LAS VEGAS

regoUniversity

2025

Sponsored by

ValueOps® by Broadcom | Clarity® by Broadcom | Rally® by Broadcom | ConnectALL by Broadcom | Insights by Broadcom

Basics of Programming

Your Guides:

James Gille and Jair Flores

Gold Sponsor

aws

Agenda

- Introduction
- Variables and Data Types
- Control Flow
- Logging
- Exception Handling
- SQL

Introduction

LAS VEGAS
regoUniversity
2025

What is GEL

- GEL stands for Generic Execution Language
- Based on Jelly, a jakarta.apache.org Commons project
- Extended and embedded into Clarity to enable custom logic to solve business problems
- GEL supports the use of Java methods, SQL, and web services
- Additional information can be found in the BC Documentation (Developer Guide) and at the Apache Jelly website at: https://jakarta.apache.org/commons/jelly/index.html

Why GEL

- Bypass limitations of blueprint rules
- Validations
- Notifications
- Complex Calculations
- Automations
- Integrations

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 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:set var="message" value="Hello World!"/>
    <gel:log>${message}</gel:log>
</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:util="jelly:util"
```

Variables and Data Types

Variables

- Data to be reused later in the script
- Ex:

```
<core:set var="total">0</core:set>
<core:set var="total" value="${total + 1}"/>
```

- Referencing a variable

```
${total}
```

- Many tags can set variables

Set Tag

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

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

  - You can do some basic math on the variable:

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

Case Sensitivity

- Variables names are 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:log>${v_projectid}</gel:log>
```

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

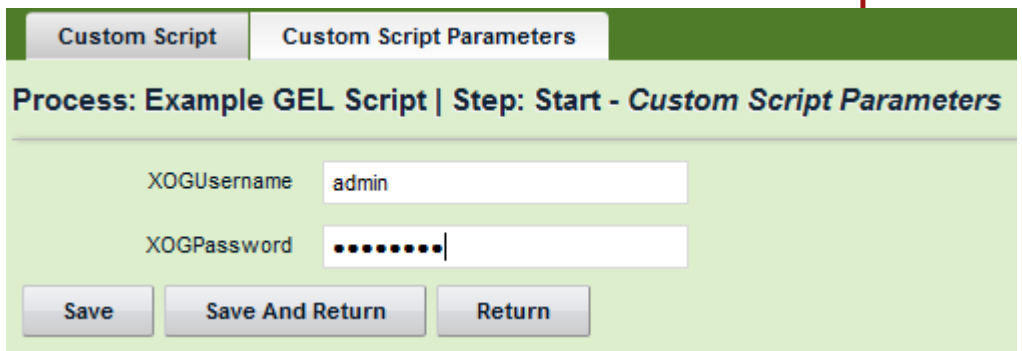
Built-in Variables

- Custom Action GEL scripts associated with processes have the following parameters available to them:
  - **Object instance ID**
    - ${gel_objectInstanceId} variable 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.
  - **Process initiator**
    - ${gel_processInitiator} is the user id of the person that kicked off the process.
- All built-in variables have a "gel_" prefix and are of data type - numeric.

