BPI Academy Final Lab Exam

BPI005EXM, Revision 1.0

Blue Planet Inventory

Final Lab Exam

LEGAL NOTICES

Security

Ciena cannot be responsible for unauthorized use of equipment and will not make allowance or credit for unauthorized use or access.

Contacting Ciena

Corporate headquarters	410-694-5700 or 800-921-1144	www.ciena.com
Customer technical support/warranty		
In North America	1-800-CIENA24 (243-6224) 410-865-4961	Email: CIENA24@ciena.com
In Europe, Middle East, and Africa	800-CIENA-24-7 (800-2436-2247) +44-207-012-5508	Email: CIENA24@ciena.com
In Asia-Pacific	800-CIENA-24-7 (800-2436-2247) +81-3-6367-3989 +91-124-4340-600	Email: CIENA24@ciena.com
In Caribbean and Latin America	800-CIENA-24-7 (800-2436-2247) 410-865-4944 (USA)	Email: CIENA24@ciena.com
Sales and General Information	410-694-5700	E-mail: sales@ciena.com
In North America	410-694-5700 or 800-207-3714	E-mail: sales@ciena.com
In Europe	+44-207-012-5500 (UK)	E-mail: sales@ciena.com
In Asia	+81-3-3248-4680 (Japan)	E-mail: sales@ciena.com
In India	+91-124-434-0500	E-mail: sales@ciena.com
In Latin America	011-5255-1719-0220 (Mexico City)	E-mail: sales@ciena.com
Training		E-mail: learning@ciena.com

For additional office locations and phone numbers, please visit the Ciena website at www.ciena.com

Change History

Blue Planet Release	Revision	Publication Date	Reason for Change
	1.0	March 2023	Initial release.

Contents

Topic 1: Graph Model 6
Introduction 6
Reference Data 6
Objectives 6
Verification 7
Topic 2: UI Customizations 8
Introduction 8
Objectives 8
Topic 3: Rapid Path Search 11
Introduction 11
Reference Data 11
Objectives 12
Verification 15
Topic 4: TMF APIs 16
Introduction 16
Reference Data 16
Objectives 16
Verification 16
Topic 5: Data Ingestion Framework 17
Introduction 17
Objectives 18
Verification 20
Topic 6: Guided Operations 21
Introduction 21

Reference Data 21
Objectives 22
Topic 7: Connection Naming 26
Introduction 26
Objectives 26
Verification..... 26

Topic 1: Graph Model

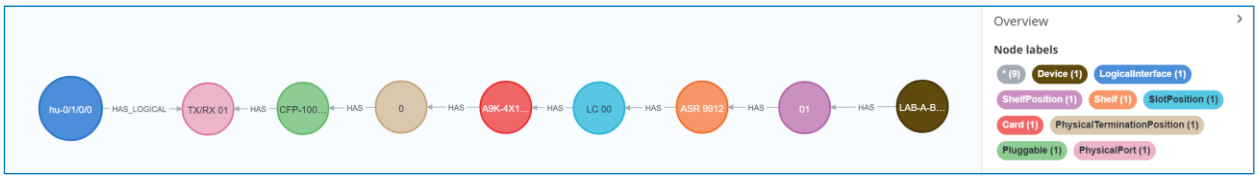
Introduction

In this task, you use the Metadata Modeller to add new equipment models and create new compatibilities to a logical interface. Your objective is to support the termination of 10 GB Ethernet logical connections on the new equipment. You add a new card and other equipment models to a Ciena OME 6500 7 Slot device and create the required compatibilities between the new equipment.

NOTE: Make sure to provide the names and all other object properties exactly as stated in the Objectives. This is important for verifying and grading your tasks.

Reference Data

Refer to the following diagram to recall how the relationships between logical and physical inventories are modeled in the graph database:



Objectives

1.1. Make sure that the LabProject Library is set as default.

1.2. Use the Metadata Modeller to add the following equipment. Make sure to provide the properties below exactly as stated, you can choose any valid values for other required properties. Also make sure you always select **No Family** for the Default Family.

