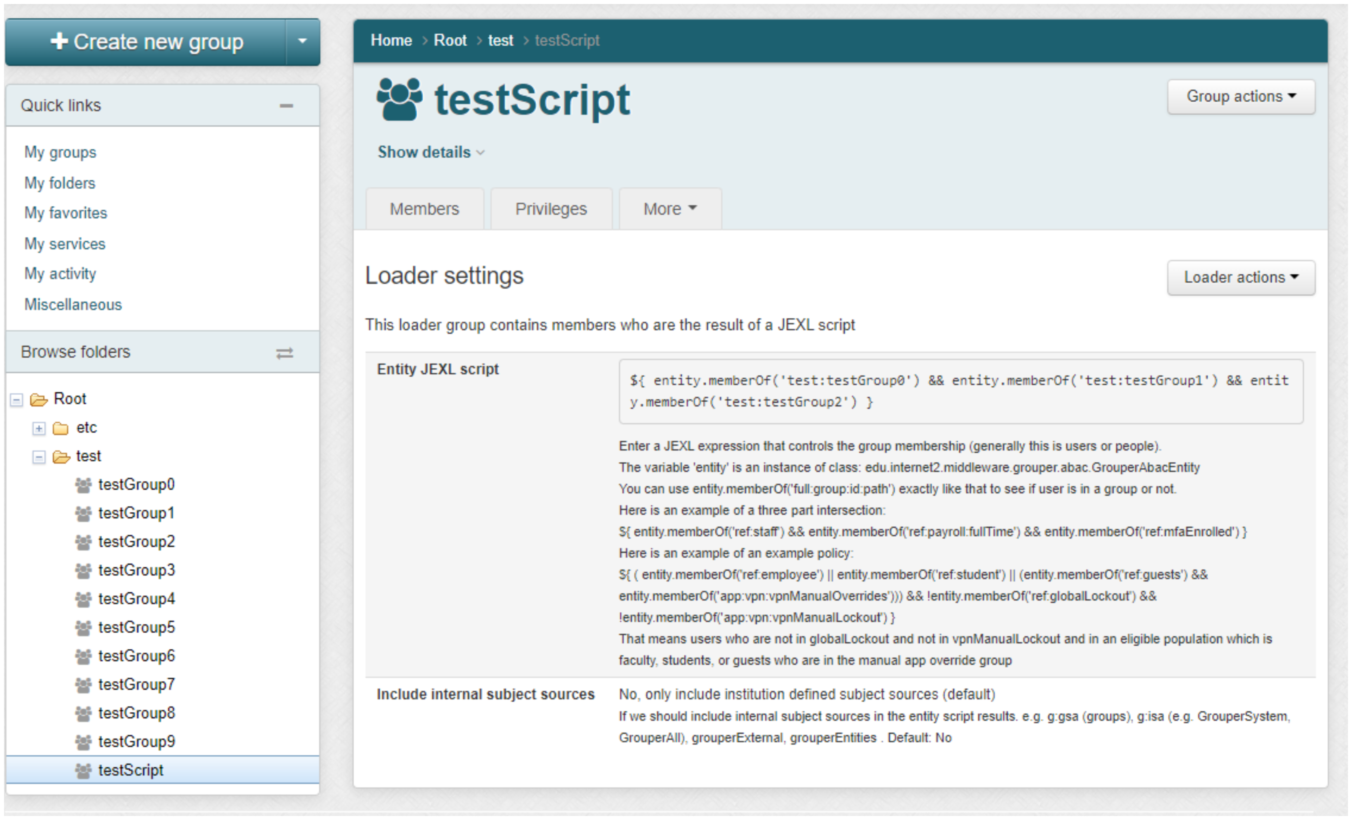
We want to be able to craft policies by an expression instead of creating loaders or tons of reference groups based on cartesian products of basis/ref groups.

Individual groups can be configured to automatically have their membership managed with individual subjects (or in future groups as members)

Why do we need this feature?