Parameters

- **<gel:parameter>**
  - Allows values to be passed into a GEL script from a Clarity process.
  - Refer to the parameter as you would any other variable, using the ${variable_name} syntax.
  - Optional attribute secure="true" causes Clarity 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"/>
```

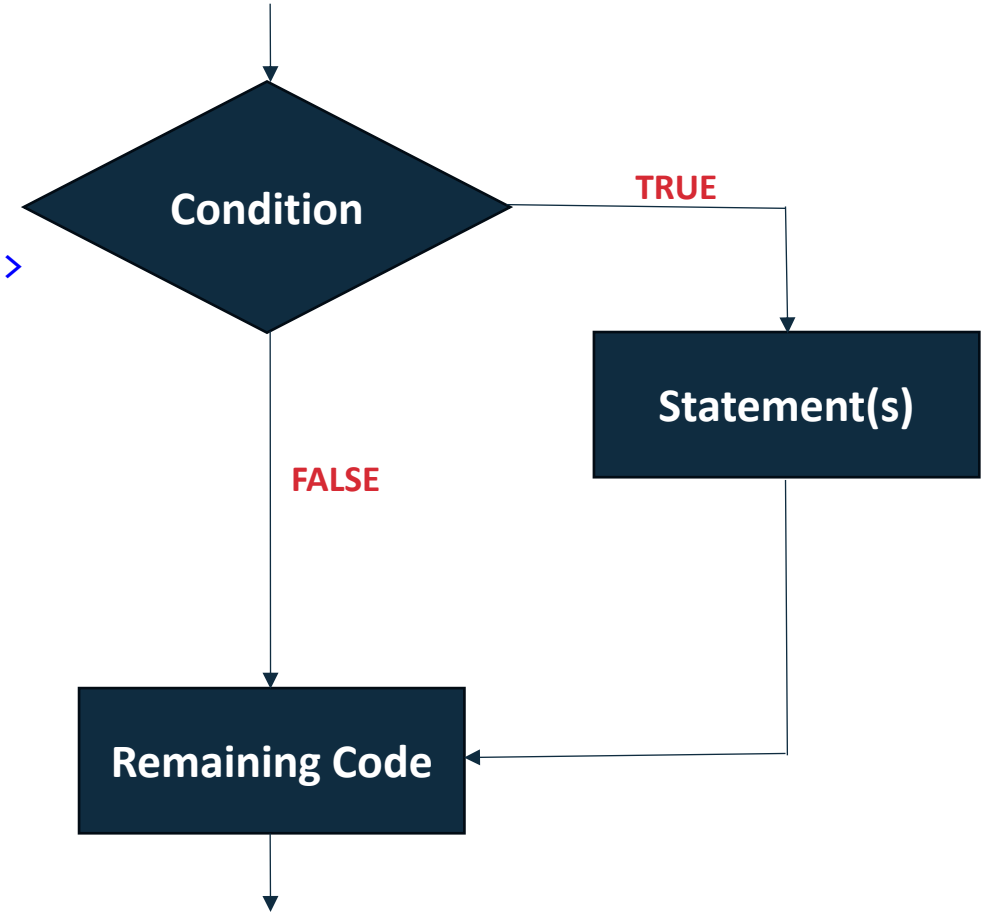


Control Flow

Control Flow - Conditionals

- **<core:if>**
  - Code block executed based on boolean condition

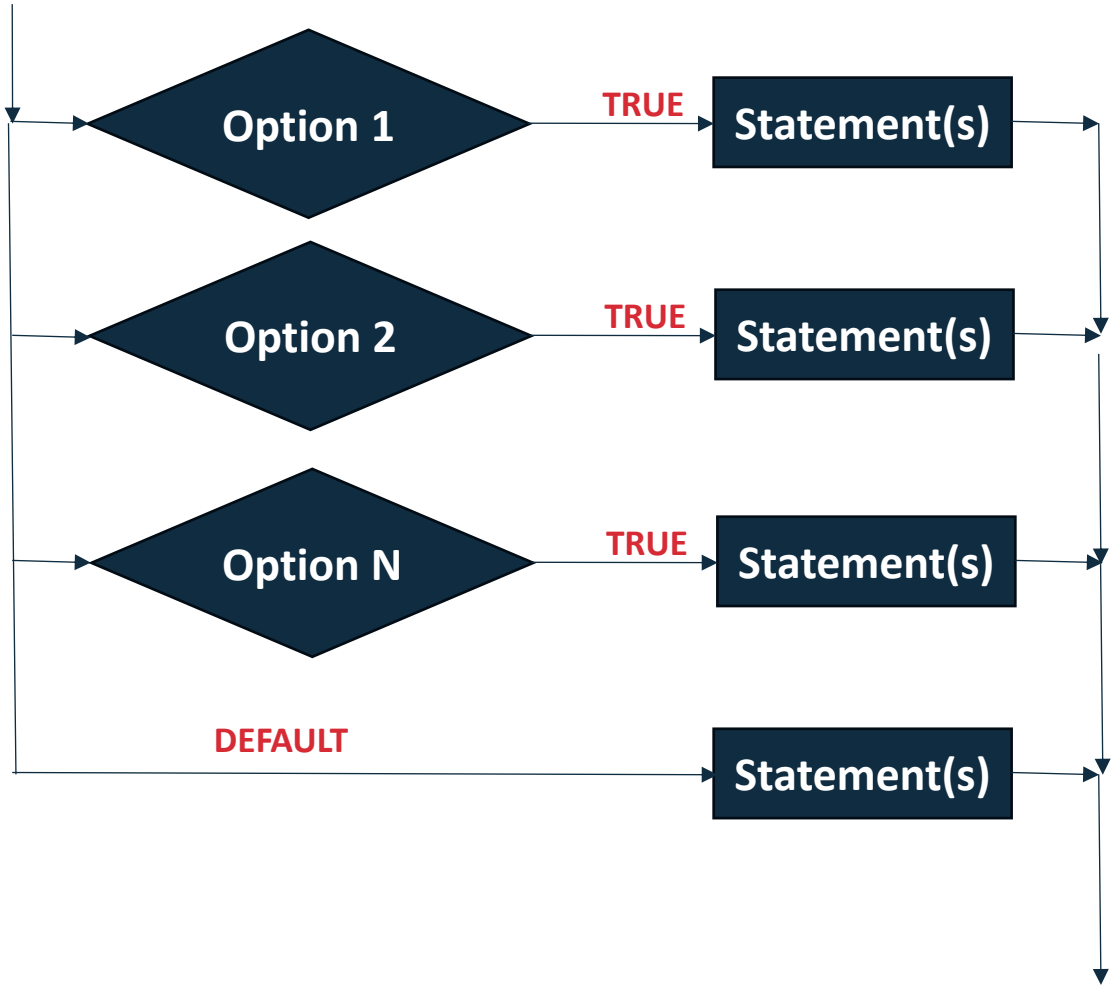
```
<core:if test="${age gt 55 and age lt 200}">
 <gel:log level="INFO">Senior discount applied</gel:log>
</core:if>
```



Control Flow - Conditionals

- **<core:choose>**
  - Conditionally execute one of several code blocks

