

(Your query will run against this database)

NSQL

```
SELECT @SELECT:DIM:USER_DEF:IMPLIED:FINANCIALS:e.entity:entity@,  
        @SELECT:DIM_PROP:USER_DEF:IMPLIED:FINANCIALS:typ.name:prj_type@,  
        @SELECT:DIM_PROP:USER_DEF:IMPLIED:FINANCIALS:NVL(tc.shortdesc, lu.name):trans_class@,  
        @SELECT:DIM_PROP:USER_DEF:IMPLIED:FINANCIALS:to_char(w.transdate, 'yyyy'):rpt_year@,  
        @SELECT:DIM_PROP:USER_DEF:IMPLIED:FINANCIALS:SUM(NVL(wv.totalcost, 0)):amt@  
  
FROM INV_INVESTMENTS inv  
INNER JOIN ODF_CA_PROJECT p on inv.id = p.id  
INNER JOIN OMN_LOOKUPS_V typ on p.obj_request_type = typ.lookup_code AND typ.lookup_type = 'OBJ_IDEA_PROJECT_TYPE'  
LEFT OUTER JOIN OMN_LOOKUPS_V lu on u.lookup_type = 'OBJ_PORTFOLIO_LOOKUP_TRANSACTION' and lu.language_code = 'en'  
INNER JOIN ppa_wip w on inv.id = w.investment_id  
INNER JOIN ppa_wip_values wv ON w.transno = wv.transno AND wv.currency_type = 'HOME'  
LEFT OUTER JOIN trans_class tc ON tc.transclass = w.transclass  
INNER JOIN ENTITY e ON inv.entity_code = e.entity  
INNER JOIN BIZ_COM_PERIODS b ON e.id = b.entity_id AND w.transdate BETWEEN b.start_date AND (b.end_date - 1) and b.  
WHERE @FILTER@  
AND @WHERE:SECURITY:PROJECT:inv.id@  
GROUP BY e.entity, lu.name, typ.name, tc.shortdesc, to_char(w.transdate, 'yyyy')
```

Hands-on with GEL Scripting, XOG and the REST API

Introductions

Let us introduce ourselves

Clarity PPM Consultant

KRITIKA RANA

2015 - TODAY

About Me

Clarity PPM Consultant with experience in implementations, support & Integrating PPM with other systems.

Active Community Member & CA Community Champion.



Agenda

Hands-on with GEL Scripting, XOG and the REST API

Session Agenda

Hands-on with GEL Scripting, XOG and the REST API



XOG

REST APIs

GEL Scripting

XOG

Hands-on with GEL Scripting, XOG and the REST API

XML OPEN GATEWAY

Session Outline

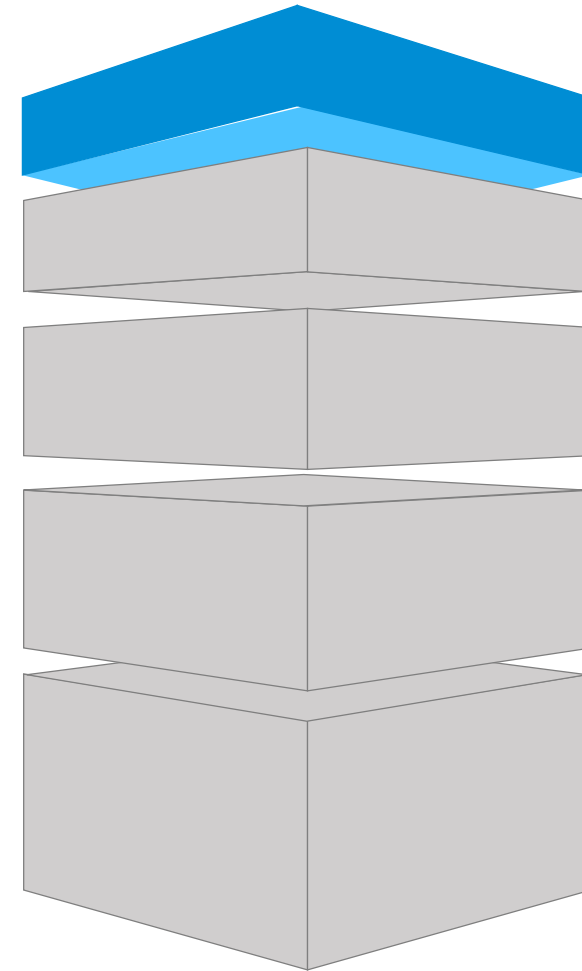
- 01 Introduction**
- 02 Working with XOG**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise**



XML OPEN GATEWAY

Session Outline

- 01 Introduction**
- 02 Working with XOG
- 03 Limitations
- 04 Best Practices
- 05 Hands On Exercise



What is XOG?

Basic Information

- Supported Web service interface
- Available since Niku6.0 onwards
- Exchange information with other applications using
 - Extensible Markup Language (XML)
 - Simple Object Access Protocol (SOAP)
 - Web Services Description Language (WSDL)

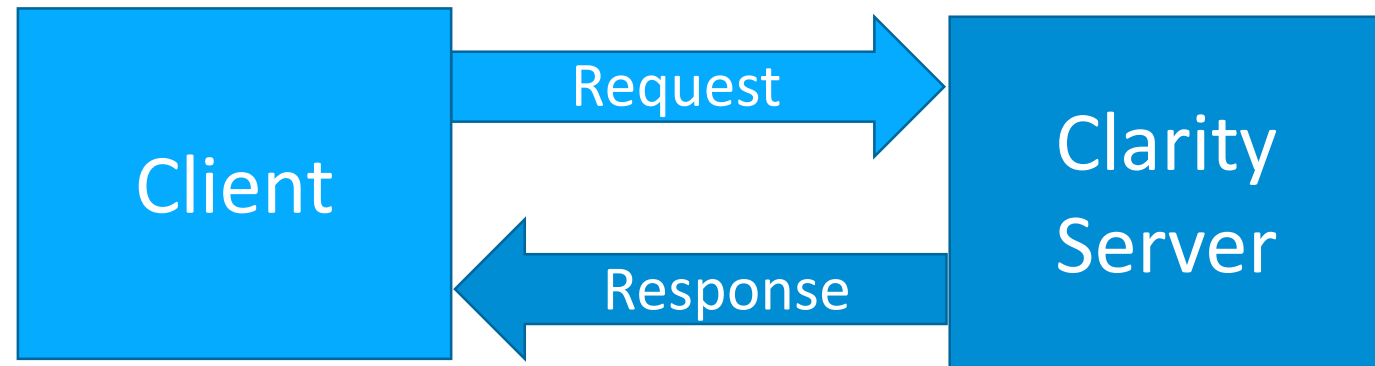
Why XOG?

- Uses industry standard Web services
- Supported Mechanism
- Capable of moving Data as well as configuration
- Secure and rights-enabled
- Typically upward compatible, making upgrades easier
- Can be used by any programming language that support SOAP
- Extensively used within Clarity workflows to update data

How XOG Works

Architecture diagram

- Client makes a SOAP call to Clarity Server
- Clarity Server processes the request
- Sends a SOAP response back to the client



- ↓
1. Installed Client
 2. Browser XOG Client
 3. External Applications
 4. Programming Languages

Sample XOG Requests

Read Request

- Read request gives us filters to be used to get the required data.

```
<NikuDataBus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../xsd/nikuxog_read.xsd">
  <Header version="6.0.11" action="read" objectType="project" externalSource="NIKU">
    <args name="order_by_1" value="name"/>
    <args name="order_by_2" value="projectID"/>
    <args name="include_tasks" value="true"/>
    <args name="include_dependencies" value="true"/>
    <args name="include_subprojects" value="true"/>
    <args name="include_resources" value="true"/>
    <args name="include_baselines" value="true"/>
    <args name="include_allocations" value="true"/>
    <args name="include_estimates" value="true"/>
    <args name="include_actuals" value="true"/>
    <args name="include_custom" value="true"/>
    <args name="include_burdening" value="false"/>
  </Header>
  <Query>
    <Filter name="projectID" criteria="EQUALS">test</Filter>
  </Query>
</NikuDataBus>
```

Sample XOG Requests

Write Request

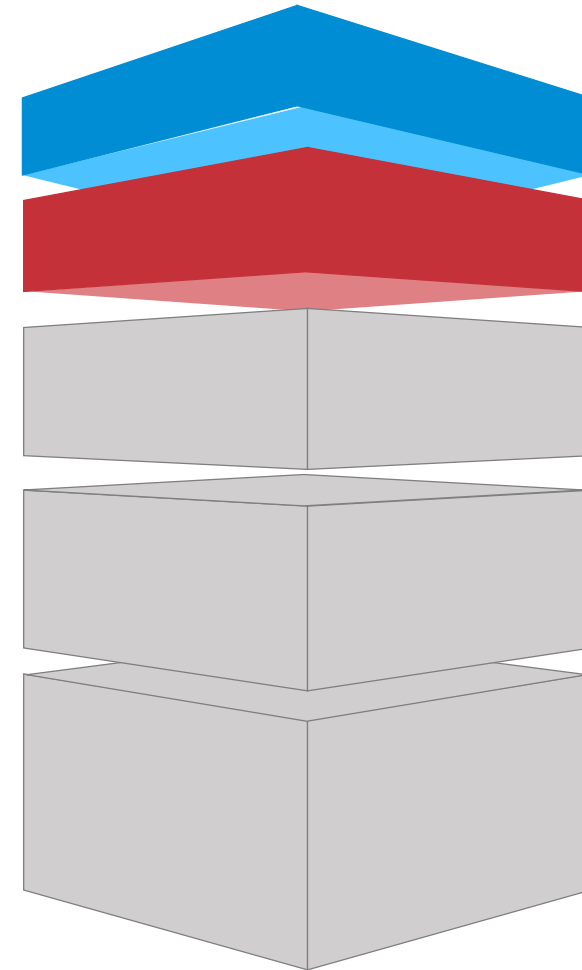
- Write request provide us with the ability to modify the data in Clarity.

```
<NikuDataBus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../xsd/nikuxog_project.xsd">
  <Header version="6.0.11" action="write" objectType="project" externalSource="NIKU"/>
  <Projects>
    <Project name="XXAproject" projectID="80AA" description="XXAProject"
      managerResourceID="admin" start="1999-01-01T00:00:00" finish="2002-01-01T00:00:00"
      active="1" openForTimeEntry="true" trackMode="2" format="0" >
      <Resources>
        <Resource resourceID="joeTime"/>
      </Resources>
      <CustomInformation>
        <ColumnValue name="PROJECT_RISK">Medium</ColumnValue>
      </CustomInformation>
      <General addedBy="admin" addedDate="2003-01-01"/>
    </Project>
  </Projects>
</NikuDataBus>
```

XML OPEN GATEWAY

Session Outline

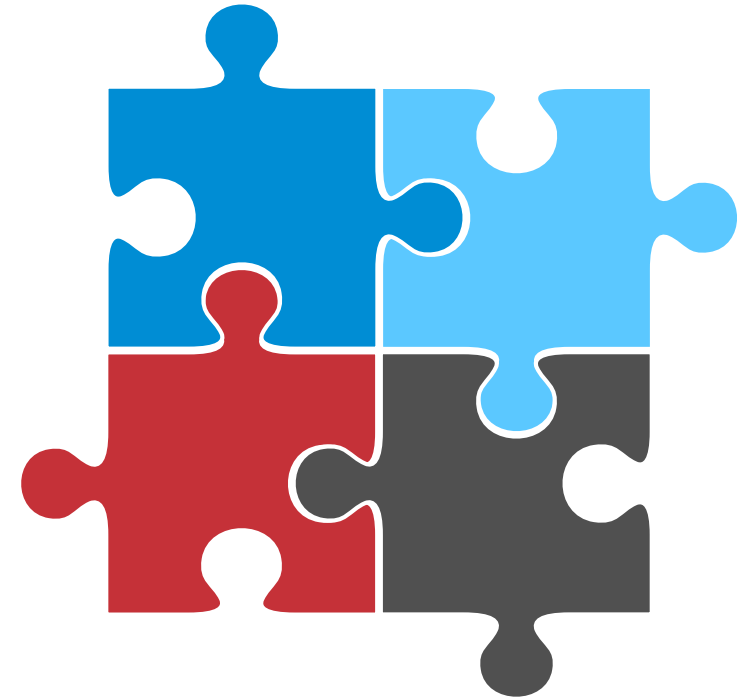
- 01 Introduction**
- 02 Working with XOG**
- 03 Limitations
- 04 Best Practices
- 05 Hands On Exercise



Working with XOG

Multiple ways to use XOG

- Browser XOG
- Command Line XOG
 - Using XOG commands
 - Using Properties file
- From external applications
- Programming Languages
 - GEL, JAVA, C# etc.



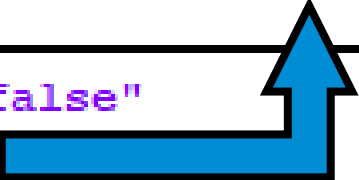
Browser XOG

Enabling browser XOG

- Navigate to the file path to enable browser XOG
[Clarity-Home]\META-INF\xog\wmd
- Take a backup of existing **xog.xml** file
- Amend **xog.xml** as in screen shot
- Flush the caches using security.caches or restart services
- Then try :
<http://server/niku/nu#action:xog.client>

`active="true"`

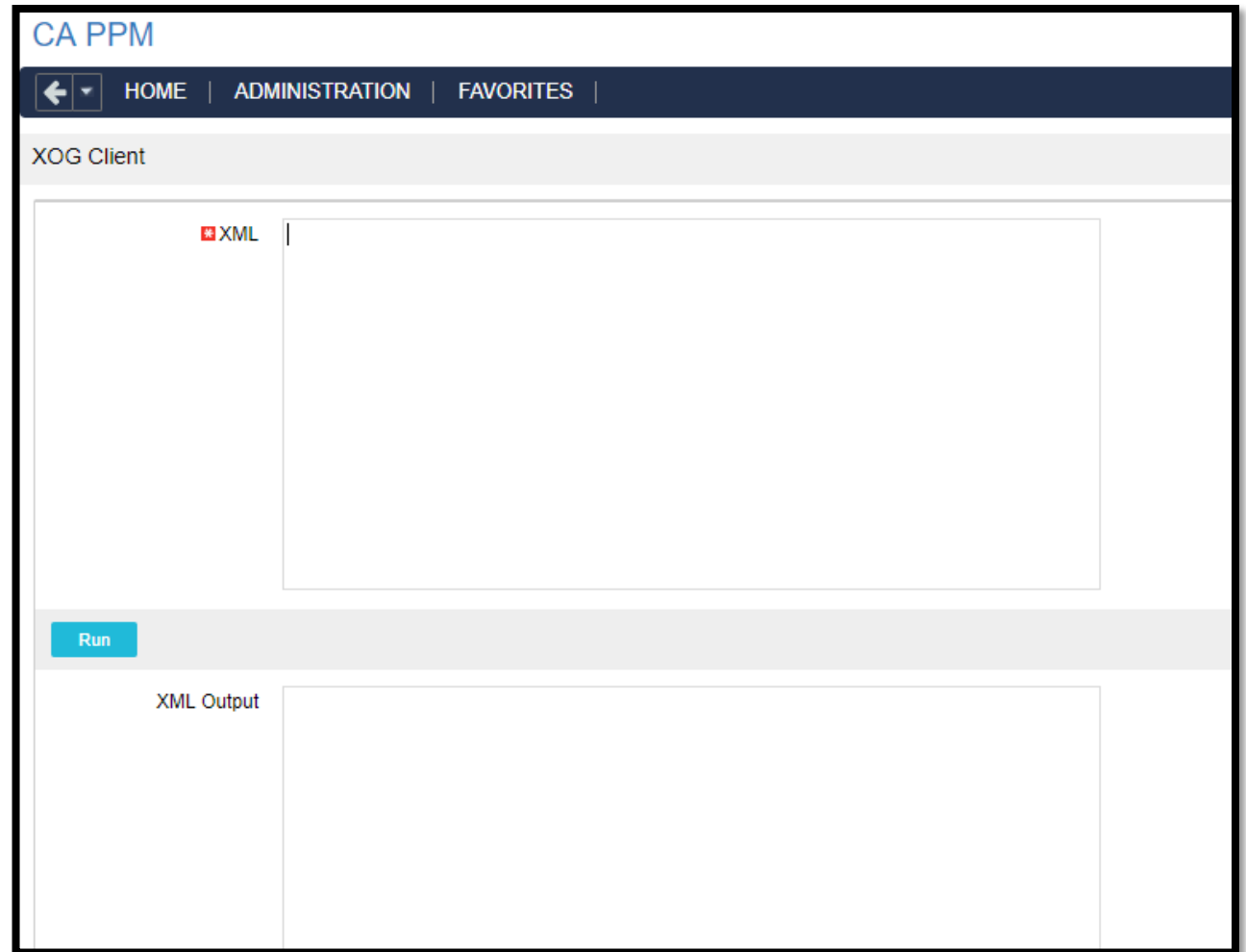
```
<pageAction id="xog.client" useSsl="false"
policyId="xogPerm" active="false">
<retrieveService type="xbl" componentId="xog"
description="getOutput.xbl" portletId="xog.client"/>
</pageAction>
```



Browser XOG

Browser Window

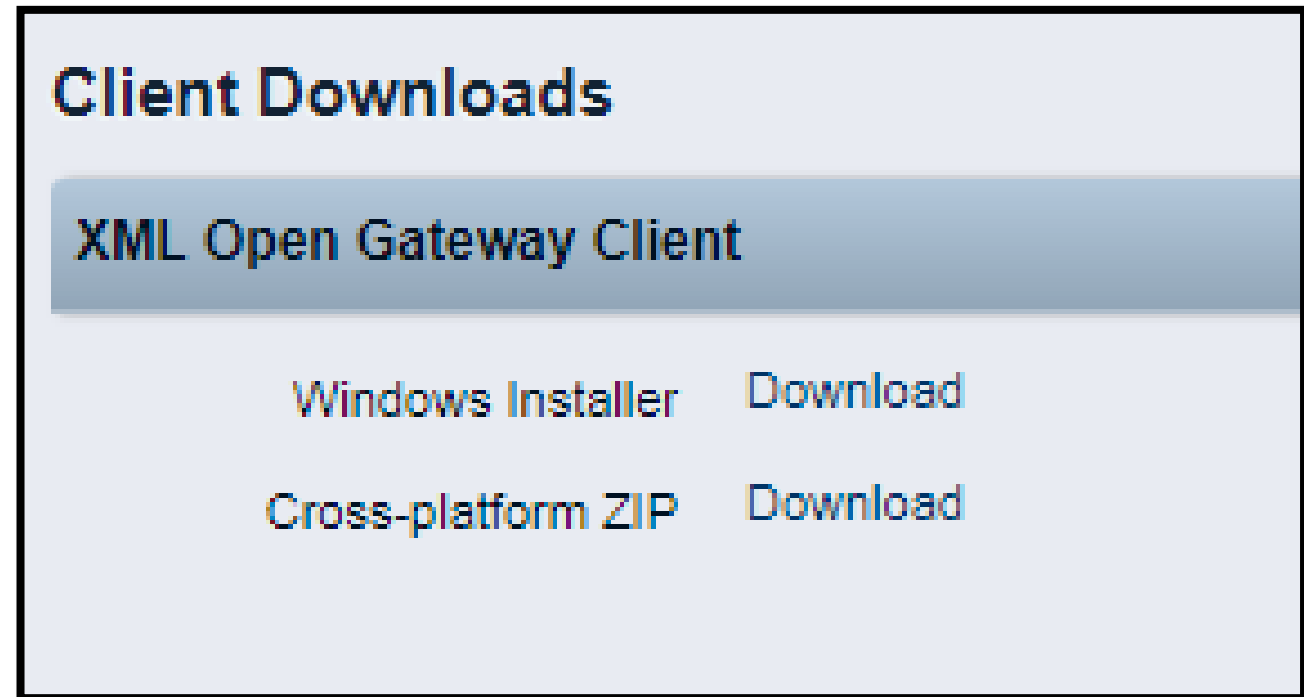
- Visible at server URL
<http://server/niku/nu#action:xog.client>
- XML field for XOG request
- XML Output is the response from Clarity Server