- Reduces pre-loaded rollups that might not be used
- You don't need a loader job for each one of these groups
- Any Grouper user could edit the policies if they can READ underlying groups. The expressions are secure (future state)
- The memberships of the ABAC groups are near real time based on an intelligent change log consumer (future state)
- You can have a UI to help build it and give good error messages
- Could visualize the policies. Perhaps could be integrated into existing visualization (future state)
- This solves the issue of composites with any number of factors

1.1. UI to configure



testScript

Loader settings

This loader group contains members who are the result of a JEXL script

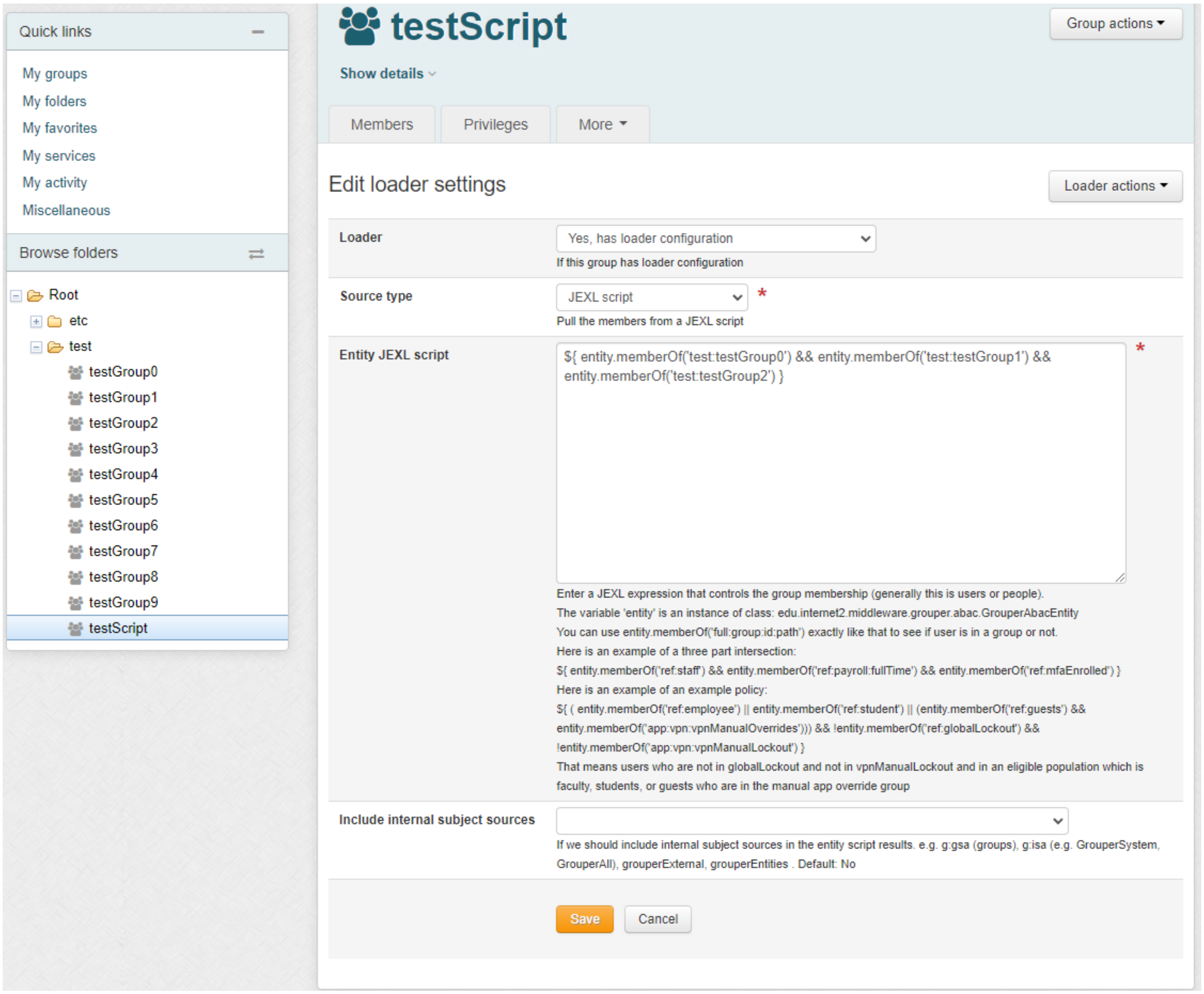
Entity JEXL script

```
${ entity.memberOf('test:testGroup0') && entity.memberOf('test:testGroup1') && entit
y.memberOf('test:testGroup2') }
```