```
<core:choose>
    <core:when test="${result eq 0}">
        <gel:log>Result equals 0!</gel:log>
    </core:when>
    <core:otherwise>
        <gel:log>Result does not equal 0!</gel:log>
    </core:otherwise>
</core:choose>
```



Control Flow - Loops

- **<core:forEach>**

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

<core:forEach indexVar="i" begin="1" end="3">
    <gel:log>Hello World ${i}!</gel:log>
</core:forEach>
```

Control Flow - Loops

- **<core:while>**

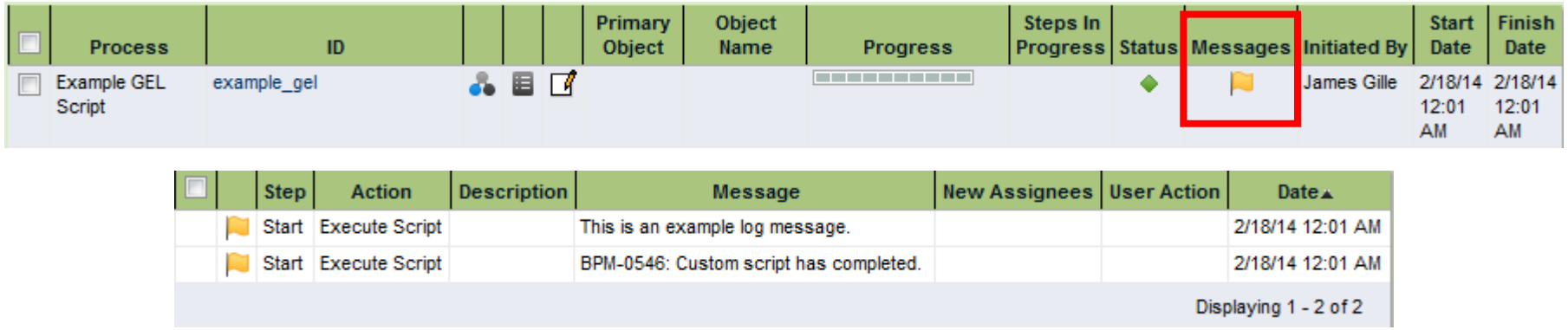
```
<core:while test="${condition eq true}">
…
</core:while>
```