Command Line XOG

Download and Install XOG Client

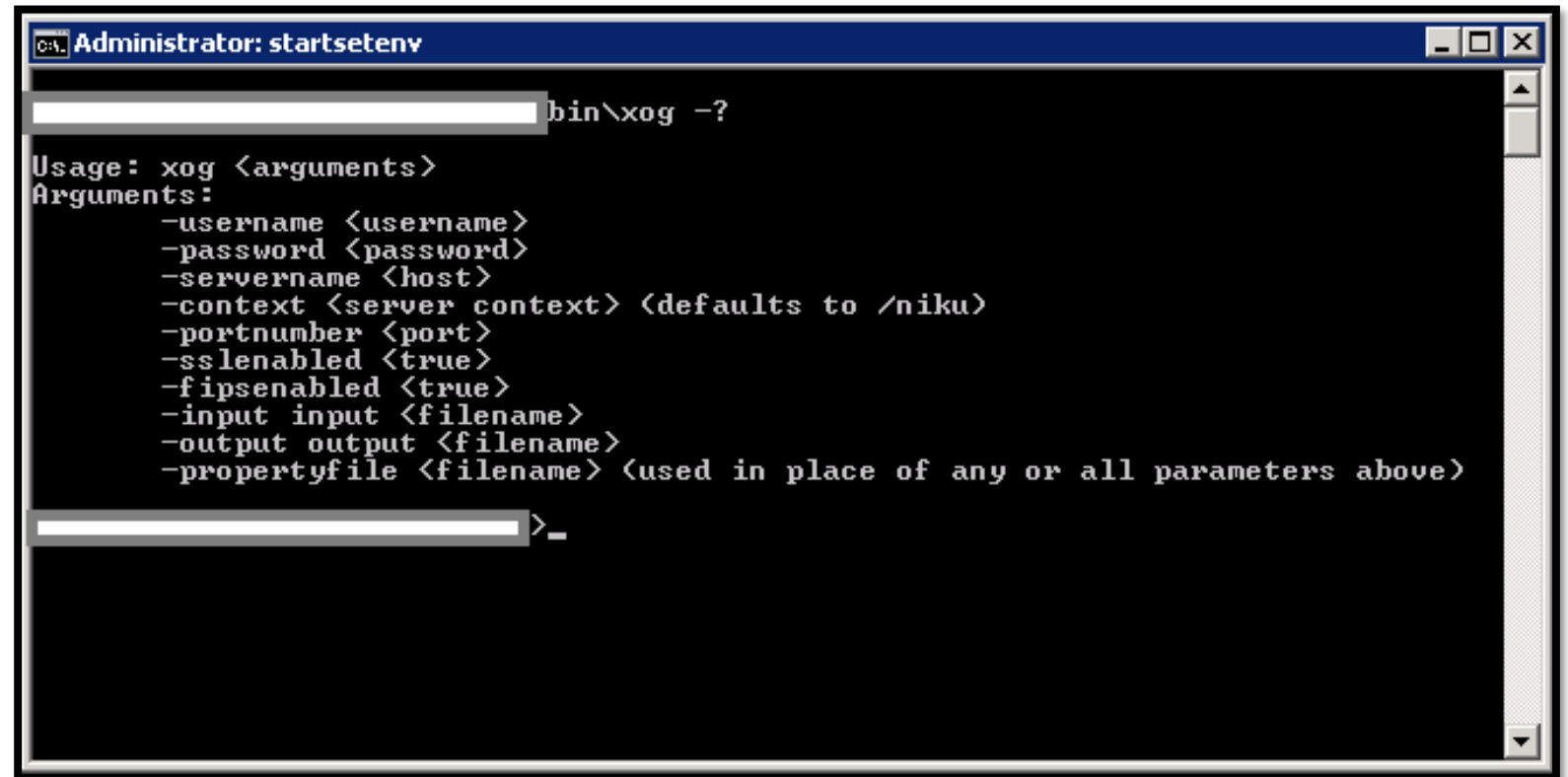
- Packaged within Clarity
- Navigate to “Administration” menu > “Client Downloads”
 - “Windows Installer” for Windows OS
 - “Cross-Platform Zip” for Windows or Linux /UNIX operating systems.



Command Line XOG

Using Command Line Parameters

- Navigate to the Clarity_home\bin folder.
- Type the required XOG command.
- To see the command usage, issue the following command:
 - bin\xog -?



```
C:\Administrator: startsetenv
bin\xog -?

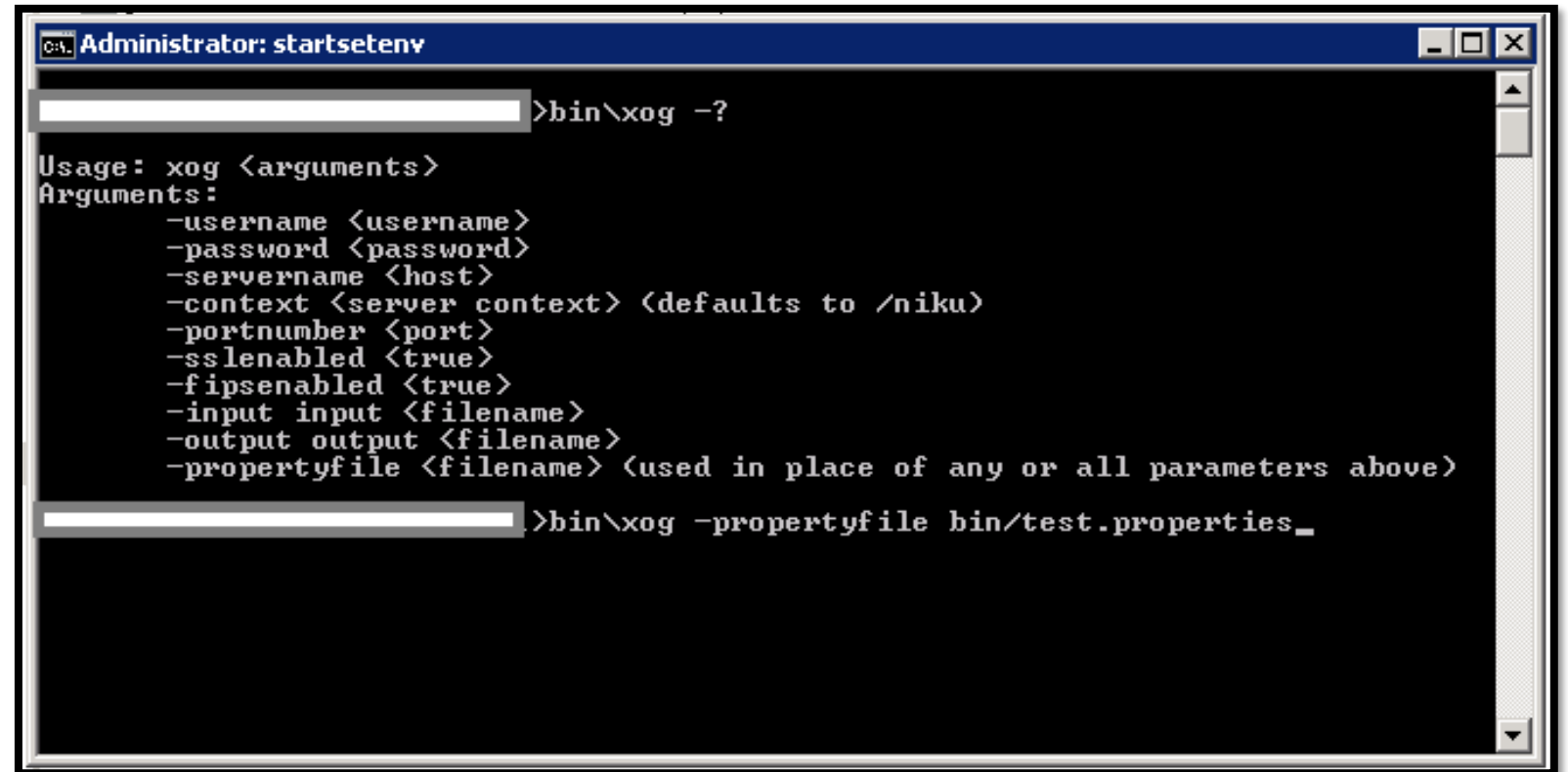
Usage: xog <arguments>
Arguments:
  -username <username>
  -password <password>
  -servername <host>
  -context <server context> (defaults to /niku)
  -portnumber <port>
  -sslenabled <true>
  -fipsenabled <true>
  -input input <filename>
  -output output <filename>
  -propertyfile <filename> (used in place of any or all parameters above)

>_
```

Command Line XOG

Using Properties File

- Modify the test.properties file.
- Location:
Clarity_Home\xog-unzipped\bin
- Another option is to make your own .properties file and store it in the bin directory.
- At the XOG prompt(Clarity_home\bin) issue the following command
 - bin\xog -propertyfile bin/test.properties
- View the output



```
C:\>bin\xog -?

Usage: xog <arguments>
Arguments:
  -username <username>
  -password <password>
  -servername <host>
  -context <server context> (defaults to /niku)
  -portnumber <port>
  -sslenabled <true>
  -fipsenabled <true>
  -input input <filename>
  -output output <filename>
  -propertyfile <filename> (used in place of any or all parameters above)

C:\>bin\xog -propertyfile bin/test.properties_
```

Command Line XOG

Sample Properties File

```
# --- server host name you want to test against
servername=

portnumber=

#default port number for ssl
#portnumber=443

#set to true if running against a SSL enabled server
sslenabled=false

#set to true if running against a SSL enabled server in FIPS 140-2 mode
fipsenabled=false

output=out.xml
username=
password=

# --- leave the one you want to test un-commented and comment out all other input entries
#input=../xml/benefitPlan_read.xml
#input=../xml/benefitPlan_write.xml
#input=../xml/big_companies_read.xml
```

Other ways to use XOG

External Applications and Programming Languages

- Any application that is capable of making a soap call, can use XOG to interact with Clarity Server.

External Applications

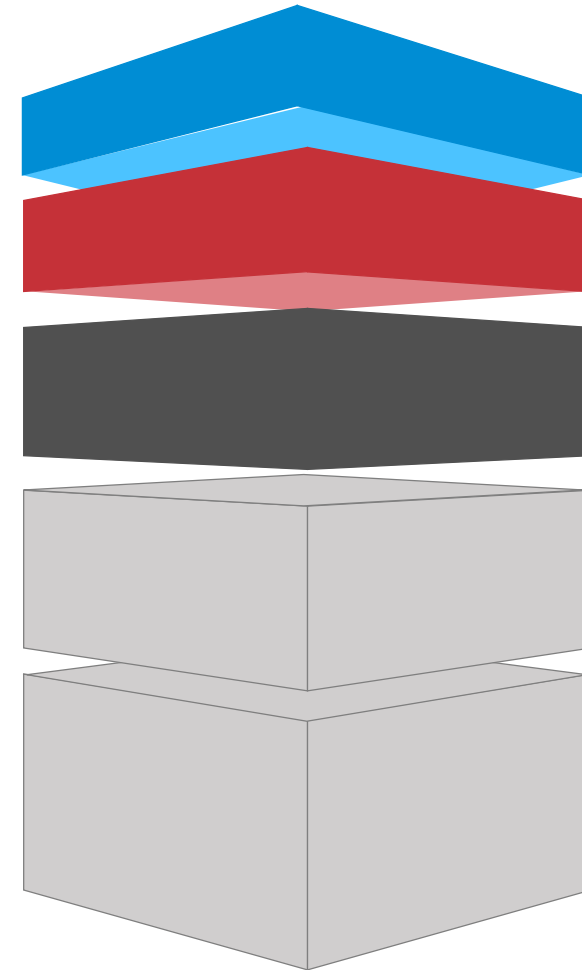
- Various programming languages can be used to form a SOAP request to use XOG.
- Examples:
 - JAVA
 - C++
 - C#
 - Visual Basic
 - Perl
 - GEL(Mostly Used)

Programming Languages

XML OPEN GATEWAY

Session Outline

- 01 Introduction**
- 02 Working with XOG**
- 03 Limitations**
- 04 Best Practices
- 05 Hands On Exercise



Limitations

Data Deletion

- Deletion is not possible
- Complete="true" tags available in some cases, but these are cumbersome
 - OBS Associations,
 - Skill Associations,
 - Group Assignments,
 - Global Rights,
 - Instance Rights,
 - OBS Rights and Rate Matrices

Limitations

Limited Granularity

- Limited options for granularity
 - To XOG-out project team, XOG-out of entire project is needed.
 - Similarly, To XOG-in a task, it must be part of a project XOG write request.

Limitations

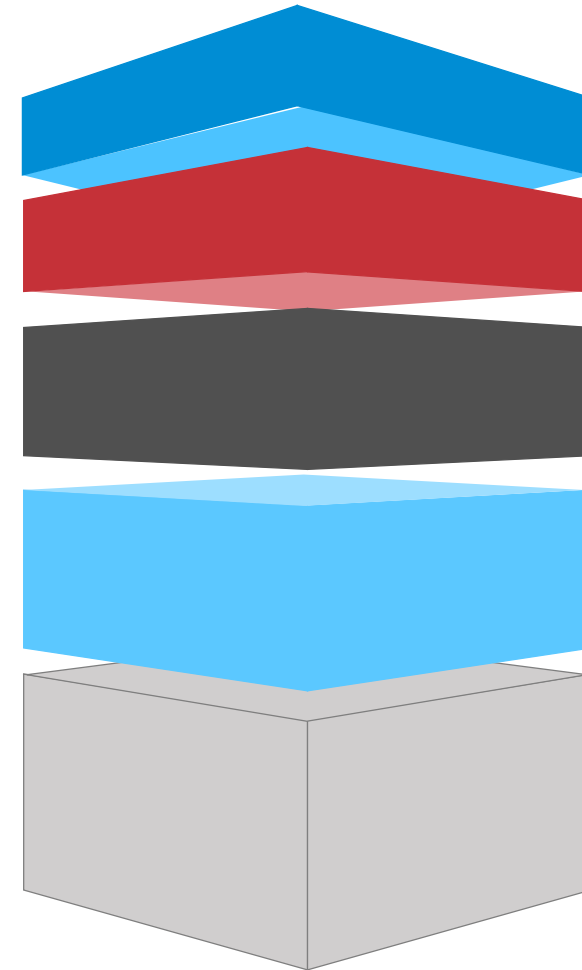
Performance

- Performance of XOG requests is exponentially affected by data volumes imported or exported
 - Large XOG requests can take hours to process, and very large requests can time-out the XOG session.
 - Although, XOG in newer versions paginates the number of records processed, reducing the possibility of timing out, but this requires special logic in the code to handle pagination.

XML OPEN GATEWAY

Session Outline

- 01 Introduction**
- 02 Working with XOG**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise



Best Practices

- Keep XOG requests as small as possible,
 - Many smaller requests will usually execute faster than one large request
- XOG-read only what is necessary
 - Use Filters while querying data
- Remove unnecessary tags while XOG-write
- XOG is very powerful, Be absolutely sure on what you are updating in the system.



Make sure to specify an external ID when defining project tasks. This provides a cleaner project XOG, especially when dealing with assignments.

XML OPEN GATEWAY

Session Outline

- 01 Introduction**
- 02 Working with XOG**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise**



Hands on Exercise



Summarize XOG

Let's take a look back on XOG session

Introduction

What is XOG

Why XOG

How XOG works

Sample XOG Requests



Summarize XOG

Let's take a look back on XOG session



Working with XOG

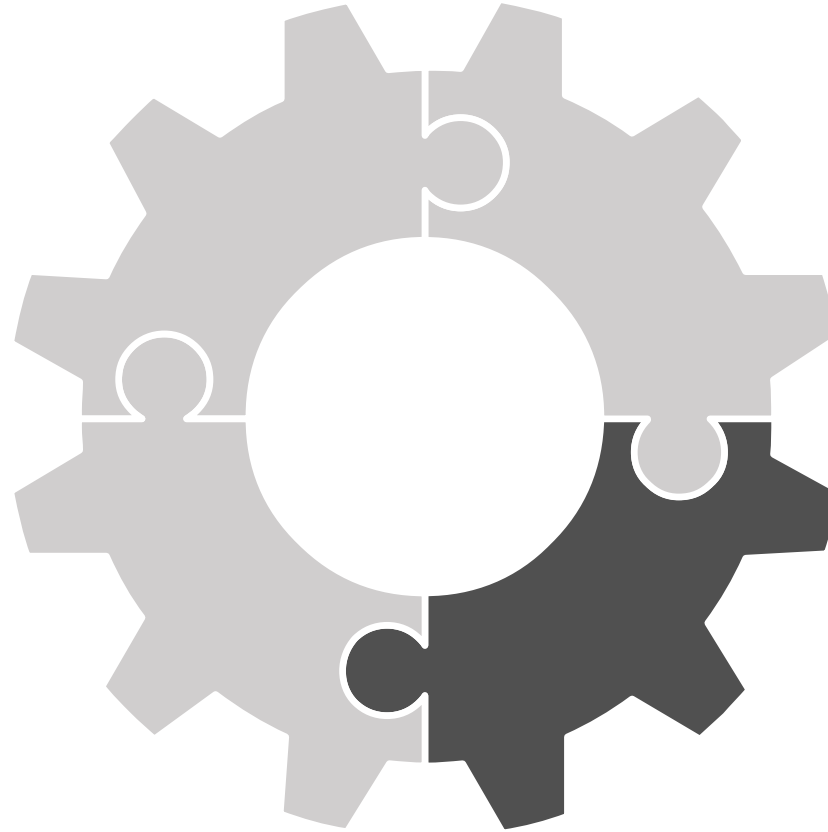
Browser XOG

Command Line XOG

Other ways to use XOG

Summarize XOG

Let's take a look back on XOG session



Limitations

- Data Deletion
- Limited Granularity
- Performance

Summarize XOG

Let's take a look back on XOG session



Best Practices

Small requests

XOG-read only what is necessary

Remove unnecessary tags

Summarize XOG

Let's take a look back on XOG session

Introduction

- What is XOG
- Why XOG
- How XOG works
- Sample XOG Requests

Working with XOG

- Browser XOG
- Command Line XOG
- Other ways to use XOG



Best Practices

- Small requests
- XOG-read only what is necessary
- Remove unnecessary tags

Limitations

- Data Deletion
- Limited Granularity
- Performance

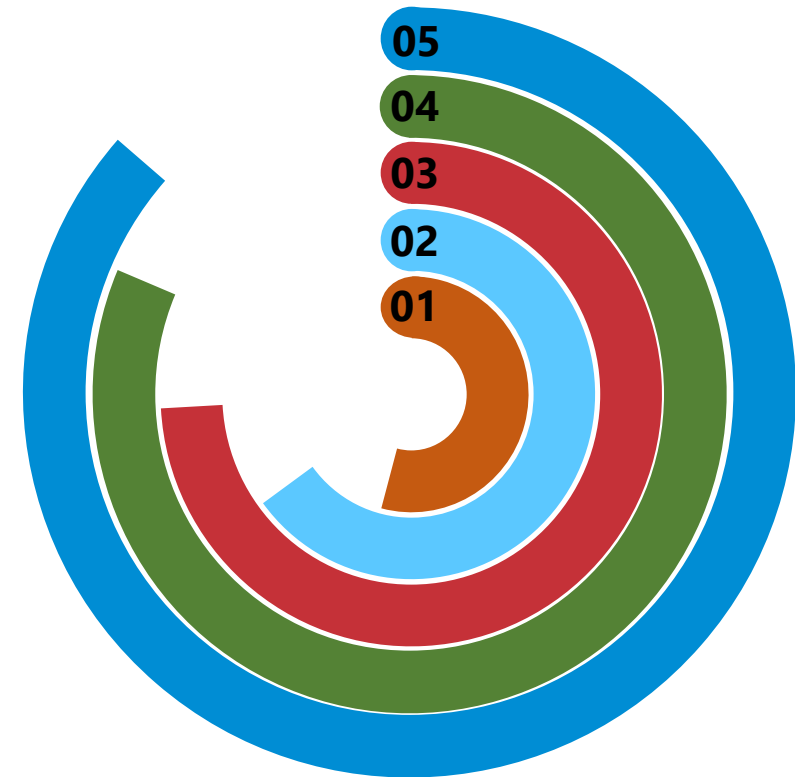
REST APIs

Hands-on with GEL Scripting, XOG and the REST API

REST APIs

Session Outline

- 01 Introduction**
- 02 Working with REST APIs**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise**



REST APIs

Session Outline

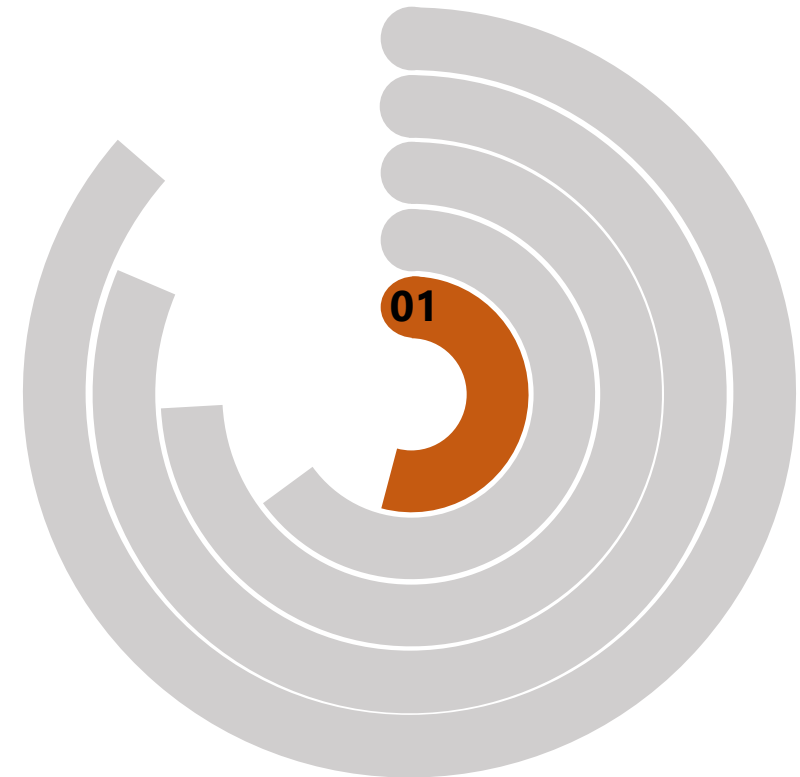
01 Introduction

02 Working with REST APIs

03 Limitations

04 Best Practices

05 Hands On Exercise



What is API

Getting Started

- API - Application Programming Interface
- A set of functions and procedures allowing the creation of applications
- There are two types of APIs
 - SOAP
 - REST

What are Rest APIs

Rest APIs Explained

- Based on representational state transfer (REST) technology, an architectural style
- Uses HTTP requests to GET, PUT, POST and DELETE data
- REST leverages less bandwidth, more suitable for internet usage.
- Breaks down a transaction to create a series of small modules



The Rest API documentation URL for Clarity can be found in System options → API → API Documentation URL.