- Card
  - Type: **Card**
  - ArchetypeName: **EXAM-CARD**
  - Position Used: **1**
  - Dimensions (HxWxD): **1.7 x 19 x 10**
- Four (4) Physical Termination Points
  - Type: **Physical Termination Position**
  - ArchetypeName: **EXAM-PTP**
  - ArchetypeInstance: **EXAM-PTP-0, EXAM-PTP-1, EXAM-PTP-2, EXAM-PTP-3**
- Pluggable
  - Type: **Pluggable**
  - ArchetypeName: **EXAM-PLUG**
- Physical Port
  - Type: **Physical Port**

- ArchetypeName: **EXAM-PP**
- ArchetypeInstance: **EXAM-PP 1**

1.3. Assure that the new card you created can be added to Slots 01-07 in the **OME 6500 7 Slot** shelf. Use the Archetype Management page to add compatibility relationships between the new Archetypes you created. Refer to the diagram in the task description for assistance.

1.4. Assure that you can terminate Cable Fibers on your new physical ports.

1.5. Assure that you can create Ethernet connections with **10 GB** channelization over that Cable Fiber.

1.6. Tag and deploy the changes to the **LabProject** Library.

Verification

- From the BPI UI, add the card EXAM-CARD to the device OME-6500-EXAM-1.
- Create a Cable Fiber connection from a port on this card to a port on ASR9001-EXAM-1.
- Create a 10 GB Ethernet connection over that Cable Fiber.

Topic 2: UI Customizations

Introduction

In this task, you need to customize the building summary page and the device summary page using the Generic UI Framework.

Objectives

2.1. Customize the fields for the building summary page. You need to add a custom field that will display the type of data center. To do that, create a Metadata item with the following parameters:

- Name: **Building Metadata EXAM**
- Priority: **101**
- Family: **No Family**

2.2. Use the created **Building Metadata EXAM** to add the custom field configuration. Group the field in the **Custom Building Details** group. The field should have the following restrictions:

- Field label: **Type**
- Database property name: **buildingType**
- Field Type: **DROPDOWN**
- Field options: **Cloud Computing, Co-Location, High Performance Computing, Enterprise, Edge Data Center**
- The field **should not** be mandatory, and it **should not** be stored as a versioned attribute.

2.3. Customize the device summary page. You need to add three custom fields to the device summary page. To do that, create a Metadata item with the following parameters:

- Name: **Device Metadata EXAM**
- Priority: **101**
- Family: **No Family**

2.4. Use the created **Device Metadata EXAM** to add the custom field configuration. Group the fields in the **Custom Device Details** group. Fields should have the following restrictions:

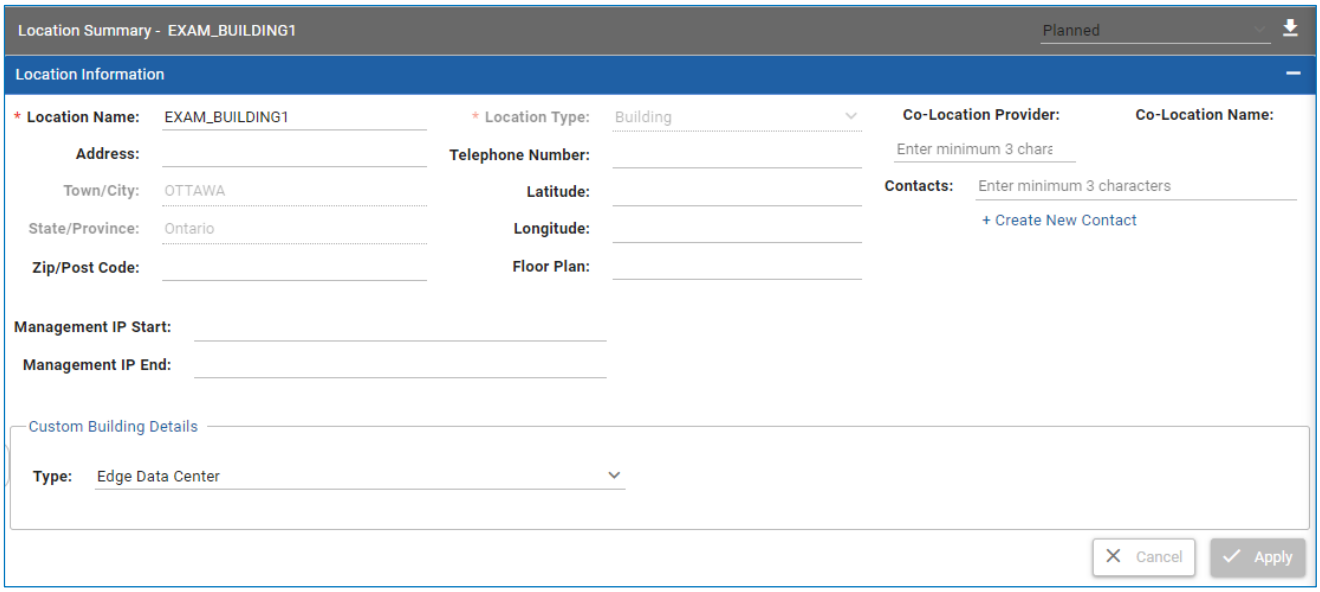
- CLLI field:
  - Field label: **CLLI**
  - Database property name: **equipmentClli**
  - Field Type: **TEXT**
  - The field **should not** be mandatory.
  - Field **should not** be stored as a versioned attribute.
- CLEI field:
  - Field label: **CLEI**
  - Database property name: **clei**
  - Field Type: **TEXT**