Enter a JEXL expression that controls the group membership (generally this is users or people).
The variable 'entity' is an instance of class: edu.internet2.middleware.grouper.abac.GrouperAbacEntity
You can use entity.memberOf('full:group:id:path') exactly like that to see if user is in a group or not.
Here is an example of a three part intersection:
${ entity.memberOf('ref:staff') && entity.memberOf('ref:payroll:fullTime') && entity.memberOf('ref:mfaEnrolled') }
Here is an example of an example policy:
${ (entity.memberOf('ref:employee') || entity.memberOf('ref:student') || (entity.memberOf('ref:guests') && entity.memberOf('app:vpn:vpnManualOverrides'))) && !entity.memberOf('ref:globalLockout') && !entity.memberOf('app:vpn:vpnManualLockout') }
That means users who are not in globalLockout and not in vpnManualLockout and in an eligible population which is faculty, students, or guests who are in the manual app override group

Include internal subject sources

No, only include institution defined subject sources (default)
If we should include internal subject sources in the entity script results. e.g. g:gsa (groups), g:isa (e.g. GrouperSystem, GrouperAll), grouperExternal, grouperEntities . Default: No

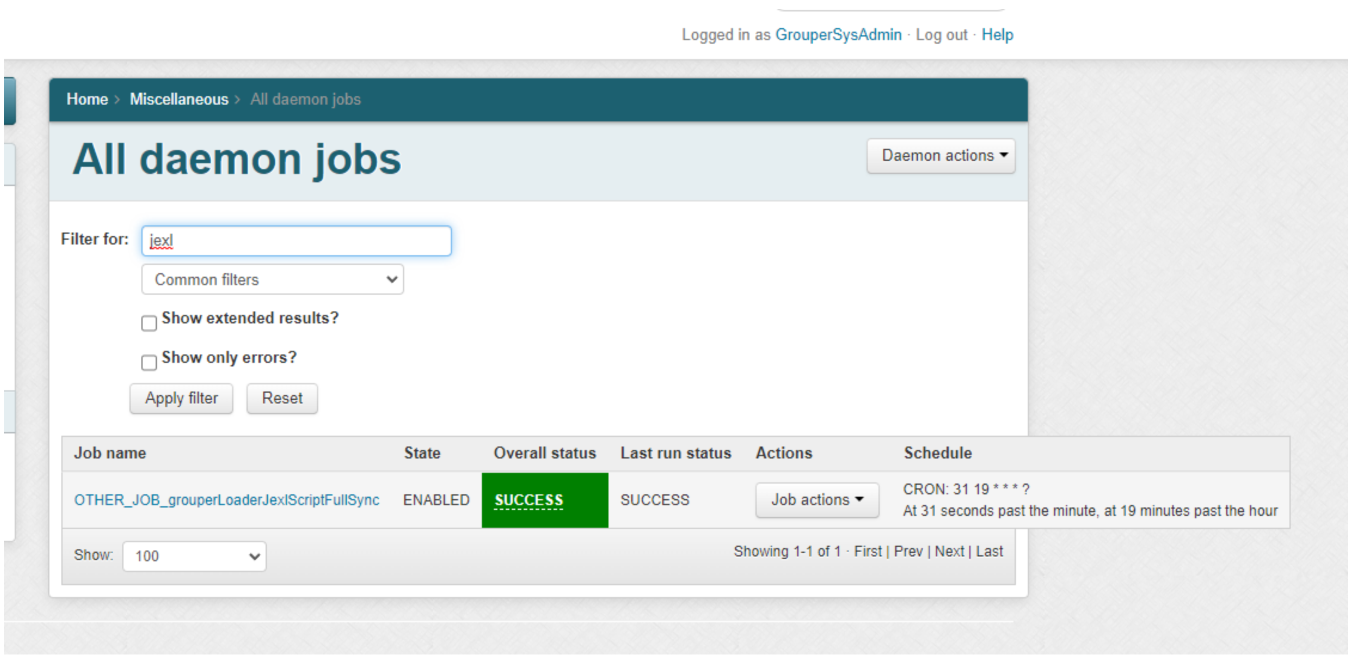


Edit loader settings

Loader	Yes, has loader configuration If this group has loader configuration
Source type	JEXL script * Pull the members from a JEXL script
Entity JEXL script	${ entity.memberOf('test:testGroup0') && entity.memberOf('test:testGroup1') && entity.memberOf('test:testGroup2') } * Enter a JEXL expression that controls the group membership (generally this is users or people). The variable 'entity' is an instance of class: edu.internet2.middleware.grouper.abac.GrouperAbacEntity You can use entity.memberOf('full:group:id:path') exactly like that to see if user is in a group or not. Here is an example of a three part intersection: ${ entity.memberOf('ref:staff') && entity.memberOf('ref:payroll:fullTime') && entity.memberOf('ref:mfaEnrolled') } Here is an example of an example policy: ${ (entity.memberOf('ref:employee') || entity.memberOf('ref:student') || (entity.memberOf('ref:guests') && entity.memberOf('app:vpn:vpnManualOverrides'))) && !entity.memberOf('ref:globalLockout') && !entity.memberOf('app:vpn:vpnManualLockout') } That means users who are not in globalLockout and not in vpnManualLockout and in an eligible population which is faculty, students, or guests who are in the manual app override group
Include internal subject sources	If we should include internal subject sources in the entity script results. e.g. g:gsa (groups), g:isa (e.g. GrouperSystem, GrouperAll), grouperExternal, grouperEntities . Default: No

1.2. Daemon screen

Note in Grouper v2.6.6 you need to wait an hour after changing a script, or run the JEXL script loader full job. In the future we will have an incremental and run the full nightly. Note: there is one full daemon that handles all of the JEXL script ABAC groups. You do not add this, it is built-in


Logged in as GrouperSysAdmin · Log out · Help

Home › Miscellaneous › All daemon jobs

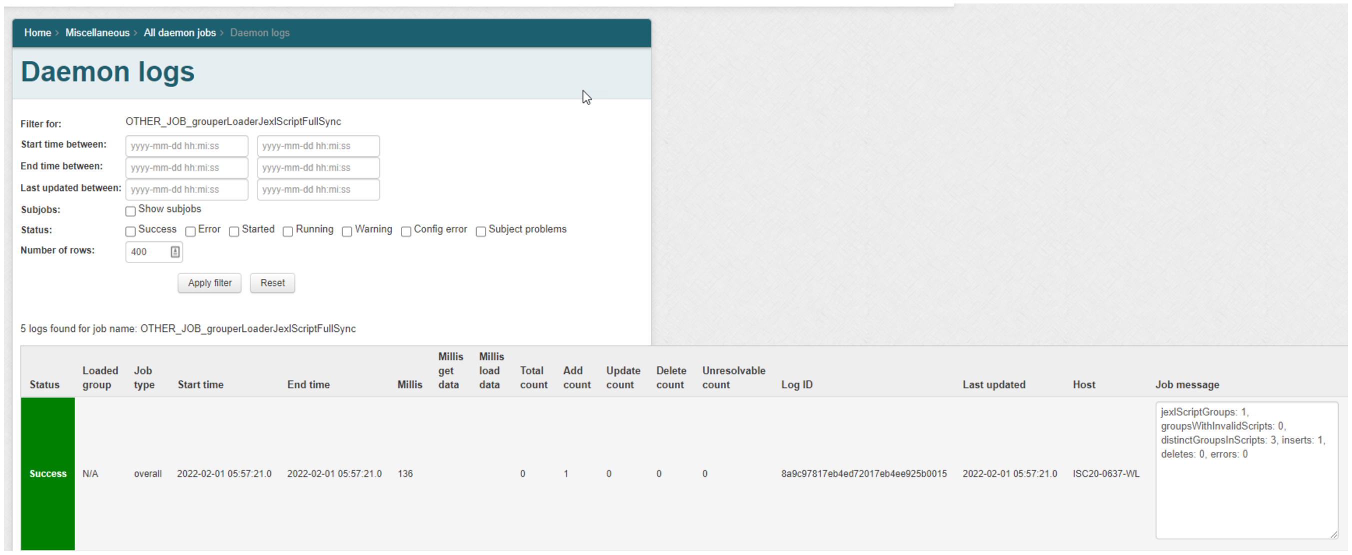
All daemon jobs

Filter for: jexl

Job name	State	Overall status	Last run status	Actions	Schedule
OTHER_JOB_grouperLoaderJexlScriptFullSync	ENABLED	SUCCESS	SUCCESS	Job actions	CRON: 31 19 * * * ? At 31 seconds past the minute, at 19 minutes past the hour