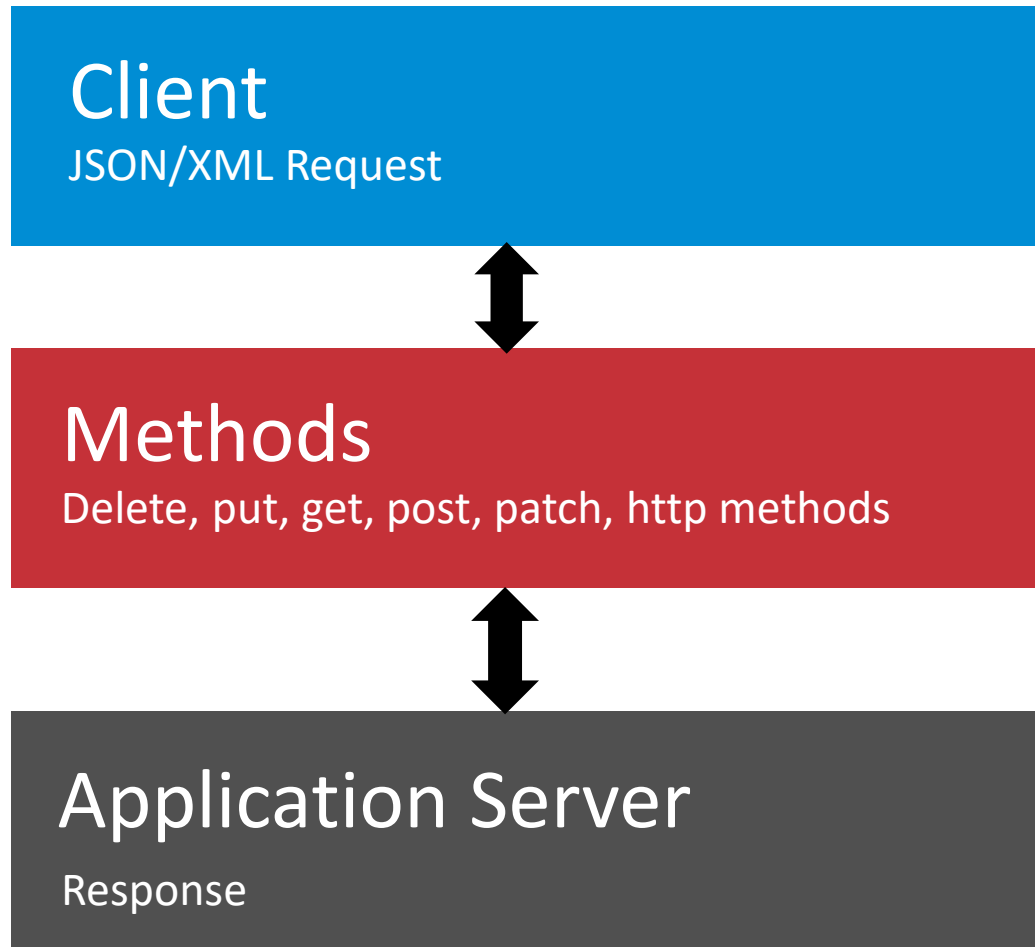
Why Rest APIs

What makes the Rest APIs so popular

- Separation between the client and the server
- Visibility, reliability and scalability
- Platform Independent
- Variety of Data formats
- Superior Performance

How REST API Works

Architecture diagram



- Client makes a REST call (JSON/XML) to Server.
- Clarity Server processes the request.
- Sends a response back to the client.
- In Clarity PPM, the responses are in JSON format.

Sample REST Requests

GET Request

- GET request is the read request for REST

```
{
  "_pageSize": 0,
  "_self": "string",
  "_totalCount": 0,
  "_results": [
    {
      "_internalId": 5000000,
      "_self": "string"
    }
  ],
  "_recordsReturned": 0
}
```

Sample REST Requests

PATCH Request

- PUT and PATCH requests allow us to modify data in Clarity.

```
{
  "actuals": {
    "dataType": "numeric",
    "_type": "tsv",
    "segmentList": {
      "segments": [
        {
          "start": "2019-01-15T00:00:00",
          "finish": "2019-01-16T00:00:00",
          "value": 36000
        }
      ]
    }
  },
  "taskId": 5000003
}
```

REST APIs

Session Outline

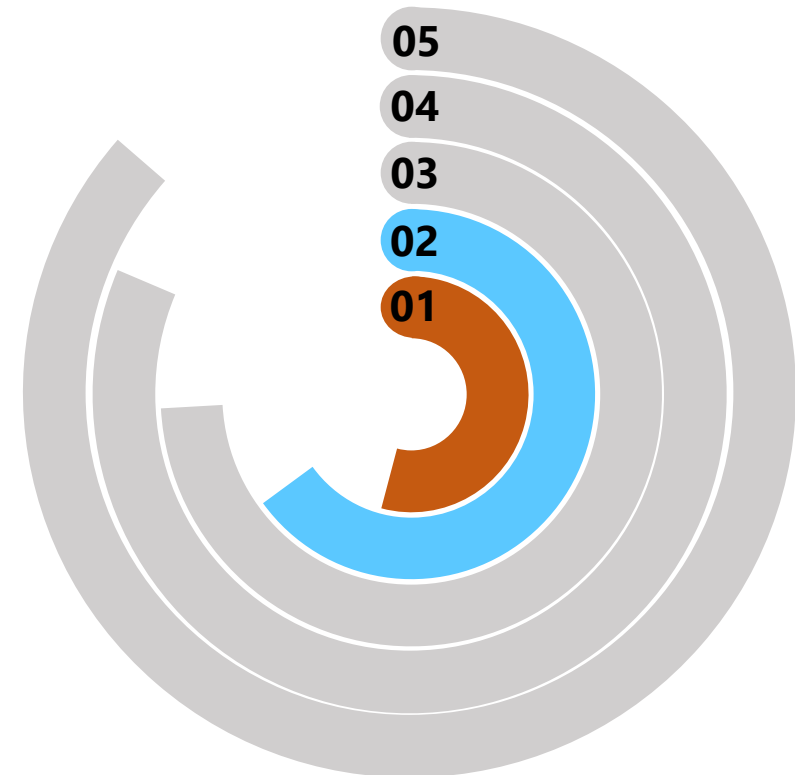
01 Introduction

02 Working with REST APIs

03 Limitations

04 Best Practices

05 Hands On Exercise



Working with Rest APIs

Demonstrate Rest APIs

- Rest API URL is : <https://lab1.pemari.com/niku/rest/describe/index.html>

CA PPM

username password

CA PPM REST API

The CA PPM REST APIs can only be used by CA PPM engineering. At this time, the REST APIs are **not** supported for customer d

Our strategy is to focus on developing robust APIs as we design our new user experience. The APIs may change as we make archi
improvements, add capabilities and optimize performance. We will review our strategy every release and make the APIs publicly a
soon as possible.

Authorization Show/Hide | List Operations | Exp

Project : Projects, Tasks, Teams, Assignments, Resources, Status Reports Show/Hide | List Operations | Exp

Supporting : Supporting Resources Show/Hide | List Operations | Exp

Timesheet : Timesheets, Time Entries, Time Periods, Notes Show/Hide | List Operations | Exp

[BASE URL: /v1 , API VERSION: 1.0]

Working with Rest APIs

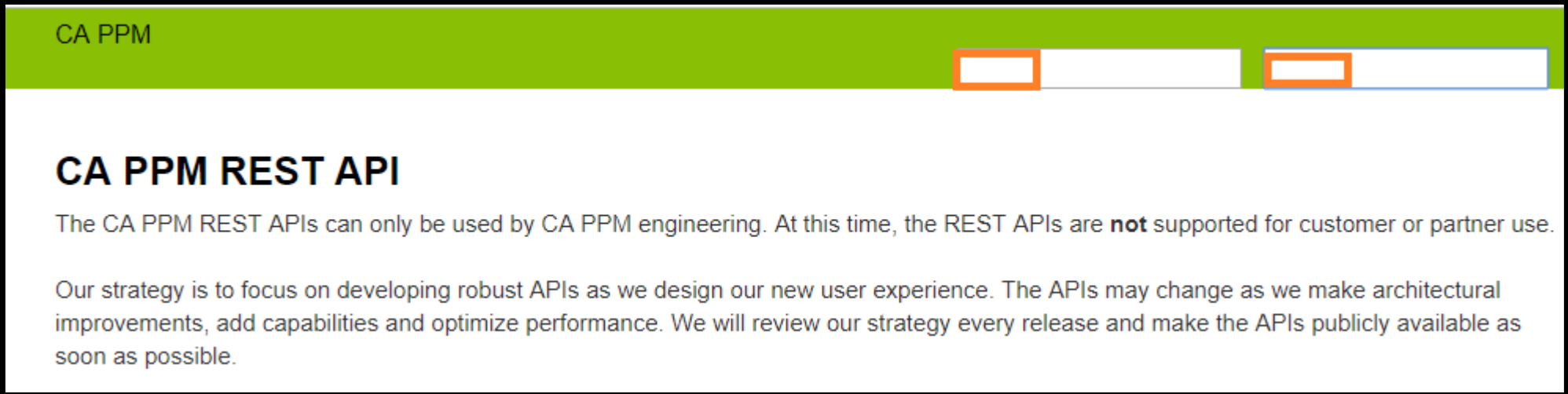
Steps

- Authenticate & Connect
- Set headers
- Form and send the request
- Process the Output

Authenticate & Connect

Authenticate the user to send a request

- Sample of Authentication
 - The username & password are the same username and password used for Clarity Login.
- URL for the request can be obtained from the Clarity Rest API Documentation.



CA PPM

CA PPM REST API

The CA PPM REST APIs can only be used by CA PPM engineering. At this time, the REST APIs are **not** supported for customer or partner use.

Our strategy is to focus on developing robust APIs as we design our new user experience. The APIs may change as we make architectural improvements, add capabilities and optimize performance. We will review our strategy every release and make the APIs publicly available as soon as possible.

STEP TWO

Set Headers

Setting the type of request

- We can set in the headers the type of request and response

The screenshot displays an API endpoint configuration. At the top, a blue bar contains the HTTP method 'GET' and the path '/projects/{projectsInternalId}'. Below this, the 'Implementation Notes' section states: 'Returns a fully materialized resource {projectsInternalId}'. The 'Response Class (Status 200)' section indicates a 'Successful operation'. A table with two columns, 'Model' and 'Example Value', shows a JSON object:

```
{  "isOpenForTimeEntry": "true",  "projectType": {    "displayValue": "string",    "id": "string"  },  "agileWSJFscore": 0,  "psaCategory": {    "displayValue": "string",    "id": "string"  }}
```

 At the bottom, a dropdown menu for 'Response Content Type' is set to 'application/json'.

STEP THREE

Form and Send the Request

Form the Request

- The REST request in JSON format.

The screenshot displays a REST client interface. At the top, it shows a "Response Class (Status 200)" with the message "Successful operation". Below this, there is a "Model" tab with an "Example Value" field containing a JSON object:

```
{
  "agileWSJFScore": 0,
  "psaCategory": {
    "displayValue": "string",
    "id": "string"
  },
  "issues": {
    "_self": "string"
  },
  "objective": "string",
  "bpPrjAccountManager": {
    "id": "string"
  }
}
```

Below the JSON, the "Response Content Type" is set to "application/json".

The "Parameters" section is highlighted with an orange border. It contains a table with the following structure:

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Operation will add this projects.	body	Model Example Value

The "Value" field for the "body" parameter is a text area containing "(required)". The "Data Type" column for the "body" parameter has a sub-table with an "Example Value" field containing a JSON object:

```
{
  "isOpenForTimeEntry": "true",
  "projectType": "string",
  "agileWSJFScore": 0,
  "psaCategory": "string",
  "objective": "string",
  "bpPrjAccountManager": {
    "id": "string"
  }
}
```

At the bottom, the "Parameter content type" is set to "application/json".

STEP THREE

Form and Send the Request

Send the Request

- The REST request in JSON format.

Response Content Type

Parameters

Parameter	Value	Description
projectsInternalId	<input type="text" value="(required)"/>	Internal
fields	<input type="text" value="Provide multiple values in new lines."/>	This det will be p Separat Example fields=n

STEP FOUR

- The Response window shows the output for the request sent.

Process the Output

Checking the output for further processing

The screenshot displays a REST client interface with the following sections:

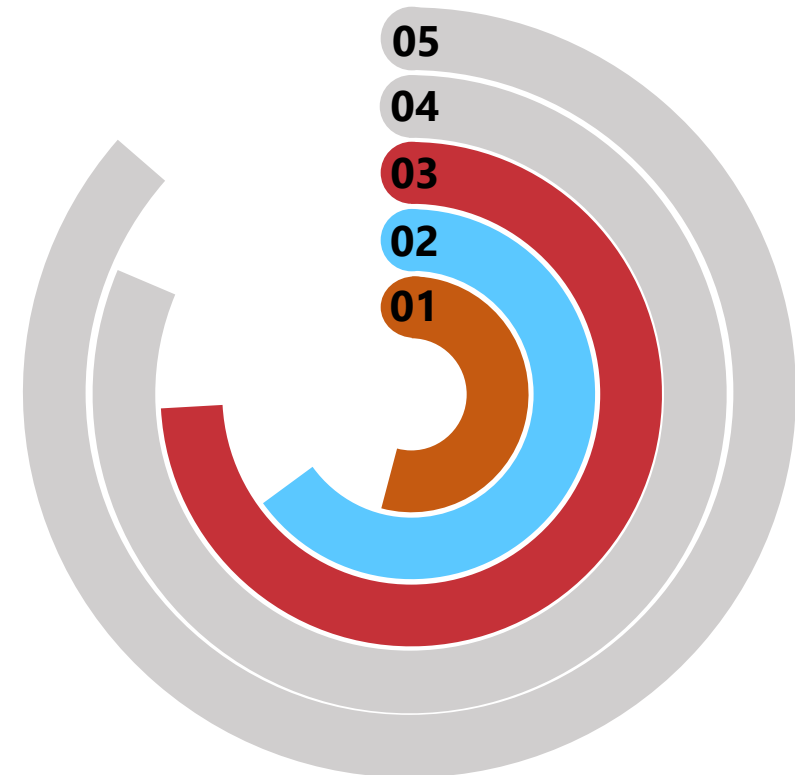
- Request URL:** `https://lab1.pemari.com/ppm/rest/v1/projects/5049000`
- Response Body:** A JSON object representing a project, with a portion highlighted in an orange box:

```
{
  "isOpenForTimeEntry": true,
  "agileWSJFScore": null,
  "projectType": {
    "displayValue": "Application Change",
    "_type": "lookup",
    "id": "type200"
  },
  "psaCategory": null,
  "issues": {
    "_self": "https://lab1.pemari.com/ppm/rest/v1/projects/5049000/issues"
  },
  "objective": null,
  "bpPrjAccountManager": null,
  "npdImprovesCompAdvantage": null,
  "npmTechFeasComp": null,
  "npmRiskAnalysisDeliverable": null,
  "npmLeanCanvasComplete": null,
```
- Response Code:** 200

REST APIs

Session Outline

- 01 Introduction**
- 02 Working with REST APIs**
- 03 Limitations**
- 04 Best Practices
- 05 Hands On Exercise



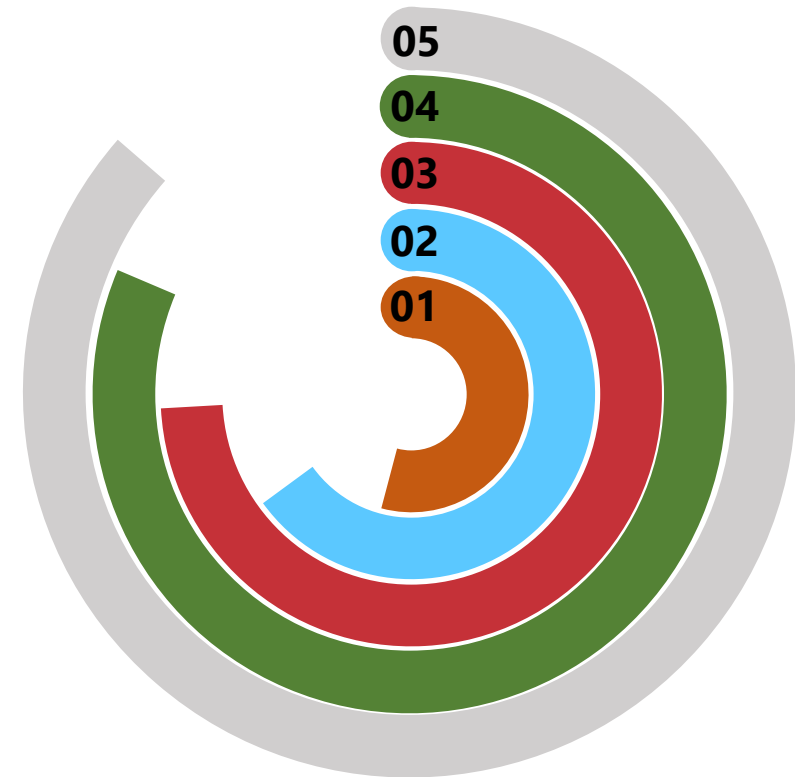
Limitations

- Stateless APIs increase the latency in request processing times and bandwidth usage
 - The reason is that the client is sending all messages with redundant information.
- Limited options for processing the requests
 - Rest APIs use the HTTP methods for processing data.
 - Typically, the protocol has a limited number of methods to operate upon the data.
- HTTP combines application-level and transport-level status codes
 - e.g., 304 Not Modified and 400 Bad Request are the HTTP codes
 - 407 Proxy Authentication Required and 502 Bad Gateway are transport level status codes
- Broadcom has not made the REST APIs available yet for public consumption.