Logging

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>
```



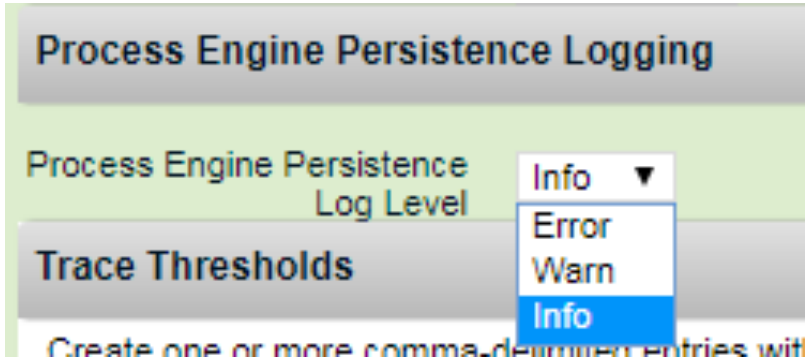
Process	ID	Primary Object	Object Name	Progress	Steps In Progress	Status	Messages	Initiated By	Start Date	Finish Date
Example GEL Script	example_gel							James Gille	2/18/14 12:01 AM	2/18/14 12:01 AM

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 have introduced a new setting that affects whether 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.



Exception Handling

Exception Handling

- Not everything goes as planned.
- You can't control, but you can prevent.
- Proper handling prevents undesired results.
- GEL offers a way to catch errors.

Exception Handling

- **<core:catch>**

```
<core:catch var="genEx">
  <!-- code here -->
</core:catch>

<!-- Check if an exception was caught and log accordingly.  -->
<core:if test="${!empty(genEx)}">
  <gel:log level="ERROR">Exception: ${genEx}</gel:log>
  <core:if test="${!empty(genEx.getCause())}">
    <gel:log level="ERROR">Cause: ${genEx.getCause()}</gel:log>
    <core:if test="${!empty(genEx.getCause().getCause())}">
      <gel:log level="ERROR">Cause of cause: ${genEx.getCause().getCause()}</gel:log>
    </core:if>
  </core:if>
</core:if>
```

Database Operations

Database Operations - Datasources

- **<gel:setDataSource/>**
  - Uses the connection properties from Clarity'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.

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

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:log>Resource Name: ${row.resName}</gel:log>
</core:forEach>

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

<!-- Single Row -->
<core:set var="resource" value="${result.rows[0]}" />
<gel:log>Resource Name: ${resource.resName}</gel:log>
```

- **Note**: When possible, avoid assigning the values from the result set to separate variables. If variables are assigned in a loop repeatedly there have been known bugs for the assignments to stop working eventually and very unexpected behavior to occur.

Database Operations - Bind Variables

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

```
<sql:query dataSource="${clarityDS}" escapeText="0" var="projectQuery">
    <![CDATA[
SELECT  i.code projectCode,
        i.name projectName
FROM    inv_investments i
WHERE   i.id = ?
]]>
    <sql:param value="${gel_objectInstanceId}"/>
</sql:query>
```

- **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 */
```

Database Operations - SQL Updates

- SQL Updates
  - Avoid Insert statements - these are best suited for REST/XOG
  - Direct database updates will not trigger a process to start - REST/XOG updates typically will trigger processes to start
  - Avoid updating OOTB tables when possible - using REST/XOG will ensure all Clarity business rules are followed
  - Generally safe to update custom attributes with SQL - tables beginning with odf_ca_
  - Extra caution should also be used within On-Demand environments
  - Update the last_updated_date and last_updated_by columns
  - **SQL Updates will not be picked up by the instantaneous DWH sync**

Database Operations - SQL Updates

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

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

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

Questions?

regoUniversity

Master Clarity with Rego University

Earn Certifications in Administration, Leadership, and Technical Proficiency

Let Rego be your guide.

Elevate Your Professional Expertise with Rego University Certifications

Rego is excited to continue our **certification programs**, designed to enhance your expertise in Clarity administration, leadership, and technical skills. These certifications provide hands-on experience and knowledge to excel in your career.

Certification Requirements:

- **Completion**: 12 units per certification track
- **Eligibility**: Open to all Rego University attendees

Important Reminder:

To have your certification **credits tracked**, ensure you **complete the class surveys in the app** after each session. This step is critical for certification progress.

Surveys

Please take a few moments to fill out the class survey.
Your feedback is extremely important for future events.

Thank You for Attending Rego University

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 Provider = **Rego Consulting**
- Class Name = **Rego University**
- Course **Description**
- 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@regoconsulting.com

Website
www.regouniversity.com

Continue to Get Resources and Stay Connected

1. Use RegoXchange.com for instructions and how-tos.
2. Talk with your account managers and your Rego consultants.
3. Connect with each other and Clarity experts at RegoGroups.com.
4. Sign up for webinars and join in-person Rego groups near you through at RegoConsulting.com
5. Join us for the next Rego University!