Show: 100

Showing 1-1 of 1 · First | Prev | Next | Last


Home › Miscellaneous › All daemon jobs › Daemon logs

Daemon logs

Filter for:	OTHER_JOB_grouperLoaderJexlScriptFullSync
Start time between:	yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss
End time between:	yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss
Last updated between:	yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss
Subjobs:	Show subjobs
Status:	Success Error Started Running Warning Config error Subject problems
Number of rows:	400

5 logs found for job name: OTHER_JOB_grouperLoaderJexlScriptFullSync

Status	Loaded group	Job type	Start time	End time	Millis	Millis get data	Millis load data	Total count	Add count	Update count	Delete count	Unresolvable count	Log ID	Last updated	Host	Job message
Success	N/A	overall	2022-02-01 05:57:21.0	2022-02-01 05:57:21.0	136			0	1	0	0	0	8a9c97817eb4ed72017eb4ee925b0015	2022-02-01 05:57:21.0	ISC20-0637-WL	jexlScriptGroups: 1, groupsWithInvalidScripts: 0, distinctGroupsInScripts: 3, inserts: 1, deletes: 0, errors: 0

1.3. Scripts

The script can only be written by people who can READ groups in the script and UPDATE the owner group. Since this is actually a JEXL script (not a JEXL expression), so you could have multiple lines, variables, conditionals, etc

In an entity script, the variable 'entity' is an instance of class: edu.internet2.middleware.grouper.abac.GrouperAbacEntity

You can use entity.memberOf('full:group:id:path') exactly like that to see if user is in a group or not.

Expression	Description
`${ entity.memberOf('ref:staff') && entity.memberOf('ref:payroll:fullTime') && entity.memberOf('ref:mfaEnrolled') }`	Three part intersection. Full time staff in MFA

`${ (entity.memberOf('ref:employee') || entity.memberOf('ref:student') // employees or students || (entity.memberOf('ref:guests') && entity.memberOf('app:vpn:vpnManualOverrides'))) // or guests who are in manual allow && !entity.memberOf('ref:globalLockout') && !entity.memberOf('app:vpn:vpnManualLockout') } // and not in either lockout group`	Example policy That means users who are not in globalLockout and not in vpnManualLockout and in an eligible population which is faculty, students, or guests who are in the manual app override group
`${ entity.memberOf('app:vpn:users') != entity.memberOf('ref:mfaEnrolled') }`	Exclusive OR This is VPN users not in MFA and MFA users not in VPN:

1.4. How it works

There are some trade-offs with performance and resources. This is the current implementation. It is optimized to reduce run-time. It does use a lot of memory, though that was a consideration.

- Sees which groups are in the script
- Get the memberships of the owner groups and all script groups (only get the memberId, sourceId, and groupId)
- Consider if configured to include internal subject sources (adjust the membership lists)
- For each member of either the owner group or the script groups for that owner
    - Setup the variables for a JEXL script based on the bulk queries
    - Evaluate the script
    - If the result does not match the current state of the membership, add or remove the member from the owner group
    - If a script evaluation fails, proceed with the job

1.5. TO DO

- Allow common reference groups to have user friendly labels

    -
      ```
      e.g. employee instead of ref:employee
      ```
    - Do not allow plurals of labels, and absorb plurals. e.g. "member of employee" same as "member of employees
    - Same for case differences
    - Absorb smart quotes
    - Give a warning (validation) and suggestion when parens are needed
- alow group friendly names, uuid, idIndex, case issues, etc to be used but converted into the id-path
- Document common allowed reference groups and syntax on the script editing page
- Allow natural language to be converted into JEXL
    - e.g.

      ```
      FROM
      Member of employee, and member of app:jiraAdminsManualTO
      ${entity.memberOf(‘ref:employee’) && entity.memberOf(‘app:jiraAdminsManual’)}
      ```
    - e.g.