REST APIs

Session Outline

- 01 Introduction**
- 02 Working with REST APIs**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise



Best Practices

Clean Request

- Keep requests clean by using Nouns
 - For an easy understanding use this structure for every resource
- Use HATEOAS
 - **Hypermedia As The Engine Of Application State** is a principle that hypertext links should be used to create a better navigation through the API.



REST APIs is planned to be available for customer use from the next release 15.6.1.

Best Practices

Error Handling

- Handle errors with HTTP error codes
 - 200 – OK – Everything is working
 - 201 – OK – New resource has been created
 - 204 – OK – The resource was successfully deleted
 - 304 – Not Modified – The client can use cached data
 - 400 – Bad Request – The request was invalid or cannot be served.
 - 401 – Unauthorized – The request requires an user authentication
 - 403 – Forbidden – The server understood the request, but is refusing it or the access is not allowed.
 - 404 – Not found – There is no resource behind the URI.
 - 422 – Not Processable Entity – Should be used if the server cannot process the entity

Best Practices

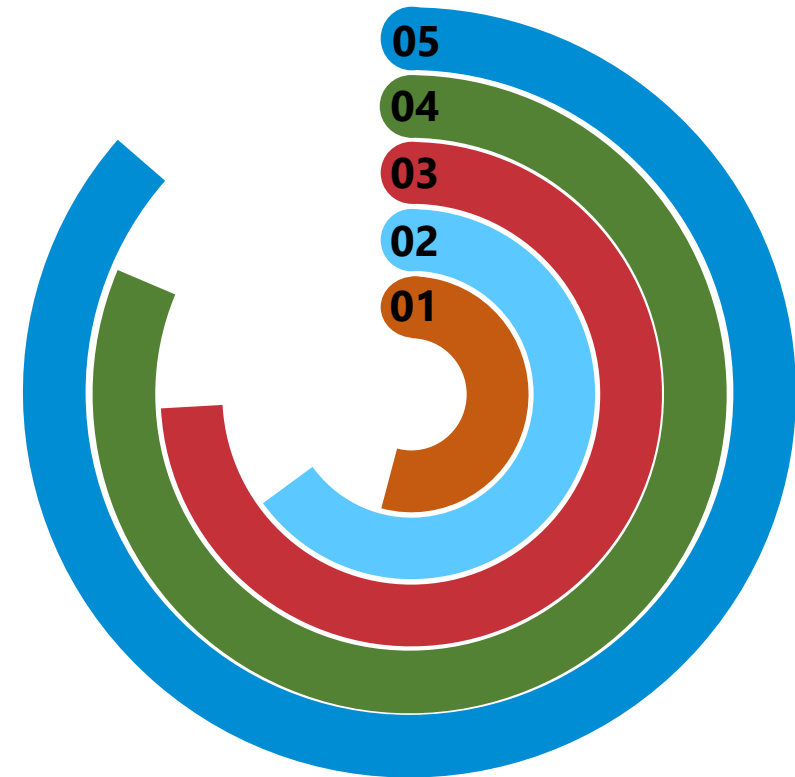
Overriding Methods

- Allow overriding HTTP method
 - Some proxies support only POST and GET methods. To support a RESTful API with these limitations, the API needs a way to override the HTTP method.
 - Use the custom HTTP Header X-HTTP-Method-Override to override the POST Method.

REST APIs

Session Outline

- 01 Introduction**
- 02 Working with REST APIs**
- 03 Limitations**
- 04 Best Practices**
- 05 Hands On Exercise**



Hands on Exercise

Practice makes a man perfect



Summarize REST

Let's take a look back on Rest APIs session

Introduction

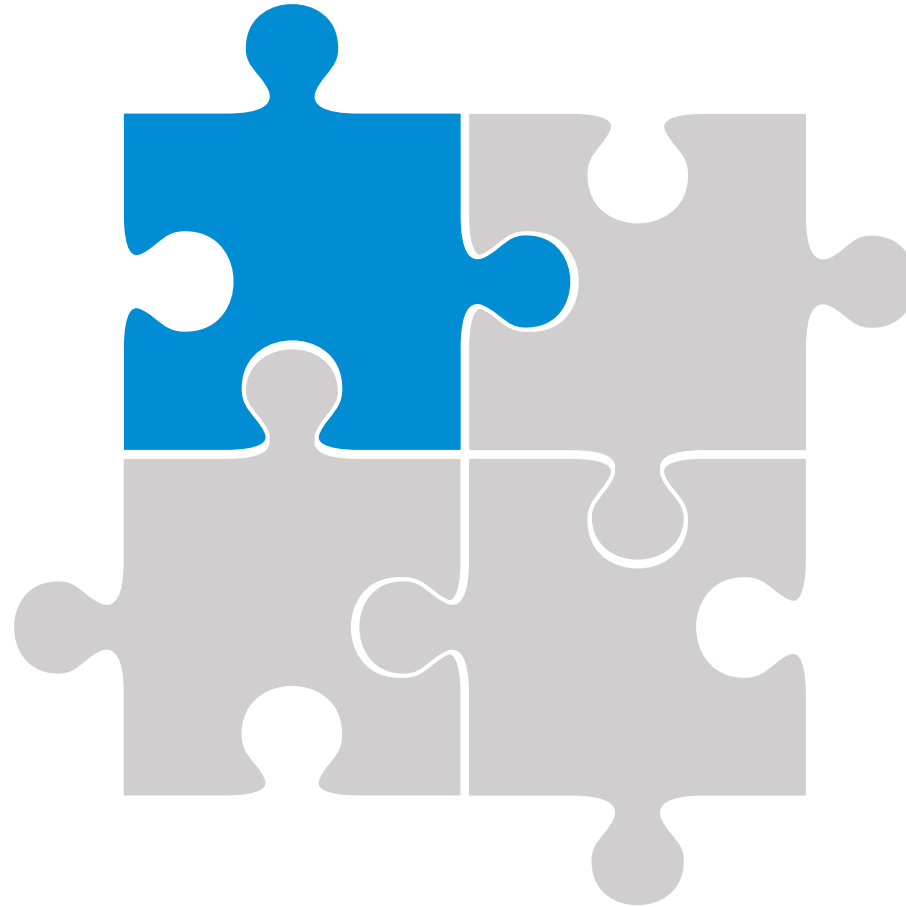
What is API

What are Rest APIs

Why Rest APIs

How Rest API Works

Sample Rest Requests



Summarize REST

Let's take a look back on Rest APIs session

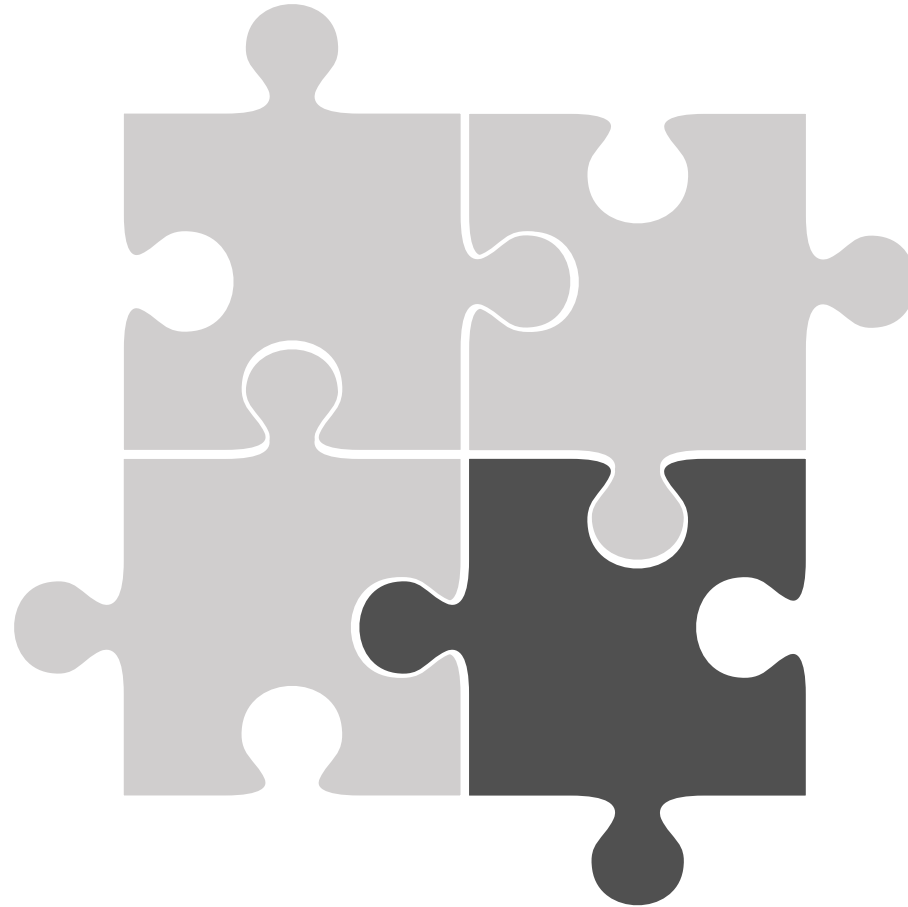
Working with Rest APIs

- Authenticate & Connect
- Set headers
- Form and send the request
- Process the Output



Summarize REST

Let's take a look back on Rest APIs session



Limitations

Latency

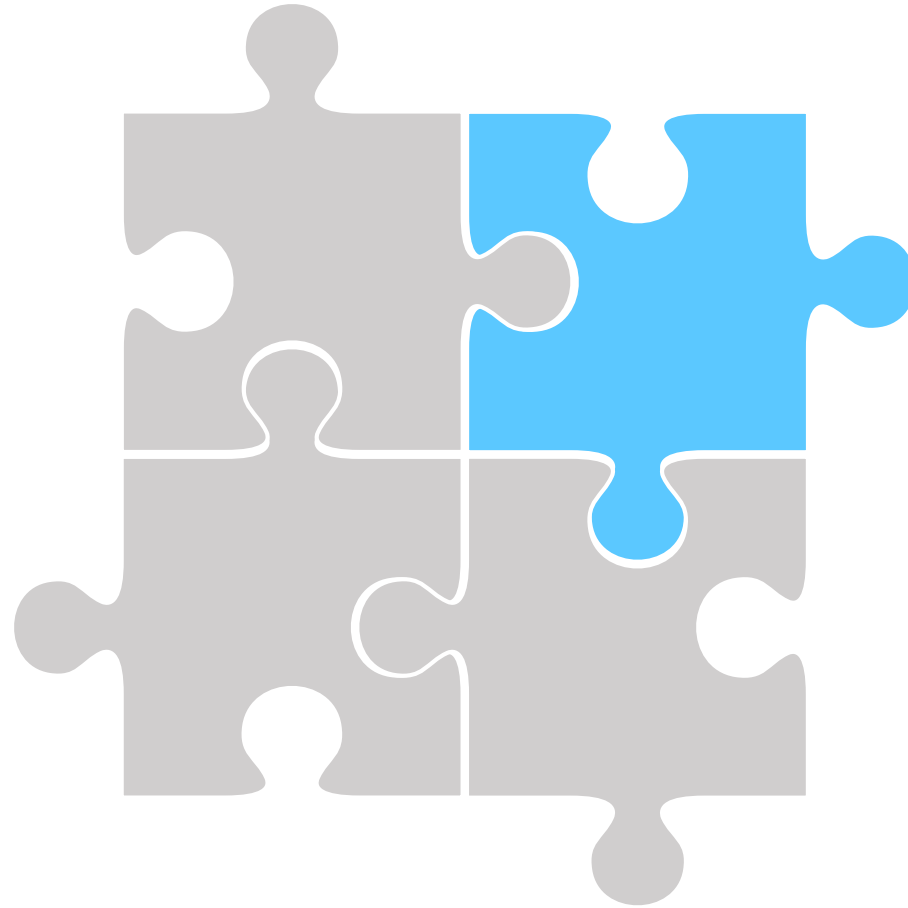
Very Few Verbs

Combined Codes are confusing

Unavailability for Public Consumption

Summarize REST

Let's take a look back on Rest APIs session



Best Practices

- Clean Request
- Error Handling
- Overriding Methods

Summarize REST

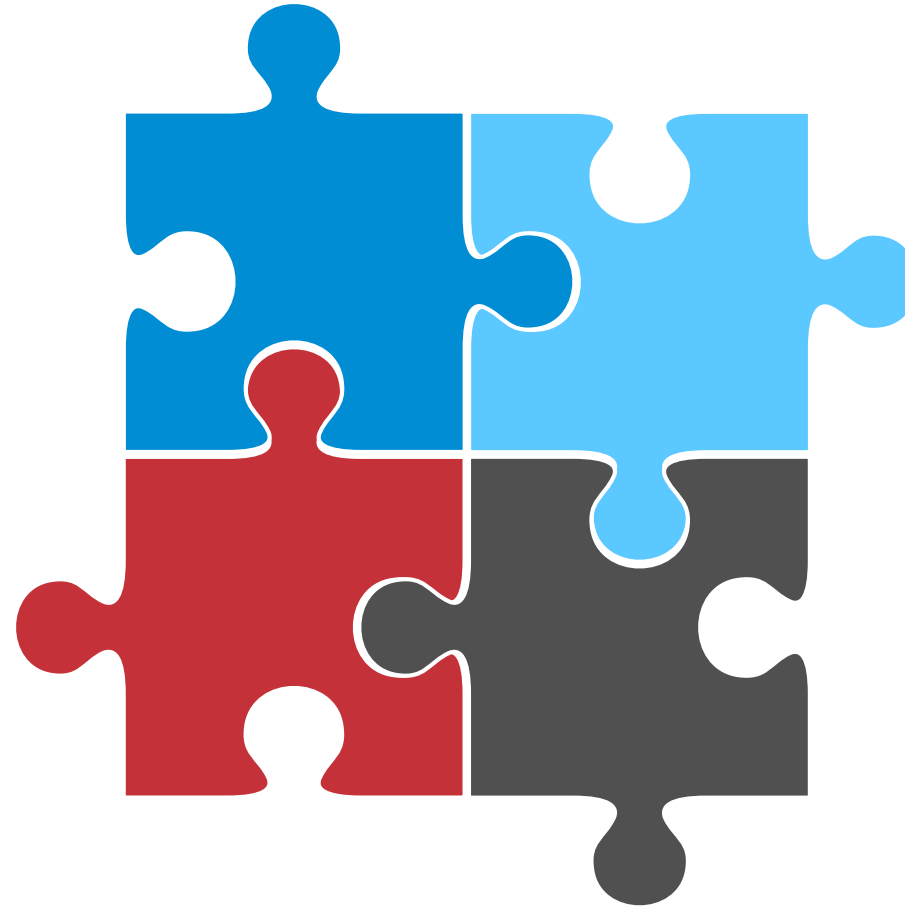
Let's take a look back on Rest APIs session

Introduction

- What is API
- What are Rest APIs
- Why Rest APIs
- How Rest API Works
- Sample Rest Requests

Working with Rest APIs

- Authenticate & Connect
- Set headers
- Form and send the request
- Process the Output



Best Practices

- Clean Request
- Error Handling
- Overriding Methods

Limitations

- Latency
- Very Few Verbs
- Combined Codes are confusing
- Unavailability for Public Consumption

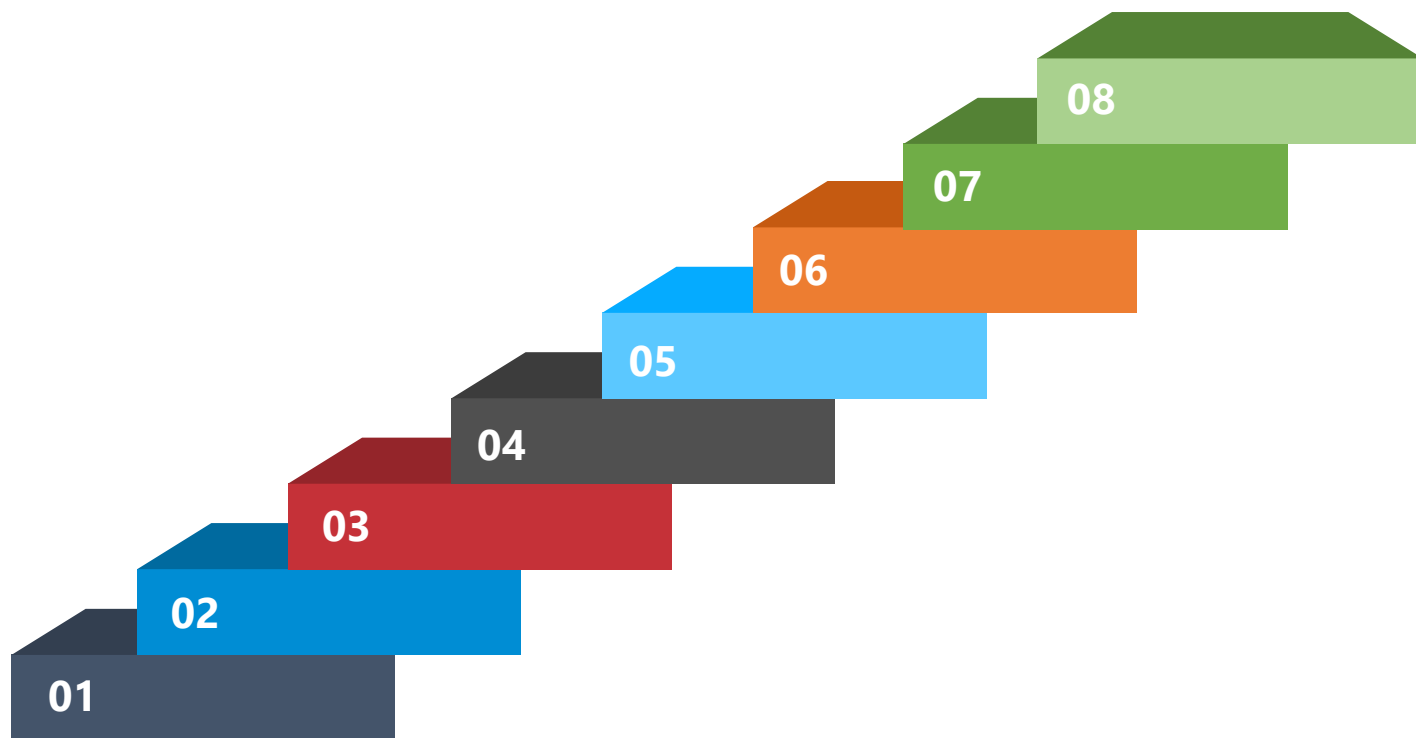
GEL Scripting

Hands-on with GEL Scripting, XOG and the REST API

GEL Scripting

Session Outline

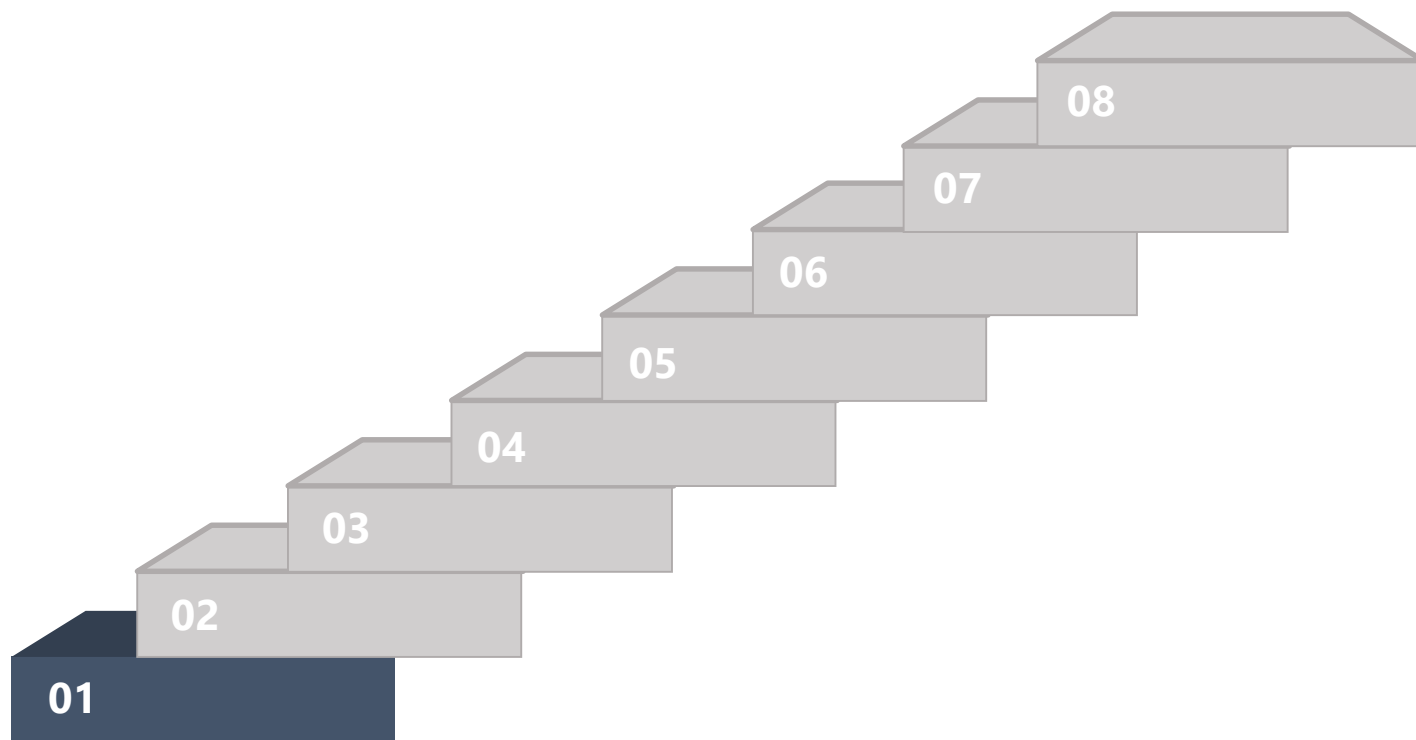
- 01 Introduction
- 02 GEL Script Structure
- 03 Operations
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