- The field **should not** be mandatory.
- Field **should not** be stored as a versioned attribute.

- Description field:
  - Field label: **Description**
  - Database property name: **description**
  - Field Type: **TEXTAREA**
  - The field **should not** be mandatory.
  - Field should be stored as a versioned attribute.
  - Versioned relationship type: **CONTAINS_PROPERTIES**

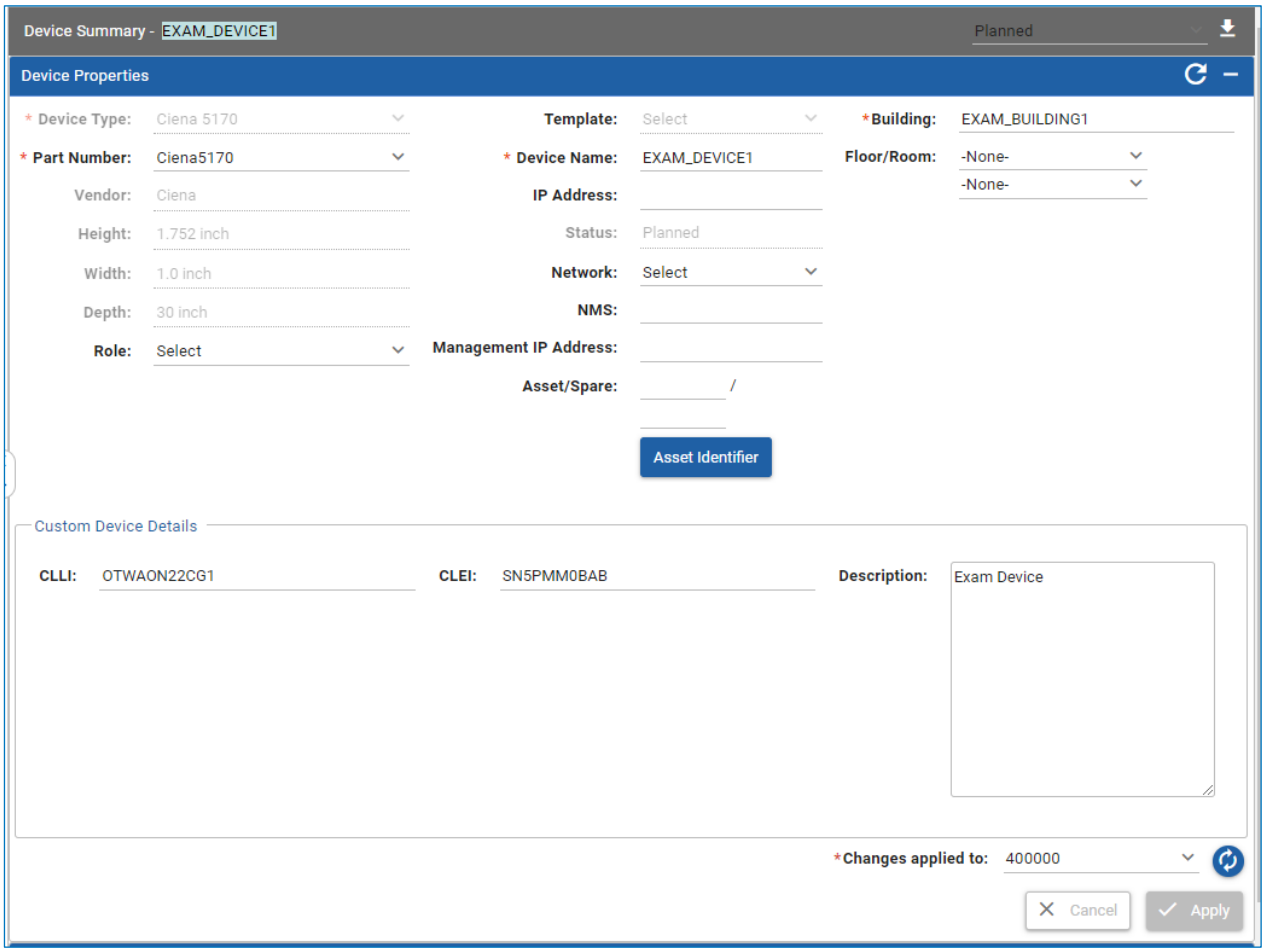
2.5. Create a new building in the location **Canada>Ontario>Ottawa**.