      ```
      FROM
      Has affiliation attribute with name of staff and dept of english,
         but not a member of lockout

      TO
      entity.hasAttribute('affiliation', 'name==staff && dept==english')
         && entity.notMemberOf('ref:lockout')
      ```
- Load groups as members
- Add incremental job
- Unit tests
- Better validation
- More methods to call (other than hasMember)

- Add subject attributes e.g. from global attribute resolver
- Failsafes
- Add delegated admin based on READ of groups
- Visualization
- Add more counts of total memberships etc
- See if comments can be included in scripts
- See if we can replace composite type with immediate in membership table if replacing a composite with jexl script
- Have a confirm screen that tells the user information about what will happen (tie into visualization)? (have a progress page)
- Add UI to configure scripts?
- Test button while writing to see if valid (maybe that does counts too)
- Add dependency graph for precedence or recalcs in full sync
- Do not allow circular references
- Identify full syncs to be hourly or daily
- Allow single quotes or double quotes when parsing scripts

1.6. Entity data fields

Setup your entity data fields and use that data in JEXL scripts. This data can come from LDAP or SQL

Gail Lift: Michigan is starting to see situations where this approach would be REALLY useful. We are also interested in near real time / changelog.

We have struggled to build widely usable reference groups, because each unit has their own special needs: "Regular staff, but only in these jobcodes" and "Regular staff, but only in those jobcodes". Because a person can have multiple jobs, or multiple student programs, simple group math is not enough. If Mary has 2 jobs, {"dept":"English","jobcode":"12345"} and {"dept":"History","jobcode":"67890"}, intersecting a dept ref group with a jobcode ref group will not work as desired.

We are hoping the the new approach with Entity Attribute Resolver Groups will help with these needs. ~~But we are concerned about how to get all the needed data into the my_people_affiliation table~~. (you could feed from arbitrary LDAP/SQL/WS sources)

We have affiliation data for employees, emeritus, Ann Arbor students, Dearborn students, Flint students, alumni, and Sponsored Affiliates (guests). Class enrollment data will be added later. Each affiliations has its own set of attributes that need to be available for group construction. A couple of typical samples, seen as we store them in LDAP in an almost-JSON format:

umichAAAcadProgram: {acadCareer=GBA}:{acadProg=00018}:{acadPlan=0010MAC}:{campus=A}:{progStatus=AC}:{admitTerm=2410}:{admitTermBegDt=2022-08-29}:{expGradTerm=}:{degrChkoutStat=}:{acadCareerDescr=Graduate Business Admin}:{acadPlanDegree=MAC}:{acadPlanDescr=Accounting MAcc}:{acadPlanField=0010}:{acadPlanFieldDescr=}:{acadPlanType=MAJ}:{acadPlanTypeDescr=Major}:{acadGroup=BA}:{acadGroupDescr=Ross School of Business}:{acadProgDescr=Accounting MAcc}

umichHR: {jobCategory=Faculty}:{campus=UM_ANN-ARBOR}:{deptId=304000}:{deptGroup=MEDICAL_SCHOOL}:{deptDescription=MM Orthopaedic Surgery}:{deptGroupDescription=Medical School}:{deptVPArea=EXEC_VP_MED_AFF}:{jobcode=201000}:{jobFamily=10}:{emplStatus=A}:{regTemp=R}:{supervisorId=12345678}:{tenureStatus=TEN}:{jobIndicator=P}

~~For a single my_people_affiliation table,~~ having a column for each distinct keyword would require about 70 columns. In any given row, most columns would have a null value. At the other extreme, we could use a single column for the affiliation data, so the columns would be employee_id, affiliation_name, affiliation_value. At this extreme, most queries would require substring matching. Would a structure between these make sense? Is a separate table for each affiliation better? (Dont worry about a table with all data)

1.7. Parse expression with JEXL (for Grouper developers)

Feed the expression through this simple program

```
  public static void main(String[] args) {

    JexlEngine jexlEngine = new JexlEngine();

    ExpressionImpl expression = (ExpressionImpl)jexlEngine.createExpression("group.campus !~ ['palmer',
'southern'] and group.termStart - 7 > sysdate");

    ASTJexlScript astJexlScript = (ASTJexlScript)GrouperUtil.fieldValue(expression, "script");
    printNode(astJexlScript, "");

    System.out.println(expression);
  }

  public static void printNode(JexlNode jexlNode, String prefix) {
    System.out.println(prefix + jexlNode.getClass().getSimpleName() + (StringUtils.isBlank(jexlNode.image) ?
"" : (": " + jexlNode.image)));
    String newPrefix = StringUtils.isBlank(prefix) ? "- " : ("  " + prefix);
    for (int i=0;i<jexlNode.jjtGetNumChildren();i++) {
      printNode(jexlNode.jjtGetChild(i), newPrefix);
    }
  }
```