GEL Scripting

Session Outline

- 01 Introduction**
- 02 GEL Script Structure
- 03 Operations
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



What is GEL

Getting Started

- Generic Execution Language
- Can be used to turn XML into executable code.
- It is based on Jelly, a jakarta.apache.org Commons project.
- extended and embedded into Clarity PPM
- GEL run-time is packaged with XOG in the XOG client
- Additional references and information can be found in the CA Documentation (Developer Guide) – <https://docops.ca.com/ca-ppm/15-6/en>
- At the Apache Jelly website at - <http://jakarta.apache.org/commons/jelly/index.html>

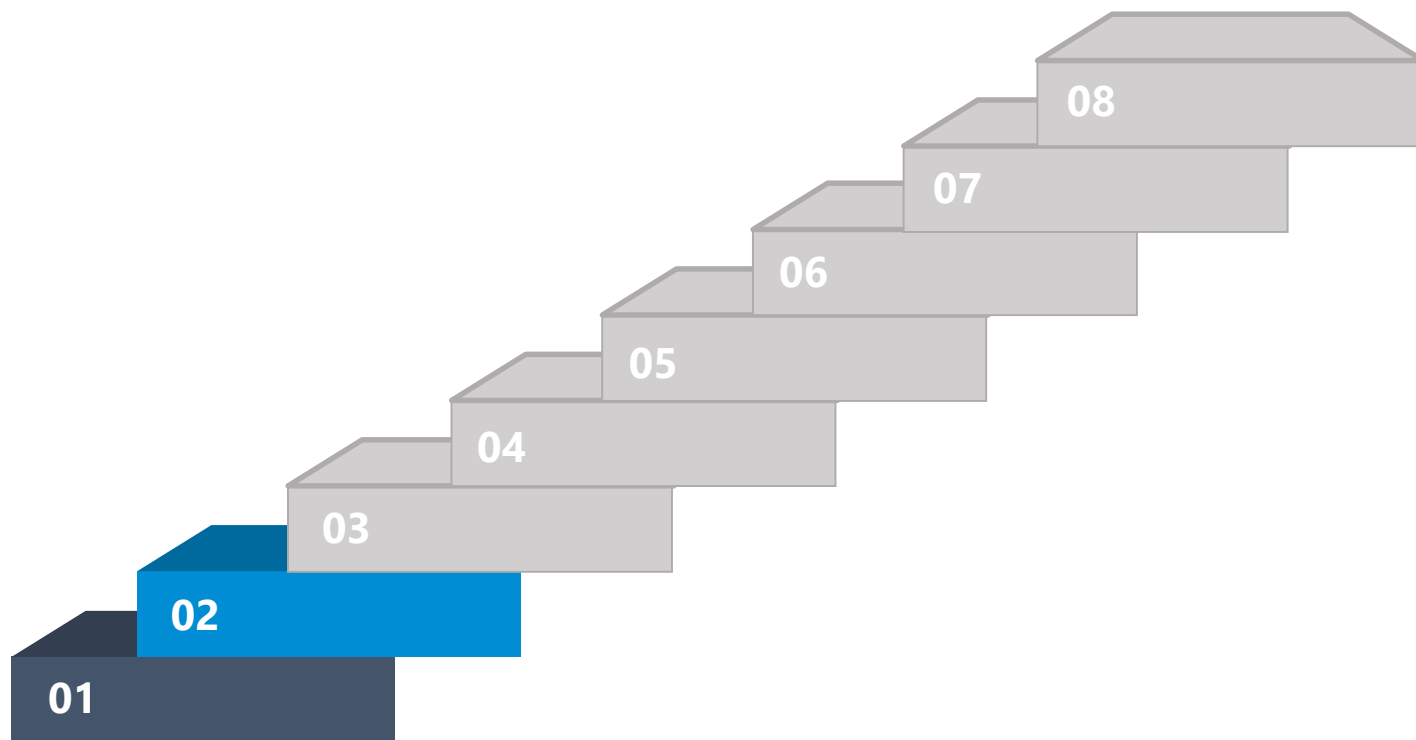
Capabilities of GEL

- GEL can be used in following areas
 - Web services
 - File system
 - JDBC
 - FTP
 - email

GEL Scripting

Session Outline

- 01 Introduction
- 02 GEL Script Structure**
- 03 Operations
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



GEL Script Structure

- A typical GEL script contains
 - Header
 - Namespaces
 - Body/Code
 - Tags
 - Footer

```
<gel:script
  xmlns:core="jelly:core"
  xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
  xmlns:sql="jelly:sql"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- CODE GOES HERE -->
</gel:script>
```


GEL Script Structure

Header

- Header contains the namespaces and the libraries to be used in the script.

```
<gel:script
  xmlns:core="jelly:core"
  xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
  xmlns:sql="jelly:sql"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

GEL Script Structure

Namespaces

- Inclusion of the namespaces for any library gives GEL the ability to perform operations defined in that library.
- Many libraries are contained as Clarity OOTB libraries.
 - Core
 - GELTagLibrary

```
xmlns:core="jelly:core"  
xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"  
xmlns:sql="jelly:sql"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

GEL Script Structure

Body/Code

- Body includes the queries, XMLs, tags for processing etc.
- Includes the logic to perform certain action using GEL script

GEL Script Structure

Tags

- A GEL script is an executable XML file that is built from qualified elements bound to Java code called tags.
- Every opening tag contains its corresponding closing tag except for singular tags.

```
<gel:script>  
...  
</gel:script>
```



Regular Tag

```
<core:set value="0" var="error_count"/>
```



Singular Tag

GEL Script Structure

Footer

- Footer marks the end of the GEL Script XML.

```
</gel:script>
```



An entire script always resides within the GEL script tag.

GEL Script Structure

Common & Workflow Control Tags

- Tags can be divided into following categories:
 - Variables/parameters
 - Loops
 - Conditionals



Information contained within GEL tags is case sensitive

GEL Script Structure

Variables/Parameters

- `<gel:parameter>`
 - Allows values to be passed into a GEL script from a Clarity process.
 - Inside the GEL script, a parameter can be referred as any other variable using the **`#{variablename}`** syntax.

```
<gel:parameter var="XOGUsername" default="admin"/>  
<gel:parameter var="XOGPassword" default="password" secure="true"/>
```



The optional attribute `secure="true"` causes Clarity to hide the actual value in the user interface with asterisks (*).

GEL Script Structure

Variables/Parameters

- `<core:set>`
 - Used to set basic java variables.
 - ones those do not need to be extracted from an XML document.
 - Refer to the variable using the `${variablename}` syntax.

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


GEL Script Structure

Variables/Parameters

- `<gel:set>`
 - Used when it is necessary to extract the value of the variable from an XML document.
 - This tag differs from the `<core:set>` tag in that it takes a `select` attribute which in turn requires an XPath statement.
 - If you are unfamiliar with XPath, think of it as a hierarchy mapping of the XML document.

```
<gel:set asString="true" select="$xogout//XOGOutput/Status/@state" var="xogStatus"/>
```

GEL Script Structure

Variables/Parameters

- `<gel:persist>`
 - This tag allows you to set variables with a scope that extends beyond the current script.
 - There are three scopes:
 - Process
 - Instance
 - Global

```
<gel:persist var="" value="" scope="PROCESS"/>  
<gel:persist var="" value="" scope="INSTANCE"/>  
<gel:persist var="" value="" scope="GLOBAL"/>
```

GEL Script Structure

Variables/Parameters

- `<gel:parse>`
 - To create an XML document in memory.
 - Used to generate an entire XML document, or specific nodes.

```
<gel:parse var="loadContent">
  <NikuDataBus xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:noNamespaceSchemaLocation="../xsd/nikuxog_resource.xsd">
    <Header version="15.4.1.215" action="write" objectType="resource"
      externalSource="ORACLE-FINANCIAL"/>
    <Resources>
      <Resource resourceId="user01" isActive="true">
        <PersonalInformation lastName="rana" firstName="kritika"
          emailAddress="kritika.rana@pemari.com"/>
      </Resource>
    </Resources>
  </NikuDataBus>
</gel:parse>
```

GEL Script Structure

Built-In Parameters

- 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.
 - Else the `#{gel_objectInstanceid}` parameter contains the object instance ID.
 - Process ID
 - `#{gel_processId}` is the process identifier; all instances of a process share this identifier.
 - Process instance ID
 - `#{gel_processInstanceid}` is the process instance identifier; all instances have a unique value.

GEL Script Structure

Loop Tags

<core:forEach>

```
<core:forEach items="1, 2, 3" var="value">  
  <gel:out>Value = ${value}</gel:out>  
</core:forEach>
```

<gel:forEach>

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



1. The core for-Each loop is used as a simple programming loop whereas the gel-for-each loop is a loop which is used to loop through tags inside a XML document.
2. In gel-For-each loop the select attribute contains the XPATH string as an input, which in-turn helps to parse the XML document.

GEL Script Structure

Loop Tags

- `<core:while>`
 - Similar to the while loop in other programming languages like JAVA, C, C++

```
<core:set value="0" var="num"/>
<core:while test="{num!=7}">
    <gel:log>Current Value: {num}</gel:log>
    <core:set value="{num+1}" var="num"/>
</core:while>
```

GEL Script Structure

Conditional Tags

- `<core:if>`

- Similar to the **if** loop in other programming languages like JAVA, C, C++



```
<core:if test="${age > 5}">
  <gel:out>Age is greater than 5</gel:out>
</core:if>
```

- `<core:choose>`

- Similar to the **if-else** loop in other programming languages like JAVA, C, C++



```
<core:if test="${age > 5}">
  <gel:out>Age is greater than 5</gel:out>
</core:if>

<core:choose>
  <core:when test="${result == 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>
```

GEL Script Structure

Conditional Tags

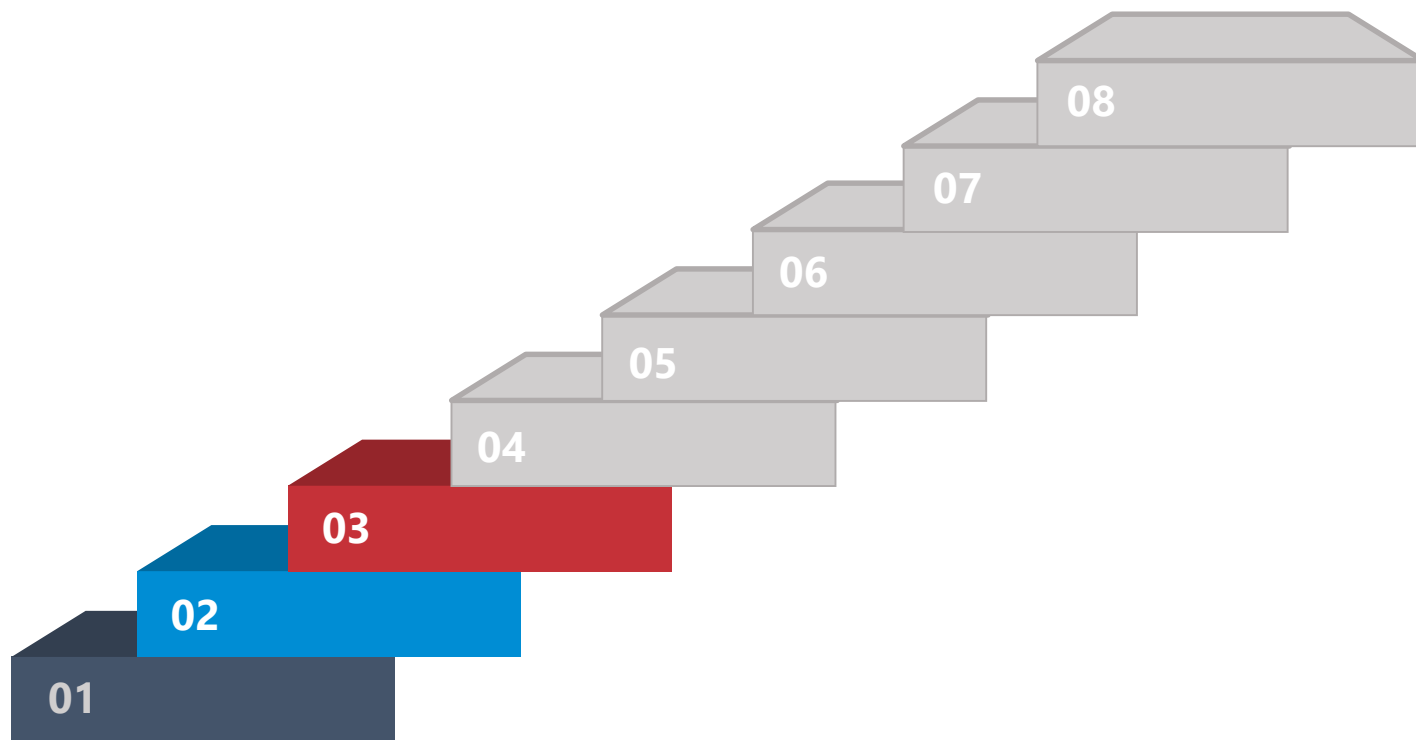
- `<core:switch>`
 - Similar to the **switch** statement in other programming languages like JAVA, C, C++

```
<core:switch on="{caseType}">
  <core:case fallThru="true" value="first"/>
  <core:case value="second">
    <gel:log>Second!</gel:log>
  </core:case>
  <core:case fallThru="true" value="third"/>
  <core:default>
    <gel:log>Default!</gel:log>
  </core:default>
</core:switch>
```


GEL Scripting

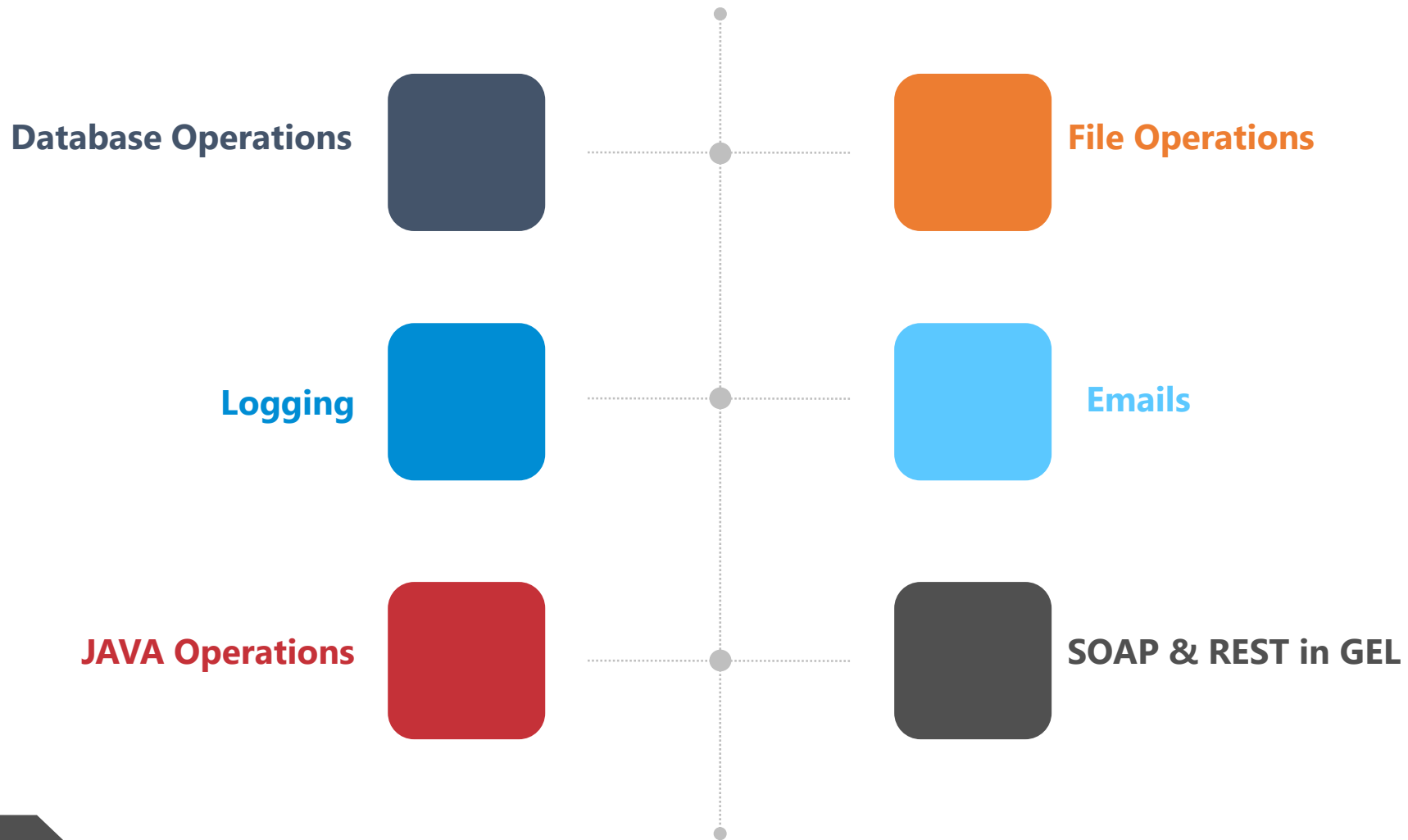
Session Outline

- 01 **Introduction**
- 02 **GEL Script Structure**
- 03 **Operations**
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



