GEL Scripts | Introduction

Your Guides: Ben Rimmasch and Luis Palacios

Introductions

- Take 5 Minutes
- Turn to a Person Near You
- Introduce Yourself
- Business Cards

Agenda

- What Is GEL
- GEL Script Structure
- Commonly Used Tags
- Best Practices
- Example Use Cases
- Exercise

Part I:
What Is GEL

Let Rego be your guide.

What Is GEL

- GEL stands for Generic Execution Language
- Based on Jelly, a jakarta.apache.org Commons project
- Extended and embedded into Clarity PPM to enable custom logic to solve business problems
- GEL scripts can be executed from within Clarity PPM processes, or from the command line
- GEL supports the use of Java methods, SQL, and web services
- Additional information can be found in the CA Documentation (Developer Guide) and at the Apache Jelly website at

  http://jakarta.apache.org/commons/jelly/index.html

Part II:
GEL Structure

Let Rego be your guide.

GEL Script Structure

Header

```
<gel:script
    xmlns:core="jelly:core"
    xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
    xmlns:sql="jelly:sql" >
```

Comment ⟶ <!-- CODE GOES HERE -->

Footer ⟶ </gel:script>

Note: An entire script always resides within the GEL script tag

GEL Script Structure - Tags

- A GEL script is built from qualified elements bound to java code called tags
- Using namespace declarations, tags are organized into tag libraries which are made available in a script
- Clarity PPM ships with many out of the box tag libraries

Hello World Example:

```
<gel:script xmlns:core="jelly:core"
            xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary">
    <core:forEach indexVar='i' begin='1' end='3'>
            <gel:out>Hello World ${i}!</gel:out>
    </core:forEach>
</gel:script>
```

GEL Script Structure - Common Namespaces

```
xmlns:core="jelly:core"
xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
xmlns:sql="jelly:sql"
xmlns:xog=http://www.niku.com/xog"
xmlns:soap="jelly:com.niku.union.gel.SOAPTagLibrary"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:file="jelly:com.niku.union.gel.FileTagLibrary"
xmlns:util="jelly:util"
xmlns:email="jelly:email"
xmlns:ftp="jelly:com.niku.union.gel.FTPTagLibrary"
```

GEL Script Structure - Variables

- Used extensively throughout GEL scripts
- Most tags can set variables
- Ex:
  - <core:set var="var_name">variable value</core:set>
  - <core:set var="var_name" value="variable value"/>
- Referencing a variable
  - ${var_name}
- Typically, variables are valid for the current script only

Part III:
Commonly Used Tags

Let Rego be your guide.

Conditionals / Loops

- **<core:if>**

```
<core:if test="${!empty(sqlException)}">
        …
</core:if>
```

- **<core:choose>**

```
<core:choose>
        <core:when test="${sessionID == null}">
                …
        </core:when>
        <core:otherwise>
                …
        </core:otherwise>
</core:choose>
```

Conditionals / Loops

- **<core:forEach>**

```
<core:forEach trim="true" items="${queryResult.rows}" var="row">
    …
</core:forEach>
```

- **<gel:forEach>**

```
<gel:forEach select="$projectsXML/NikuDataBus/Projects/Project" var="currentPrj">
    …
</gel:forEach>
```

- Two types of forEach loop types.
  - CORE version performs basic FOR looping, iterating over elements.
  - GEL version iterates over XML elements

Conditionals / Loops

- **<core:while>**

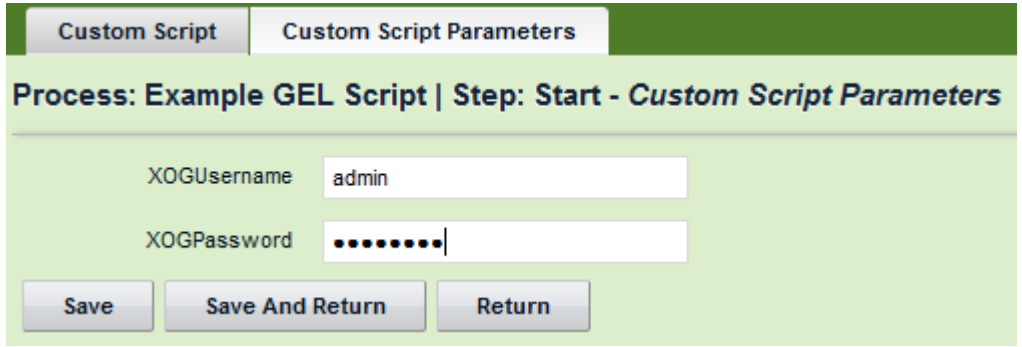
```
<core:while test="${condition == true}">
    …
</core:while>
```

Variables / Parameters - Built-in Parameters

- Custom Action GEL scripts associated with processes have the following parameters available to them:
  - **Object instance ID**
    - If no object is associated with the process, the ID is -1. Otherwise the ${gel_objectInstanceId} parameter contains the object instance ID.
  - **Process ID**
    - ${gel_processId} is the process identifier; all instances of this process share this identifier.
  - **Process instance ID**
    - ${gel_processInstanceId} is the process instance identifier; all instances have a unique value.
- All built-in parameters have a "gel_" prefix and are of data type - numeric.