Output

```
ASTJexlScript
- ASTAndNode
  - ASTNRNode
    - ASTReference
      - ASTIdentifier: group
      - ASTIdentifier: campus
    - ASTReference
      - ASTArrayLiteral
        - ASTReference
          - ASTStringLiteral: palmer
        - ASTReference
          - ASTStringLiteral: southern
  - ASTGTNode
    - ASTAdditiveNode
      - ASTReference
        - ASTIdentifier: group
        - ASTIdentifier: termStart
      - ASTAdditiveOperator: -
      - ASTNumberLiteral: 7
    - ASTReference
      - ASTIdentifier: sysdate
```

Grouper can take that object model and see which group and subject attributes are related, print out a nice analysis of the policy, and know which policies are affected by real time changes

Expression 2: campus is palmer or southern, or the term is current with some overlap

```
group.campus =~ ['palmer', 'southern'] or (group.termStart - 7 > sysdate and group.termStart - 7 < sysdate)
```

```
ASTJexlScript
- ASTOrNode
  - ASTERNode
    - ASTReference
      - ASTIdentifier: group
      - ASTIdentifier: campus
    - ASTReference
      - ASTArrayLiteral
        - ASTReference
          - ASTStringLiteral: palmer
        - ASTReference
          - ASTStringLiteral: southern
  - ASTReference
    - ASTReferenceExpression
      - ASTAndNode
        - ASTGTNode
          - ASTAdditiveNode
            - ASTReference
              - ASTIdentifier: group
              - ASTIdentifier: termStart
            - ASTAdditiveOperator: -
            - ASTNumberLiteral: 7
          - ASTReference
            - ASTIdentifier: sysdate
        - ASTLTNode
          - ASTAdditiveNode
            - ASTReference
              - ASTIdentifier: group
              - ASTIdentifier: termStart
            - ASTAdditiveOperator: -
            - ASTNumberLiteral: 7
          - ASTReference
            - ASTIdentifier: sysdate
```

Expression 3: primaryAffiliation is faculty or staff and dept is physics or math

```
person.primaryAffiliation =~ ['faculty', 'staff'] and person.dept =~ ['physics', 'math']
```

```
ASTJexlScript
- ASTAndNode
  - ASTERNode
    - ASTReference
      - ASTIdentifier: person
      - ASTIdentifier: primaryAffiliation
    - ASTReference
      - ASTArrayLiteral
        - ASTReference
          - ASTStringLiteral: faculty
        - ASTReference
          - ASTStringLiteral: staff
  - ASTERNode
    - ASTReference
      - ASTIdentifier: person
      - ASTIdentifier: dept
    - ASTReference
      - ASTArrayLiteral
        - ASTReference
          - ASTStringLiteral: physics
        - ASTReference
          - ASTStringLiteral: math
```

1.8. Analyze policy

To confirm a policy is correct, a long form translation of the policy can be displayed along with group names and group counts (future state)

1.9. Visualization

This is a complicated topic since it is parsing a programming language.

As a first pass we could have the overall group and lines from all component groups with count and the exact policy isnt there. This would apply for any unparsable policies once we have better visualization.

In a future pass, if the script follows certain standards and is "parsable" (allow-deny where parens and multiples could be involved), then I could picture a visualization for that. Note: either the visualization will be slow, or lots of data will be cached, or it will be stale from the last full sync. This is because JEXLs cannot be transformed into queries with counts, it needs the data in memory to allow JEXL to do its thing.

We need to get a list of sample policies people want to use so we can make sure we are going in the right direction.

1.10. Full sync

A nightly full sync will occur. The incremental sync should stop. Make sure all the loaded groups are up to date.

1.11. Incremental sync (future state)

An incremental change log consumer can

- If group attributes change, see if it affects group attributes (future state)
- If attributes change, see which policies those refer to, and incrementally adjust the membership of those groups
- Policy changes should change the population