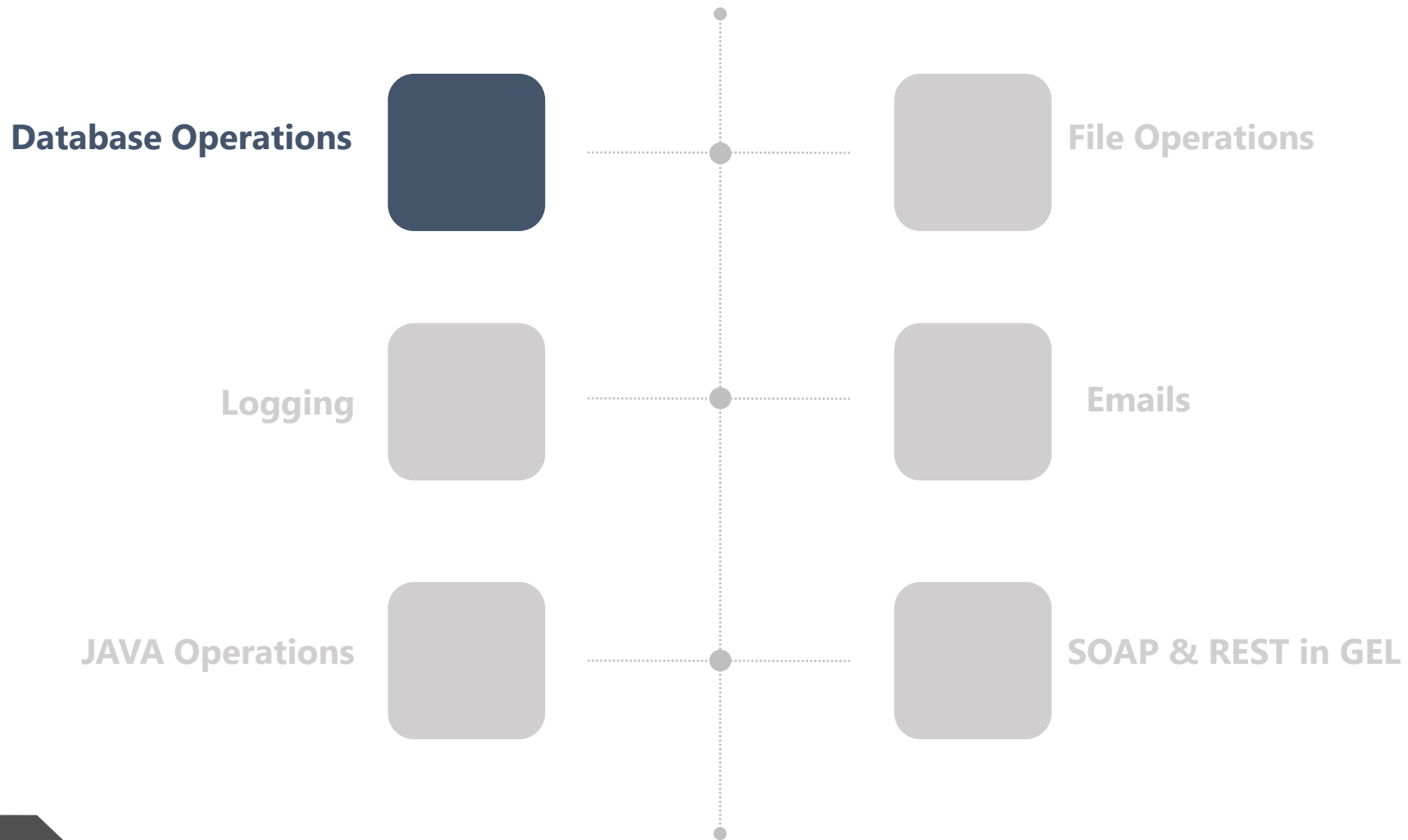
GEL Scripting

Various Operations



GEL Scripting

Various Operations



Various Operations

Connect to DB

- There are two ways to connect to the database
 - `<gel:setDataSource>`
 - `<sql:setDataSource/>`

Database Operations

Connect to DB

- `<gel:setDataSource>`
 - Uses the connection properties from Clarity PPM's properties (set in the CSA)
 - **Var** attribute is optional.
 - If not specified and only one datasource is set, then all SQL calls will use that
 - If a datasource variable is set, it is required to reference it in subsequent tags.

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

Database Operations

Connect to DB

- `<sql:setDataSource>`
 - To use any external connection, by creating an external connection entry in the NSA
 - Value of dbID is name of the connection in NSA.
 - To use the external database connection; without having an entry in NSA

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

Database Operations

Single Result Query

- Single Result Query
 - Given syntax can be used to extract data from a query which returns single row.

```
<sql:query escapeText="0" var="Res">
<![CDATA[
    SELECT ID
    FROM SRM_RESOURCES
    WHERE UNIQUE_NAME = 'admin']]>
</sql:query>

<core:set value="{Res.rows[0].ID}" var="firstRow"/>
```

Database Operations

Multiple Result Queries

- Multiple Result Query
 - Following ways can be used to extract data from multiple result queries

Using Column labels

```
<sql:query escapeText="0" var="MultiRes">
<![CDATA[
    SELECT ID
    FROM SRM_RESOURCES
]]>
</sql:query>

<core:forEach items="{MultiRes.rows}" var="row">
    <core:set value="{row.ID}" var="firstColumn"/>
</core:forEach>
```

Using Row by Index

```
<sql:query escapeText="0" var="MultiRes">
<![CDATA[
    SELECT ID
    FROM SRM_RESOURCES
]]>
</sql:query>

<core:forEach items="{MultiRes.rowsByIndex}" var="row">
    <core:set value="{row[0]}" var="firstColumn"/>
</core:forEach>
```


Database Operations

Update Via SQL

- Updates should be done using the transaction tag.
- This ensures that they follow the **all or none** update.

```
<core:set value="0" var="errorCount"/>
<sql:transaction>
  <core:catch var="error">
    <sql:update escapeText="0" var="totalInserts">
      <![CDATA[ Update TABLE_NAME Set Column1 = 'Value1'
                where Column2 = 'Value2' ]]>
    </sql:update>
  </core:catch>
  <core:if test="{error != null}">
    <core:set value="1" var="errorCount"/>
  </core:if>
  <!-- COMMIT OR ROLLBACK SQL TRANSACTION -->
</sql:transaction>
```

Database Operations

Binding Variables

- Binding variables help in:
 - code reusability
 - prevention of SQL injection

```
<sql:update escapeText="0" var="updCheck">
  <![CDATA[
    UPDATE odf_ca_project cust_prj
    SET cust_prj.cust_start_date= sysdate,
        cust_prj.cust_progress= ?
    WHERE cust_prj.id = ?
  ]]>
<sql:param value="{updCheck.resourceId}"/>
<sql:param value="{gel_objectInstanceId}"/>
</sql:update>
```

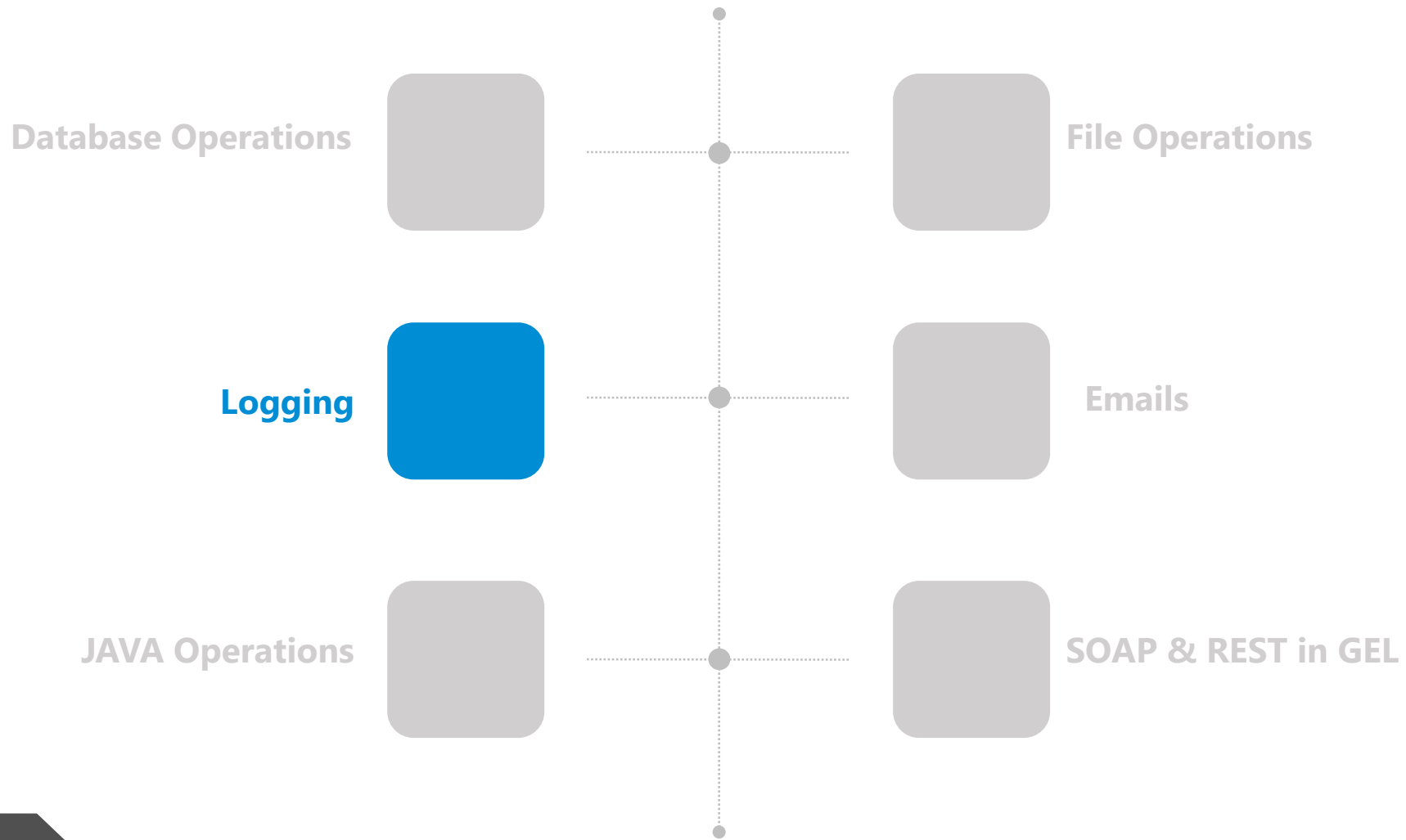
Database Operations

SQL DOs and DON'Ts

- ❌ Try not to set any unnecessary variables.
- ❌ Do not insert using SQL.
- ❌ Do not update OOTB tables.
- ✅ SQL updates are best suited for Custom objects and custom attributes.
- ✅ Do update the 'last_updated_date' and 'last_updated_by' columns when updating audited attributes.
- ✅ CDATA tag should be used in all queries to prevent the errors due to ('<', '>') characters in the query.

GEL Scripting

Various Operations



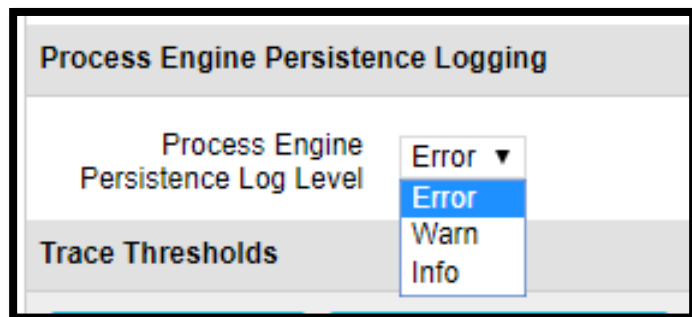
Logging

- `<gel:log>` tag is used to add logs in the process and to create the log data in the log tables.
- Logs prove to be useful in debugging and troubleshooting
- Different levels of logging are:
 - Info
 - Warn
 - Error

Logging

- In recent Clarity releases, an additional log setting has been introduced to control logging, which can be found at:

<servername>/niku/nu#action:security.loggerConfig



- Its default value after upgrade is set to **'Error'**.

Logging

Different Levels of Logging

- Info

<input type="checkbox"/>	Process	ID		Primary Object	Object Name	Progress	Steps In Progress	Status	Messages
<input type="checkbox"/>	Idea Sync	tso_idea_sync	  						

- Warning

<input type="checkbox"/>	Process	ID		Primary Object	Object Name	Progress	Steps In Progress	Status	Messages
<input type="checkbox"/>	RallyPPM Presales Data Sync	rallyPPMDataSync	  						

- Error

<input type="checkbox"/>	Process	ID		Primary Object	Object Name	Progress	Steps In Progress	Status	Messages
<input type="checkbox"/>	Upgrade Status Report Subobject	cop_upgradeStatusRpt	  				Upgrade Status Report SubObject		

GEL Scripting

Various Operations



JAVA Operations

- Core library defines the JAVA methods that can be used inside GEL.
- Not all the JAVA methods are available inside the library.

JAVA Operations

Basic Methods

- `<core:new>`
 - It is used to instantiate Java classes

```
<core:new className="java.net.URL" var="test">  
<core:arg type="java.lang.String" value="${testString}"/>  
</core:new>
```

JAVA Operations

Basic Methods

- `<core:invoke>`
 - It is used to call a method on an instantiated object

```
<core:invoke method="openConnection" on="{test}" var="testConnect"/>
```

JAVA Operations

Basic Methods

- `<core:expr>`
 - It is used to call a method on an instantiated Java object where the access to the result of the operation is not required.

```
<core:expr value='${connection.setRequestMethod("POST")}'/>
```

JAVA Operations

Basic Methods

- `<core:invokeStatic>`
 - It is used to call a static method of a Java class

```
<core:invokeStatic className="com.niku.union.utility.Base64"  
    method="encode" var="encodedString">  
    <core:arg type="java.lang.String" value="{dataValue}"/>  
</core:invokeStatic>
```

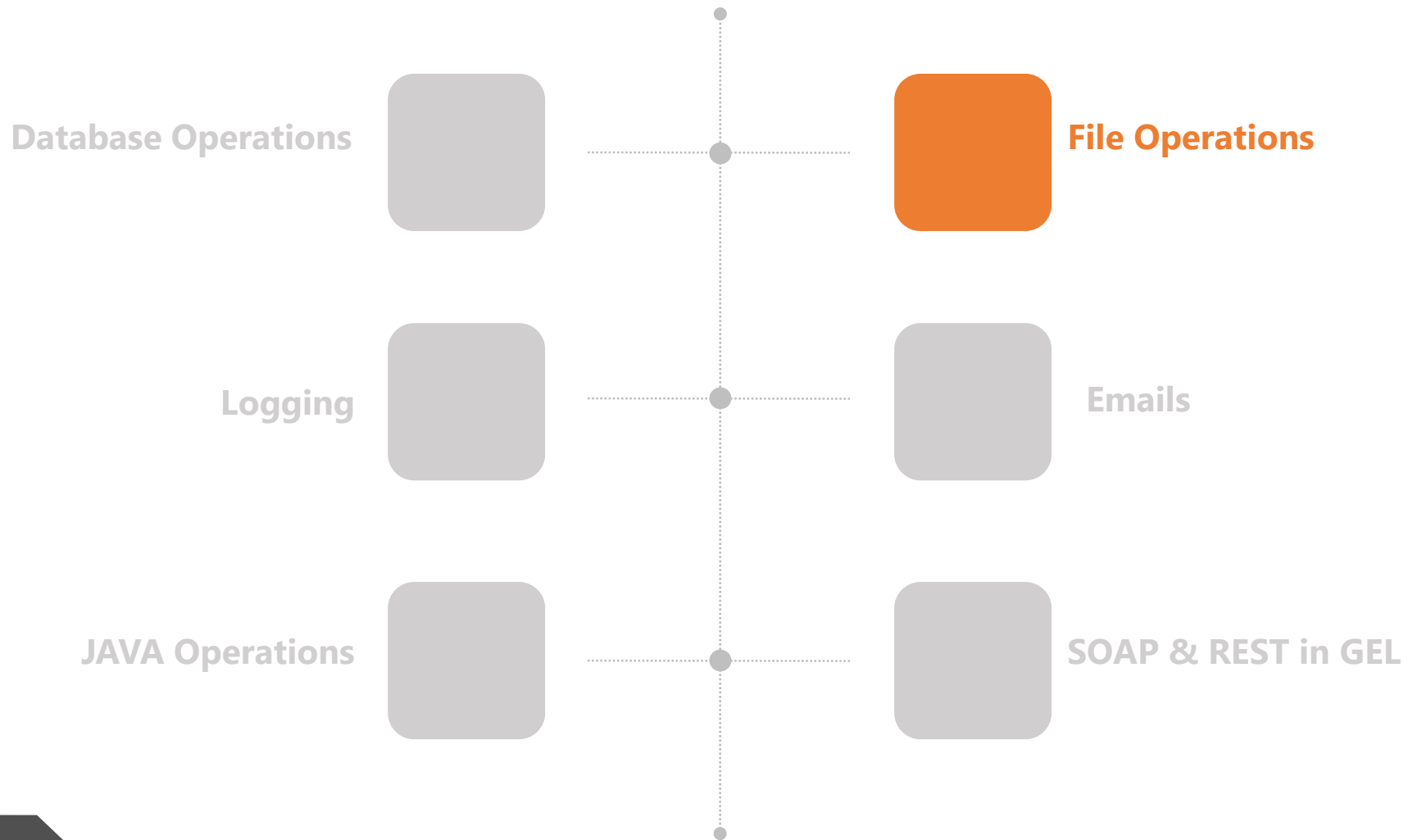
JAVA Operations

Usage of JAVA in GEL

- To fill the gaps in functionality which cannot be provided by GELTagLibrary
 - E.g., Moving, copying, and deleting files or directories.
 - Making file operations more flexible
 - To perform any action which cannot be done using GEL libraries
 - Better exception handling (covered in later slides)

GEL Scripting

Various Operations



File Operations

Introduction to File Operations

- There are various operations that GEL can perform for handling files
 - GEL can open a file
 - read the file
 - parse out all the nodes and attributes
 - write to the file
 - It can also perform FTP operations on files.
- Following are not allowed
 - It cannot create a directory
 - Move files around
 - Delete files



JAVA operations can overcome the exceptions for file handling.