Variables / Parameters

- **<gel:parameter>**
  - Allows values to be passed into a GEL script from a Clarity PPM process.
  - Refer to the parameter as you would any other variable, using the ${variable_name} syntax.
  - Optional attribute secure="true" causes Clarity PPM to hide the actual value in the user interface with asterisks (*).
- Ex:
  - <gel:parameter var="XOGUsername" default="admin"/>
  - <gel:parameter var="XOGPassword" default="password" secure="true"/>



Variables / Parameters

- **<core:set>**
  - Used to set basic variables

```
<core:set var="yes" value="1"/>
<gel:out>${yes}</gel:out>
```

  - You can do some basic math on the variable:

```
<gel:out>${yes+2}</gel:out>
```

- **<gel:set>**
  - Use this tag when it is necessary to extract the value of the variable from an XML document.
  - Requires a select attribute which in turn requires an XPath statement.
  - Ex: the select statement points the way to the Statistics node of a XOG output file.

```
<gel:set asString="true" select="$result//XOGOutput/Statistics" var="xogStats"/>
```

Variables / Parameters

- **<gel:persist>** Allows you to set variables with a scope that extends beyond the current script.

- **<gel:parse>** Used to create an XML document in memory. This is how you will build XOG requests. The tag can be used to generate an entire XML document, or specific nodes that can later be attached into an existing XML document.

```
<gel:parse var="loadContent">
 <NikuDataBus xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:noNamespaceSchemaLocation="../xsd/nikuxog_resource.xsd">
 <Header version="12.0.0.5028" action="write" objectType="resource“ externalSource=“PS"/>
  <Resources>
   <Resource resourceId="abc" isActive="true">
    <PersonalInformation lastName="doe" firstName="john" emailAddress="jdoe@ca.com"/>
   </Resource>
  </Resources>
 </NikuDataBus>
</gel:parse>
```

Variables / Parameters – Case Sensitive Issues

- Information contained within GEL tags is case sensitive. For example, if you declare a variable as follows:

```
<core:set var="v_ProjectID">PRJ-123456</core:set>
```

- Then reference the variable as follows:

```
<gel:out>${v_projectid}</gel:out>
```

- This will not output PRJ-123456. However, this will not cause an actual error to occur, so you can't "catch" this error.

Database Operations - Datasources

- **<gel:setDataSource/>**
  - Uses the connection properties from Clarity PPM's properties (set in the CSA)
  - On Premise installations can create additional external database definitions, and reference them in scripts as well
  - Var attribute is optional. If not specified and only one datasource is set, then all SQL calls will use that datasource.
  - If you set a datasource variable you are required to reference it in subsequent tags.

```
<gel:setDataSource dbId="Niku" var="clarityDS"/>
<sql:query dataSource="${clarityDS}" var="result">
    SELECT 1 from dual
</sql:query>
```

Database Operations - Datasources

- **<sql:setDataSource>**
  - Reasons to use the <sql:setDataSource> tag
    - You are unable to add the database connection to the CSA
    - You need to connect to a DB that isn't Oracle or SQL Server
    - You want to run the script from the command line and don't have a copy of the properties.xml file on your computer

```
<sql:setDataSource url="jdbc:oracle:thin:@localhost:1521:NIKU"
    driver="oracle.jdbc.driver.OracleDriver"
    user="${ClarityUser}"
    password="${ClarityPassword}"
    var="clarityDS"/>
```

Database Operations – SQL Sources

- **<sql:query>**
  - The result set is stored in the variable declared within the tag

```
<sql:query dataSource="${clarityDS}" escapeText=“false" var="result">
        <![CDATA[
        SELECT  r.full_name resName,
                    r.email resEmail
        FROM    srm_resources r
        ]]>
</sql:query>
```

- **Note**: Due to the nature of XML, using certain characters (‘<‘, ‘>’) within your query would cause an error without enclosing the query in the CDATA tag. Because of this, it is a best practice to include this tag in all queries.

Database Operations – SQL Queries

- Retrieving the data from the query

```
<!-- By Column Name (Recommended Method) -->
<core:forEach trim="true" items="${result.rows}" var="row">
      <gel:out>Resource Name: ${row.resName}</gel:out>
</core:forEach>

<!-- By Index -->
<core:forEach trim="true" items="${result.rowsByIndex}" var="row">
      <gel:out>Resource Name: ${row[0]}</gel:out>
</core:forEach>
```

- **Note**: When possible, avoid setting a bunch of variables with the results of the query.

Database Operations – SQL Updates

- Processes often require updating the values of specific attributes.
- One of the ways to do this is with a SQL Update statement.
- Keep in mind the following when performing direct database updates
  - Avoid Insert statements – these are best suited for XOG
  - Direct database updates will not trigger a process to start – XOG updates typically will trigger processes to start
  - Avoid updating OOTB tables when possible – using XOG will ensure all Clarity business rules are followed
  - It is generally safe to update custom attributes with a SQL update. These are typically found in tables beginning with odf_ca_
  - Extra caution should also be used within On-Demand environments
  - If the table has them, make sure to also update the last_updated_date and last_updated_by columns

Database Operations – SQL Updates

- **<sql:update>**
  - The declared variable will contain the number of rows that the update statement affects

```
<sql:update dataSource="${clarityDS}" escapeText=“false" var="updateCnt">
    <![CDATA[
    UPDATE   odf_ca_project ocp
    SET      ocp.rego_appr_date = sysdate
             , ocp.last_updated_date = sysdate
             , ocp.last_updated_by = ${gel_processInitiator}
    WHERE    ocp.id = ${gel_objectInstanceId}
    ]]>
</sql:update>
```

- **Note**: Using CDATA tags in update statements is also recommended.

Database Operations – Bind Variables

- Ensures data types are correct
- Prevents SQL injection
- Allows reuse of SQL statements for performance gains

```
<sql:update dataSource="${clarityDS}" escapeText="0" var="updateCnt"><![CDATA[
        UPDATE   odf_ca_project ocp
        SET      ocp.rego_appr_date = sysdate
                 , ocp.last_updated_date = sysdate
                 , ocp.last_updated_by = ?
        WHERE    ocp.id = ?
        ]]>
        <sql:param value="${gel_processInitiator}"/>
        <sql:param value="${gel_objectInstanceId}"/>
</sql:update>
```

- **Note:** You must have a <sql:param> tag for each ? in the SQL statement, even if they are using the same value. The parameters also must be placed in the order that they appear in the statement.

Database Operations – SQL Comments

- Comments are useful guides in large complex queries to describe the intended behaviors
- SQL can be defined as either inline or block comment styles
  - E.g.

```
/* This is a block comment
 * It begins and ends with slash-star and star-slash markers
 */
-- This is an inline comment, starting from the double-hyphen until the end of the current line
```

- Block comments cannot be nested, just as XML block comments cannot be nested.
  - E.g.

```
/* Do /* not */ do comments this way, you will get an error */
```

- Inline comments are to be avoided
  - Due to XML parsing, line-endings may be ‘lost’ in SQL statements. This will then cause the inline comment to block out the rest of the query completely. The result can be script errors, or worse, unintended data changes and loss

Database Operations – SQL Comments

- Use block comments for augmenting the SQL to make it clearer and reduce complexity
- Do not use comments to disable chunks of SQL. Doing so increases the chances that you will accidentally nest block comments within each other, and increases complexity.
- Do this:

```
/* Apply to Contractors (301) only, not Employees (300) */
WHERE srm_resources.person_type = 301
```

- Don’t do this:

```
-- Apply to Contractors (301) only, not Employees (300)
WHERE srm_resources.person_type = 301
```

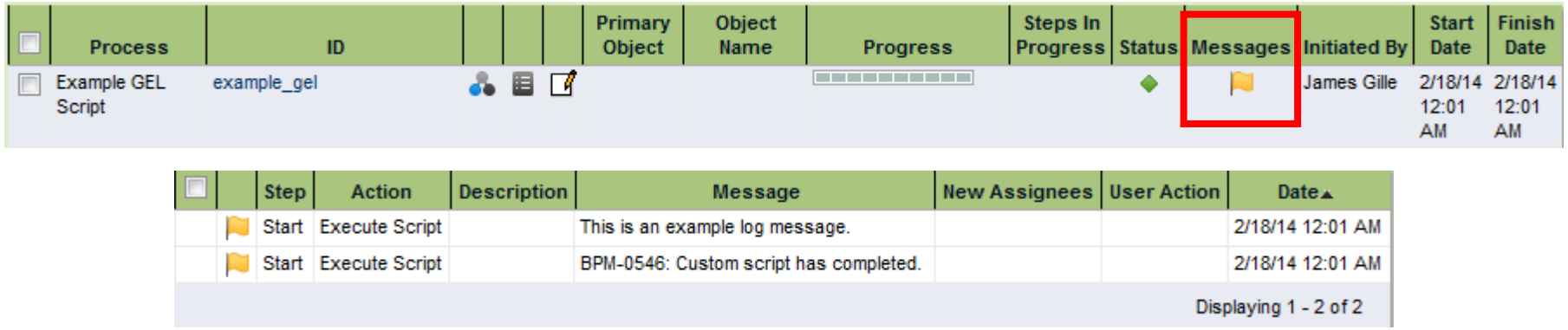
- Or this:

```
WHERE srm_resources.person_type = 301
/* or srm_resources.person_type = 300 */
```