File Operations

Read File

- GEL Script code block to read a file

```
<gel:script xmlns:core="jelly:core"
  xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:files="jelly:com.niku.union.gel.FileTagLibrary">
  <files:readFile fileName="MyFile.csv" delimiter="," var="MyData" embedded="false"/>
  <core:forEach items="{MyData.rowsByIndex}" var="row" begin="1" end="10">
  <gel:log level="INFO"> Name: ${row[0]} </gel:log>
  <gel:log level="INFO"> Age: ${row[1]} </gel:log>
  </core:forEach>
</gel:script>
```

File Operations

Read File

- GEL Script code block to write to a file

```
<gel:script xmlns:core="jelly:core"
  xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:files="jelly:com.niku.union.gel.FileTagLibrary">
  <file:writeFile delimiter="," embedded="false" fileName="myFile.csv ">
    <file:line>
      <file:column value="hello"/>
      <file:column value="world"/>
    </file:line>
  </file:writeFile>
</gel:script>
```

File Operations

FTP

- FTPTagLibrary can be used to read or write files on FTP server
- Following tags are available
 - ftp:open
 - ftp:put
 - ftp:get

File Operations

FTP

- Following are the sample FTP read & write code blocks

Read Operation

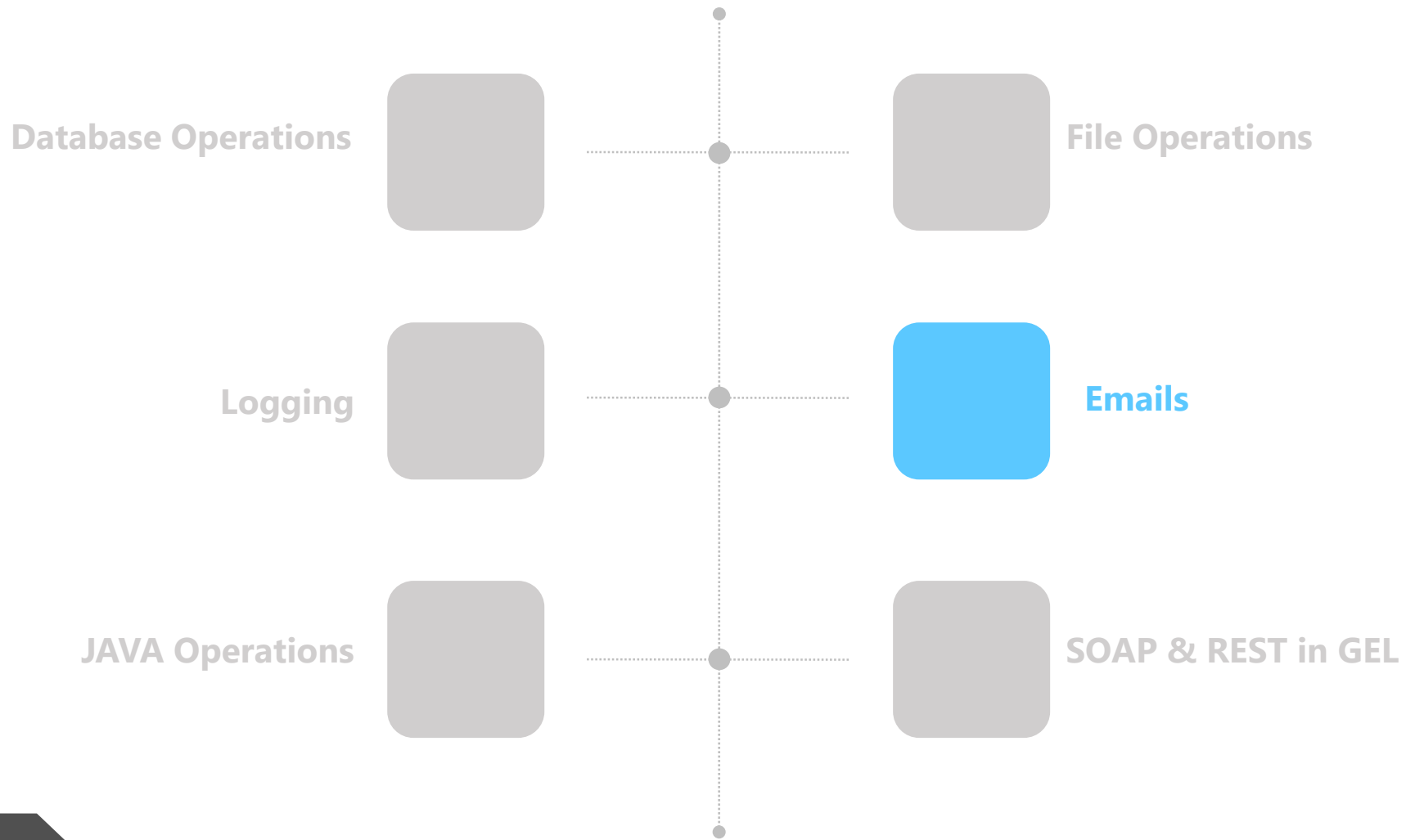
```
<ftp:open hostName="myclarityserver"  
  user="niku" password="clarity">  
<ftp:get localDir="c:/temp" fileName="app-ca.log"  
  remoteDir="/niku/clarity/logs"/>  
</ftp:open>
```

Write Operation

```
<ftp:open hostName="localhost"  
  user="niku" password="clarity">  
<ftp:put localDir="/home/niku/xog/bin"  
  fileName="gel.bat" remoteDir="/tmp"/>  
</ftp:open>
```

GEL Scripting

Various Operations



Emails

Sending emails

- To send dynamic emails based on specific events.
- There are two tags for email:
 - `<gel:email>` : to send text/html emails
 - `<email:email>` : to send text emails with attachments



Make Sure to choose the right type as per your requirement of content in the email being sent.

File Operations

FTP

- Following are the sample <gel:email> and <email:email> code blocks

<gel:email>

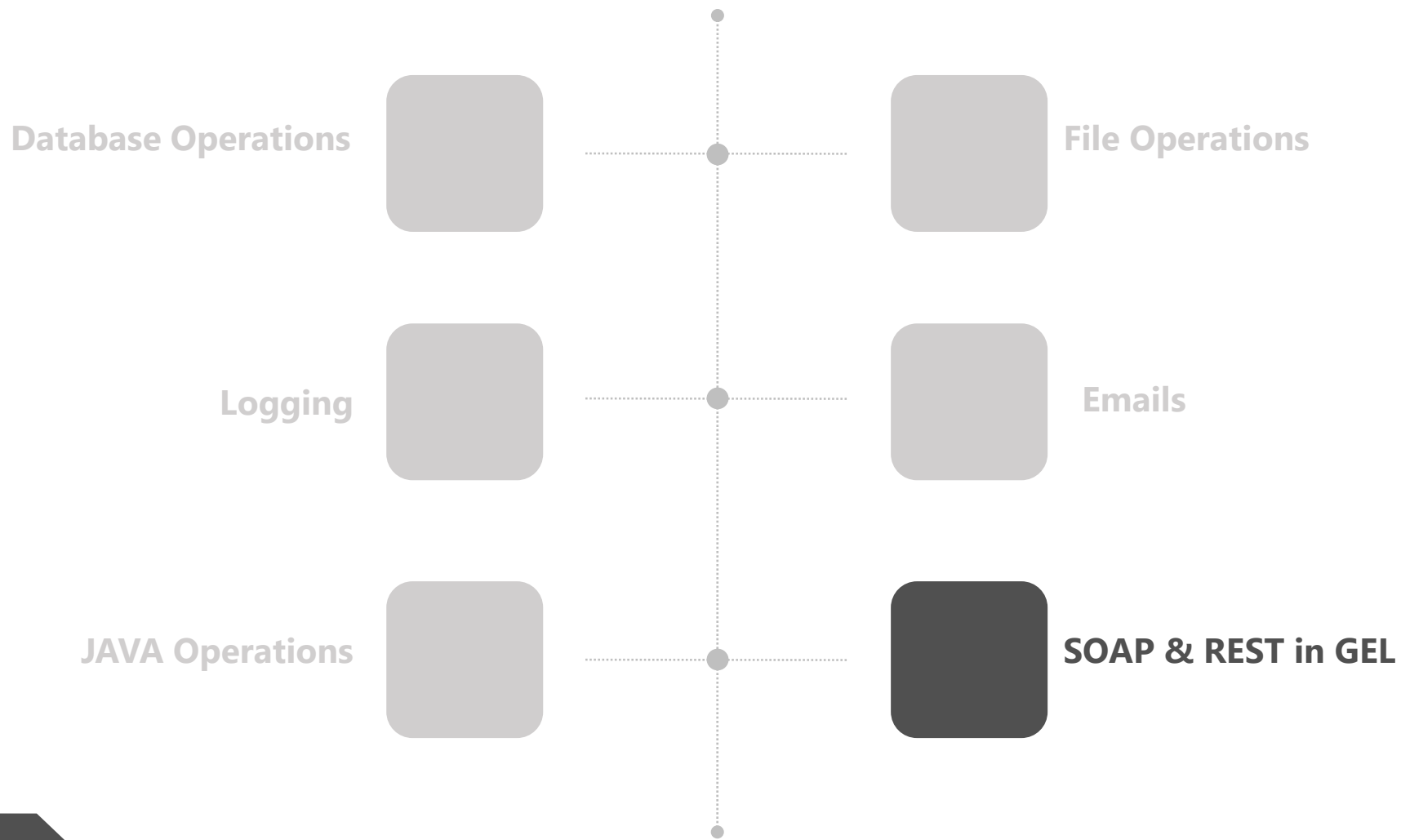
```
<gel:script xmlns:core="jelly:core"
xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:files="jelly:com.niku.union.gel.FileTagLibrary">
<gel:email from="clarity-do-not-reply@ca.com"
  subject="Clarity - Test Email"
  to="kritika.rana@pemari.com">
  <![CDATA[
    Hi User,
    <br/>
    <br/>
    This is a sample <b>HTML</b> email.
    <br/>
    -Regards
    Clarity Admin ]]>
</gel:email>
</gel:script>
```

<email:email>

```
<gel:script xmlns:core="jelly:core"
  xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
  xmlns:email="jelly:email">
  <core:invokeStatic className="java.lang.System"
    method="getenv" var="NIKU_HOME">
    <core:arg value="NIKU_HOME"/>
  </core:invokeStatic>
  <gel:parse file="{NIKU_HOME}/config/properties.xml"
    var="properties"/>
  <gel:set asString="true"
    select="$properties/properties/mailServer/@host"
    var="mailServer"/>
  <email:email to="kritika.rana@pemari.com"
    from="clarity@pemari.com"
    subject="app-ca.log file"
    server="{mailServer}"
    attach="{NIKU_HOME}/logs/app-ca.log">
    App-ca.log File
  </email:email>
</gel:script>
```

GEL Scripting

Various Operations



SOAP & REST in GEL

SOAP & Rest requests in GEL

- SOAP & REST calls can be used to read and write the data in Clarity PPM.
- SOAP calls are generally done by XOG
- REST calls are performed by the REST APIs given by Clarity PPM

SOAP in GEL

Namespaces required for SOAP Call

- In order to use a SOAP call via GEL, we need to import respective namespaces.

```
<gel:script
xmlns:core="jelly:core"
xmlns:gel="jelly:com.niku.union.gel.GELTagLibrary"
xmlns:xog="http://www.niku.com/xog"
xmlns:soap="jelly:com.niku.union.gel.SOAPTagLibrary"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" />
```

SOAP in GEL

Basic SOAP structure

- Basic soap structure:
 - Envelope: (Mandatory) - defines the start and the end of the message.
 - Header: (Optional) - Contains any optional attributes of the message
 - Body: (Mandatory) - Contains the XML data comprising the message being sent.

SOAP in GEL

Steps

- Steps:
 - Login & Authenticate
 - Form & Send the Request
 - Process the output
 - Log out

SOAP in GEL

Login & Authenticate using Session ID

- Code block for obtaining Session ID for authentication

```
<gel:parameter var="username" default="admin"/>
<core:new className="com.niku.union.security.DefaultSecurityIdentifier"
  var="secId" />
<core:invokeStatic var="userSessionCtrl"
  className="com.niku.union.security.UserSessionControllerFactory"
  method="getInstance" />
<core:set var="secId" value="{userSessionCtrl.init(username, secId)}/>
<core:set var="sessionId" value="{secId.getSessionId()}/>
<core:choose>
<core:when test="{sessionId == null}">
<gel:log level="ERROR"> Unable to obtain a Session ID. </gel:log>
</core:when>
<core:otherwise>
<!-- Execute XOG -->
</core:otherwise>
</core:choose>
```

SOAP in GEL

Form the request

- Code block for forming the request

```
<gel:parameter var="username" default="admin"/>
<core:new className="com.niku.union.security.DefaultSecurityIdentifier"
  var="secId" />
<core:invokeStatic var="userSessionCtrl"
  className="com.niku.union.security.UserSessionControllerFactory"
  method="getInstance" />
<core:set var="secId" value="{userSessionCtrl.init(username, secId)}/>
<core:set var="sessionId" value="{secId.getSessionId()}/>
<core:choose>
<core:when test="{sessionId == null}">
<gel:log level="ERROR"> Unable to obtain a Session ID. </gel:log>
</core:when>
<core:otherwise>
<!-- Execute XOG -->
</core:otherwise>
</core:choose>
```

SOAP in GEL

Processing the Response

- Code block for processing the output

```
<gel:set asString="true"  
  select="$xogResponse//XOGOutput/Status/@state"  
  var="xogStatus"/>  
<core:if test="{xogStatus != 'SUCCESS'}">  
<gel:set asString="true"  
  select="$xogResponse//XOGOutput/ErrorInformation/Description/text()" "  
  var="xogErrorDescription"/>  
<gel:log level="error">Failed to update Object. ${xogErrorDescription}</gel:log>
```

SOAP in GEL

Logging out

- Code block logging out

```
<soap:invoke endpoint="internal" var="result">
  <soap:message>
    <soapenv:Envelope>
      <soapenv:Header>
        <xog:Auth>
          <xog:SessionID><![CDATA[{sessionID}]]></xog:SessionID>
        </xog:Auth>
      </soapenv:Header>
      <soapenv:Body>
        <xog:Logout/>
      </soapenv:Body>
    </soapenv:Envelope>
  </soap:message>
</soap:invoke>
```


REST in GEL

Using REST in GEL

- Rest calls can be made using the GEL scripts as well.
- The steps are as following:
 - Authenticate
 - Connect
 - Set headers
 - Form the request
 - Send the request
 - Process the Output

REST in GEL

Authenticate

- GEL Script Code block to Authenticate

```
<core:invokeStatic className="com.niku.union.utility.Base64"  
  method="encode" var="encodedString">  
  <core:arg type="java.lang.String"  
    value="{userName}:{password}"/>  
</core:invokeStatic>  
<core:set var="basicAuth"  
  value="Basic {encodedString}"/>
```

REST in GEL

Connect

- GEL Script Code block to connect

```
<core:set var="restEndPoint"  
  value="http://ppm.example.com/ppm/rest/v1/projects"/>  
<core:new className="java.net.URL" var="restUrl">  
<core:arg type="java.lang.String" value="{restEndPoint}"/>  
</core:new>  
<core:invoke var="connection"  
  method="openConnection" on="{restUrl}">
```

REST in GEL

Set Headers

- GEL Script Code block to Set Headers

```
<core:expr value=' ${connection.setRequestMethod("POST")} ' />
<core:expr value=' ${connection.setRequestProperty("Authorization", authKey)} ' />
<core:expr value=' ${connection.setRequestProperty("Content-type", "application/json")} ' />
<core:expr value=' ${connection.setRequestProperty("Accept", "application/json")} ' />
<core:expr value=' ${connection.setRequestProperty("Connection", "keep-alive")} ' />
<core:expr value=" ${connection.setDoOutput(true)} " />
```

REST in GEL

Sample Request

- Sample Request for Rest call

```
<core:set var="requestJSON" escapeText="false">
  {
    "code": "REST01",
    "isOpenForTimeEntry": "true",
    "description": "Project Created via REST",
    "isActive": "true",
    "name": "REST Project"
  }
</core:set>
```

REST in GEL

Send the request

- Code block for sending the request

```
<core:invoke var="outputStream" method="getOutputStream"  
.....  
  on="{connection}">  
<core:new className="java.io.OutputStreamWriter"  
.....  
  var="outputStreamWriter">  
<core:arg type="java.io.OutputStream"  
.....  
  value="{outputStream}" />  
</core:new>  
<core:expr value="{outputStreamWriter.write(requestJSON)}" />  
<core:expr value="{outputStreamWriter.close()}" />
```

REST in GEL

Process the output

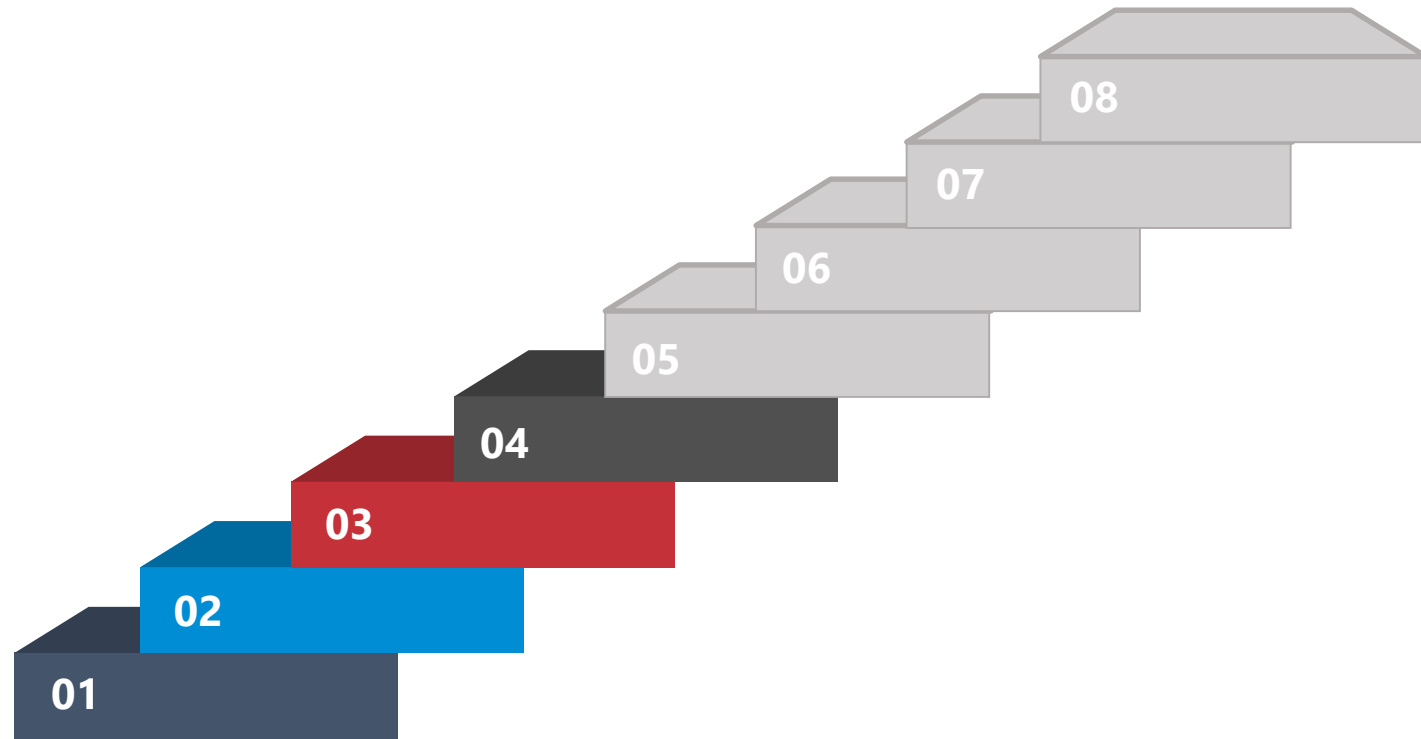
- Code block for processing the response

```
<core:invoke var="restOutput" method="getInputStream"
  on="{connection}">
<core:invoke var="restResponseCode" method="getResponseCode"
  on="{connection}">
<core:choose>
<core:when test="{restResponseCode == '200'}">
<gel:log> Successfully created CA PPM Project </gel:log>
<!-- Convert REST output to String -->
<core:invokeStatic className="org.apache.cxf.helpers.IOUtils"
  method="toString" var="projectOutputString">
<core:arg type="java.io.InputStream" value="{restOutput}"/>
<core:arg value="UTF-8"/>
</core:invokeStatic>
<core:expr value="{restOutput.close()}" />
<core:new className="org.json.JSONObject" var="projectJsonObject">
<core:arg type="java.lang.String" value="{projectOutputString}" />
</core:new>
<core:set var="prjInternalId"
  value="{projectJsonObject.get('_internalId')}" />
<gel:log> Project ID: {prjInternalId} </gel:log>
</core:when>
<core:otherwise>
<gel:log> Failed to create Project </gel:log>
</core:otherwise>
</core:choose>
```

REST APIs

Session Outline

- 01 **Introduction**
- 02 **GEL Script Structure**
- 03 **Operations**
- 04 **XML Manipulation**
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



XML Manipulation

Different ways of Manipulating XML

- XML is at heart of GEL scripting.
- Following are the possible ways to manipulate an XML in Memory.
 - Create the complete XML
 - Read, Update & Delete a node
 - Read & Update an attribute
 - Insert smaller XML into Existing XML
 - Save & Print the XML

XML Manipulation

Manipulating the Nodes

- Read, Update & Delete a node
- <gel:set> tag is used for xml operations
- Value of a node from XML can be saved in a variable as below:

```
<gel:set asString="true" select="$userXML/NikuDataBus/Users/User/text()" var="textContent"/>
```

- Updating Value of a node

```
<gel:set asString="true" select="$userXML/NikuDataBus/Users/User/text()" value="New Text"/>
```

XML Manipulation

Manipulating the Nodes

- Delete a Node

```
<gel:set select="$userXML//Users/User/Groups/Group[@id='pm'] "  
    var="groupToDelete"/>  
<core:set value="{groupToDelete.getParentNode()}" var="parent"/>  
<core:set value="{parent.removeChild(groupToDelete)}" var="void"/>
```

XML Manipulation

Manipulating the Attributes

- Read & Update an attribute
- <gel:set> tag is used for xml operations
- Value of an attribute can be saved in a variable as below:

```
<gel:set asString="true" select="$userXML/NikuDataBus/Users/User/@userName" var="userName"/>
```