Logging

- **<gel:log>**
  - Use this tag to insert status messages into the process engine log table
  - Very useful when debugging scripts
  - Avoid using too many logging statements, as it can impact performance
  - Different levels of logging – INFO, WARN, ERROR

```
<gel:log level="INFO">This is an example log message.</gel:log>
```



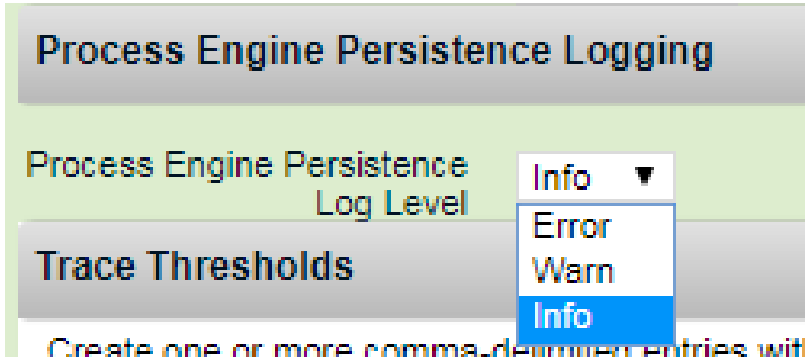
Process	ID	Primary Object	Object Name	Progress	Steps In Progress	Status	Messages	Initiated By	Start Date	Finish Date
Example GEL Script	example_gel							James Gille	2/18/14 12:01 AM	2/18/14 12:01 AM

Step	Action	Description	Message	New Assignees	User Action	Date
Start	Execute Script		This is an example log message.			2/18/14 12:01 AM
Start	Execute Script		BPM-0546: Custom script has completed.			2/18/14 12:01 AM

Displaying 1 - 2 of 2

Logging

- Process Engine Persistence Logging Setting
  - Newer versions of Clarity PPM have introduced a new setting that affects whether or not the logging statements show up in the process logs
  - Setting can be found at <servername>/niku/nu#action:security.loggerConfig
  - The default value is set to Error after the initial upgrade



Part IV:
Best Practices / Use Cases

Let Rego be your guide.

Best Practices

- If a variable can be changed by an admin, or stores a password, use the <gel:parameter> tag instead of <core:set>
- Comment your code
- Do not use comments to disable chunks of code (use source control instead)
- Properly format and indent your code
- Place all code within the <gel:script> tags, even comments
- Avoid excessive logging, but keep enough for debugging purposes – Create and use debug parameters to turn additional logging on and off
- When possible, pull server information from the properties file on the server

Best Practices

- Name variables clearly but simply, and avoid using arithmetic operators:

```
<gel:script xmlns:core="jelly:core" xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary">
  <core:set var="auth" value="2" />
  <core:set var="result" value="2" />
  <gel:out>First output: ${auth-result}</gel:out>

  <core:set var="auth-result" value="not zero" />
  <gel:out>Second output: ${auth-result}</gel:out>
</gel:script>

D:\Rego\Clarity>bin\gel moral.gel
First output: 0
Second output: 0
```

Example Use Cases

- Approval Workflows
- Timesheet Reminder Emails
- Updating Resource Rights
- Data Extraction
- Integrations with HR system
- Capturing Action Item Details on Object for Reporting

Part V: Exercise

Let Rego be your guide.

Exercise

- Write a script that updates an approver field with the resource that ran the process and a date field with the current date

  - SQL to get Resource that initiated the process:

```
SELECT   r.id resourceId
         , r.full_name resourceName
FROM     bpm_run_processes brp
         JOIN srm_resources r
      ON brp.initiated_by = r.user_id
WHERE    brp.id = ?
```

  - The bind parameter's value will be: ${gel_processInstanceId}

Questions?

regoUniversity 2019

SAN DIEGO

Let Rego be your guide.

Thank You For Attending regoUniversity

Instructions for PMI credits

- Access your account at pmi.org
- Click on **Certifications**
- Click on **Maintain My Certification**
- Click on **Visit CCR's** button under the **Report PDU's**
- Click on **Report PDU's**
- Click on **Course or Training**
- Class Name = **regoUniversity**
- Course Number = **Session Number**
- Date Started = **Today's Date**
- Date Completed = **Today's Date**
- Hours Completed = **1 PDU per hour of class time**
- Training classes = **Technical**
- Click on **I agree** and **Submit**

Let us know how we can improve!
Don't forget to fill out the class survey.

Phone
888.813.0444

Email
info@regouniversity.com

Website
www.regouniversity.com