- Updating Value of an attribute

```
<gel:set asString="true" select="$userXML/NikuDataBus/Users/User/@userName" value="krana"/>
```

XML Manipulation

Add XMLs to make a big one

- Insert smaller XML into Existing XML

```
<gel:parse var="groupXML">
    <Group id="{groupRow.groupCode}"/>
</gel:parse>
<gel:set insert="true" value="{groupXML}"
select="{userXML/NikuDataBus/Users/User/Groups}"/>
```

XML Manipulation

Print XML

- Printing the XML
 - `<gel:expr>` tag is used for printing the xml by converting that into string.
 - `<gel:out>` tag is used to print the xml in bg-ca logs

```
<gel:log level="WARN"><gel:expr select="$userXML/" /></gel:log>  
<gel:out level="WARN"><gel:expr select="$userXML/" /></gel:out>
```

XML Manipulation

Save XML

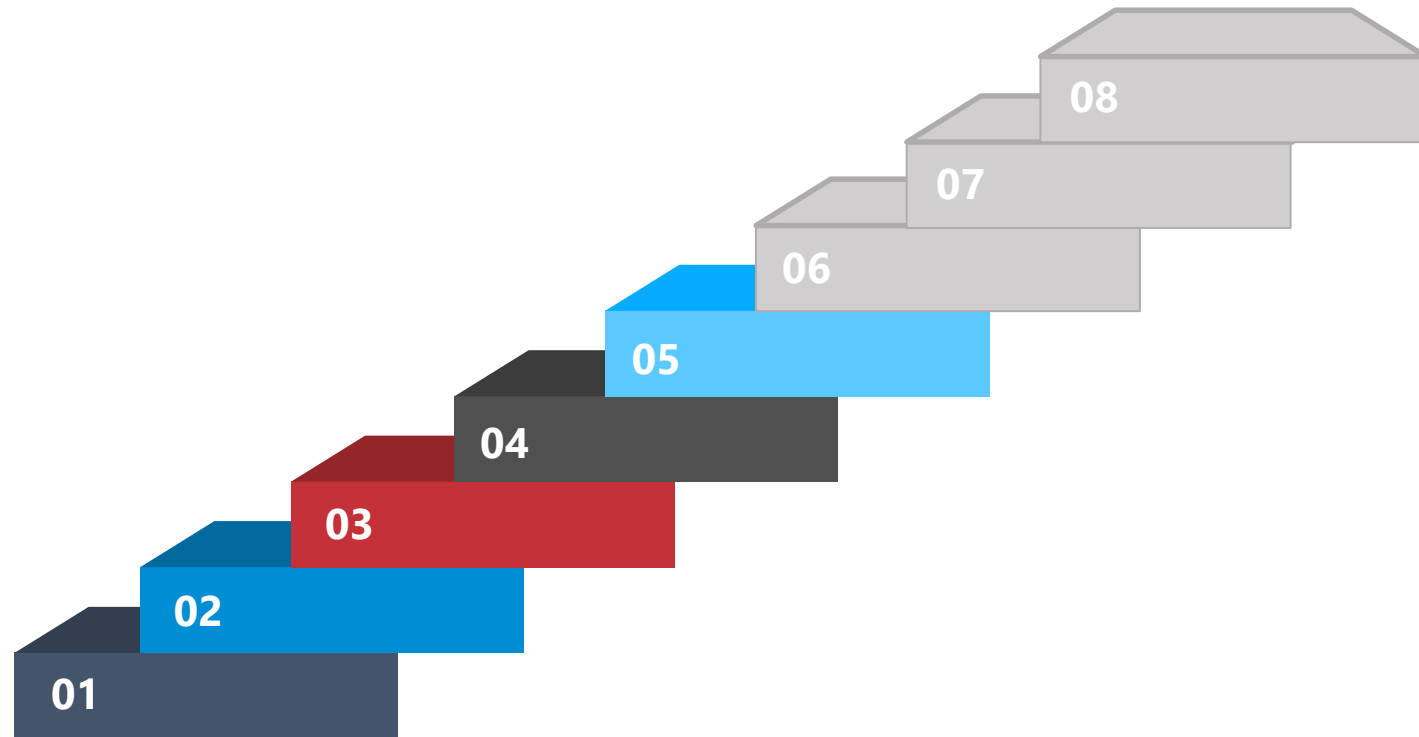
- Saving the XML
 - `<gel:serialize>` tag is used to save the XML document

```
<gel:serialize fileName="/fs0/clarity1/share/userInput.xml" var="{userXML}"/>
```

REST APIs

Session Outline

- 01 **Introduction**
- 02 **GEL Script Structure**
- 03 **Operations**
- 04 **XML Manipulation**
- 05 **Exception Handling**
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



Exception/Error Handling

Catch & Rectify the Exceptions/Errors

- `<core:catch>` tag is used (Replica of try-catch in JAVA)
- Errors/exceptions can occur while doing XOG, SQL query or file related operations.
- Example of SQL exception handling

```
<core:catch var="TableException">
<sql:query escapeText="false" var="fetchQuery">
<![CDATA[
Select * From odf_ca_resource
    WHERE id in (5000000, 5000001)]]>
</sql:query>
</core:catch>
<core:if test="{TableException!=null}">
    <gel:log level="Warn">EXCEPTION : {TableException}</gel:log>
</core:if>
```

Exception/Error Handling

Catch & Rectify the Exceptions/Errors

- Java tags also support error handling on that specific tag

```
<core:invoke method="delete" on="{outXml}"  
var="void" exceptionVar="outException"/>
```

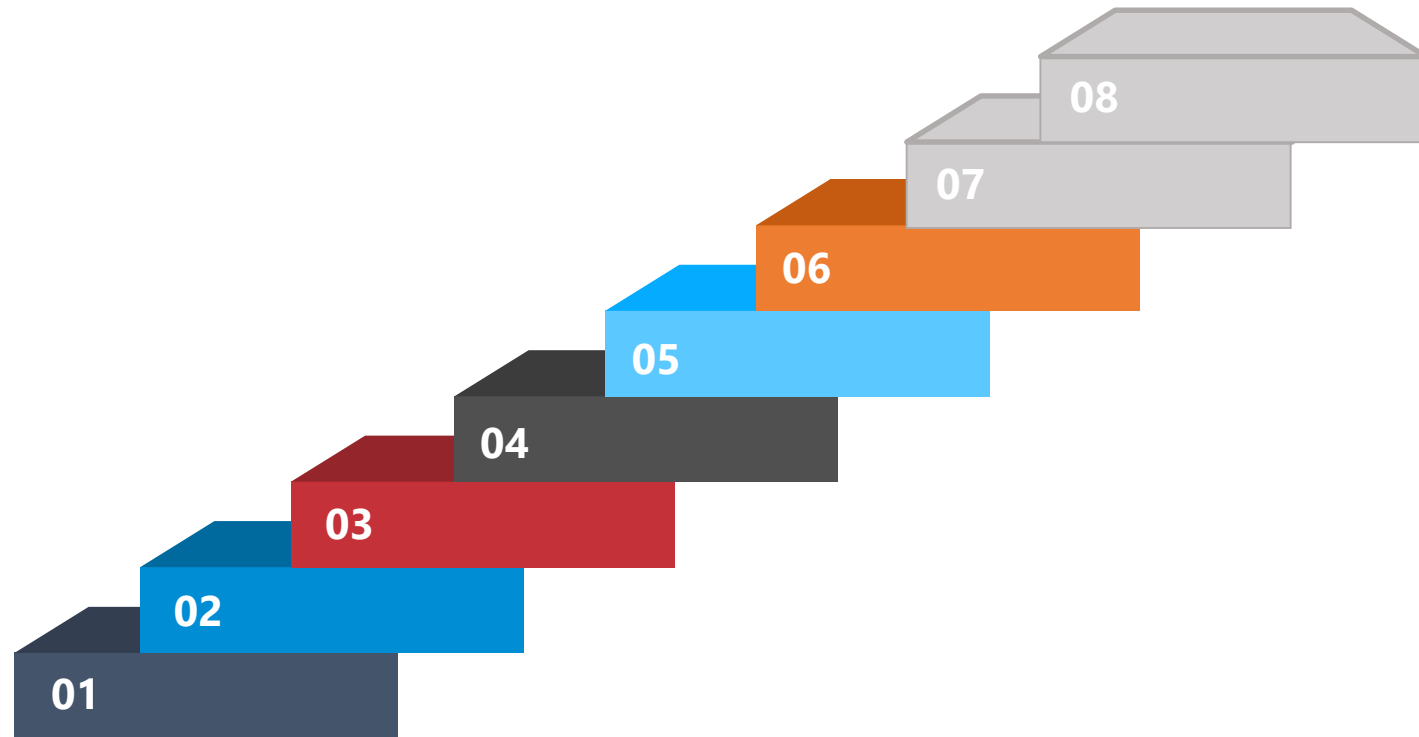


1. The exception handling of GEL is only meant for handling programming exceptions but if there is an error in your script which is a parsing error then that will not be caught by Catch tag.
2. For example if you use symbols like '>' '<' or '&' in your script which will cause parsing error , then these are these are not caught by catch tag.

REST APIs

Session Outline

- 01 **Introduction**
- 02 **GEL Script Structure**
- 03 **Operations**
- 04 **XML Manipulation**
- 05 **Exception Handling**
- 06 **Limitations**
- 07 **Best Practices**
- 08 **Hands On Exercise**



Limitations

Limited Functionality

- JAVA method access is restricted by Core library
 - GEL is dependent on core library.
 - If any JAVA method is not included in the core library, that is not accessible using GEL script.
- Casting a data type to another is not supported in Core library
 - Thus, making it impossible to cast data types

Limitations

Sequential Execution and Absence of Customization

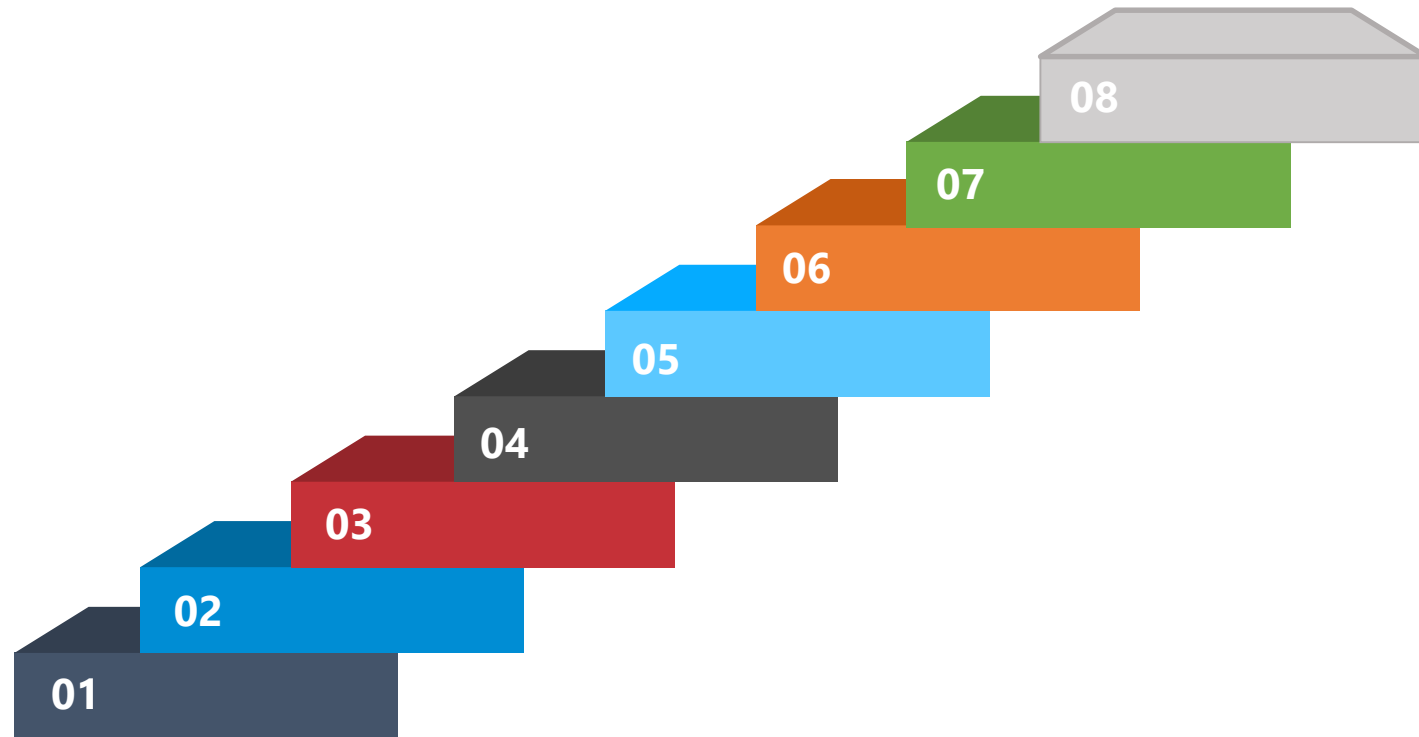
- GEL script executes code sequentially.
 - The drawback is loss of modularity.
- Customization is not possible
 - The code cannot incorporate any custom defined methods, classes etc.

 This drawback is well handled by the structure of Processes in Clarity PPM

REST APIs

Session Outline

- 01 Introduction
- 02 GEL Script Structure
- 03 Operations
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



Best Practices

Syntax Related Practices

- GEL is case sensitive. This statement includes variable names.
- All GEL scripts are contained in XML, therefore all XML rules apply to structure, tags, and special characters.
- Use BETWEEN instead of less than and greater than

Best Practices

Make the code Easy to Understand

- Always make sure to use the proper name spaces and aliases
- Properly indent and format your script, which makes it readable

Best Practices

Avoid Impacting Business

- Avoid sending emails in non-prod environments
 - by either disabling mail server or
 - masking emails
- Use caution when dealing with large XML files.
 - GEL XML tags typically load the entire XML document into memory.

Best Practices

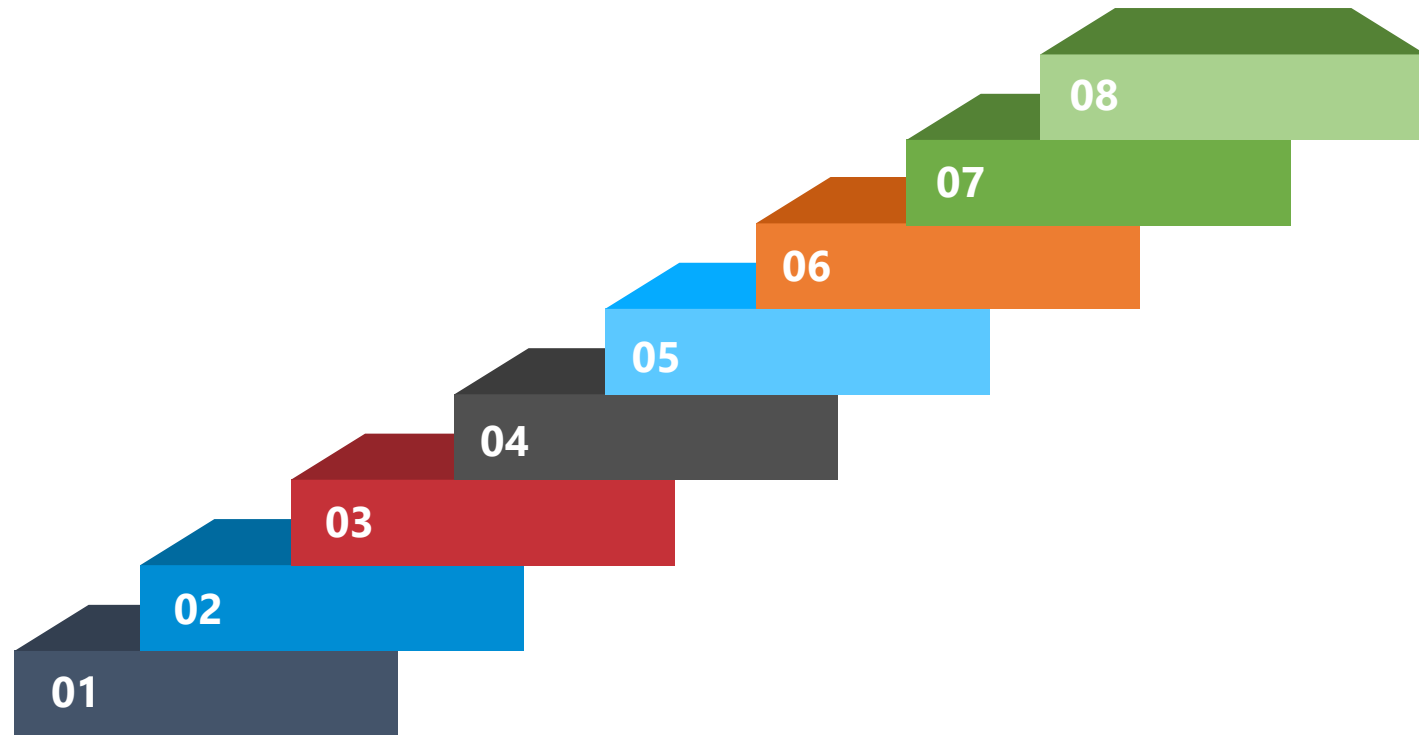
Best coding practices

- When possible, pull server info from properties file on server.
- Dynamic values should be pulled from the gel parameters.
- Make sure you write all your code inside a catch block.
- Put enough comments inside your script, so that the code is readable.
- Avoid excessive logging as it can slow the performance.
- Try to keep the hardcoding as minimum as possible.

REST APIs

Session Outline

- 01 Introduction
- 02 GEL Script Structure
- 03 Operations
- 04 XML Manipulation
- 05 Exception Handling
- 06 Limitations
- 07 Best Practices
- 08 Hands On Exercise



Hands on Exercise

Practice makes a man perfect



Summarize GEL Scripting

Introduction

What is GEL
Capabilities of GEL

Summarize GEL Scripting

- **Introduction**

- What is GEL
 - Capabilities of GEL

- **GEL Script Structure**

- **Most Common & WorkFlow Tags**

Summarize GEL Scripting

- **Introduction**

- What is GEL
 - Capabilities of GEL

- **GEL Script Structure**

- **Most Common & WorkFlow Tags**

- **Operations**

- DB Operations
 - Logging
 - JAVA Operations
 - File Operations
 - Emails
 - SOAP & REST in GEL

Summarize GEL Scripting

- **Introduction**

What is GEL
Capabilities of GEL

- **GEL Script Structure**
- **Most Common & WorkFlow Tags**

- **Operations**

- DB Operations
- Logging
- JAVA Operations
- File Operations
- Emails
- SOAP & REST in GEL

- **XML Manipulation**

Summarize GEL Scripting

- **Introduction**

What is GEL
Capabilities of GEL

- **GEL Script Structure**
- **Most Common & WorkFlow Tags**

- **Operations**

- DB Operations
- Logging
- JAVA Operations
- File Operations
- Emails
- SOAP & REST in GEL

- **XML Manipulation**

Exception Handling

Summarize GEL Scripting

- **Introduction**

What is GEL
Capabilities of GEL

- **GEL Script Structure**

- **Most Common & WorkFlow Tags**

- **Operations**

- DB Operations
- Logging
- JAVA Operations
- File Operations
- Emails
- SOAP & REST in GEL

- **XML Manipulation**

Exception Handling

Limitations

Summarize GEL Scripting

- **Introduction**

What is GEL
Capabilities of GEL

- **GEL Script Structure**

- **Most Common & WorkFlow Tags**

- **Operations**

- DB Operations
- Logging
- JAVA Operations
- File Operations
- Emails
- SOAP & REST in GEL

- **XML Manipulation**

Exception Handling

Limitations

Best Practices

Questions?

Hands-on with GEL Scripting, XOG and the REST API

Thank you for attending

Hands-on with GEL Scripting, XOG and the REST API



Phone

+44 844 736 2500



Email

ppmacademy@pemari.com



Website

www.pemari.com



Let us know how we can
improve!
Don't forget to fill out the
feedback forms!

References

Hands-on with GEL Scripting, XOG and the REST API

- <https://supportcontent.ca.com/>
- <http://www.gelscripting.com>
- <https://www.restapitutorial.com/>