- Location name: **EXAM_BUILDING1**
- Type: **Edge Data Center**



2.6. Create a new Ciena 5170 device and place it in the **EXAM_BUILDING1** location.

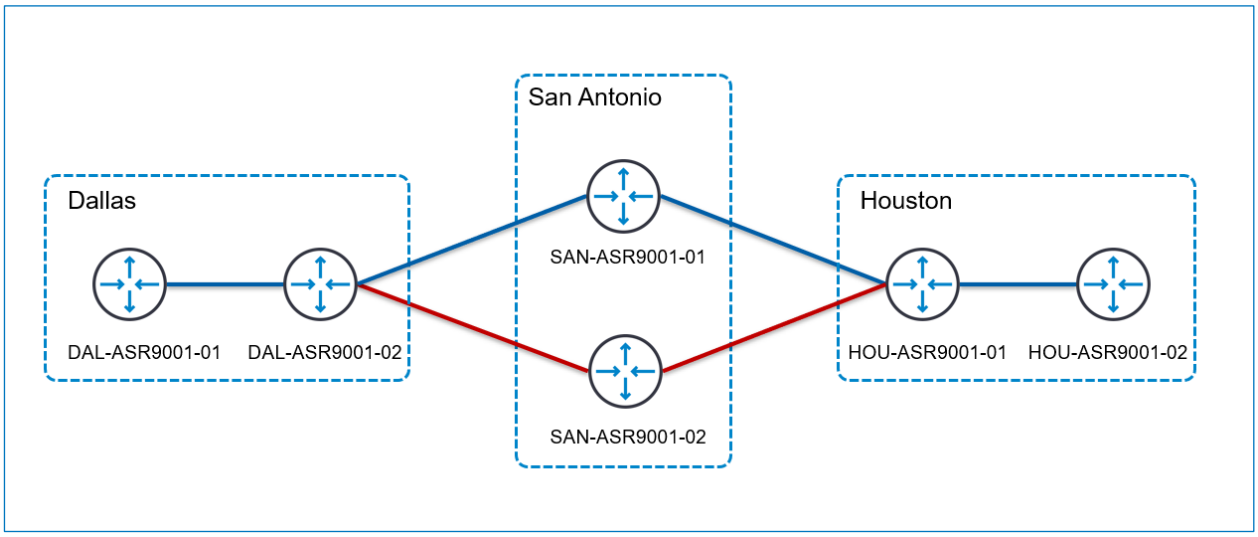
- Device type: **Ciena 5170**
- Device name: **EXAM_DEVICE1**
- Building: **EXAM_BUILDING1**
- CLLI: **OTWAON22CG1**
- CLEI: **SN5PMM0BAB**
- Description: **Exam Device**



Topic 3: Rapid Path Search

Introduction

Your lab topology consists of 6 routers interconnected with physical fiber cables as shown in the following figure.



There are two possible paths between DAL-ASR9001-01 on the left and HOU-ASR9001-02 on the far right. The object of this exercise is to write a custom Rapid Path Search (RPS) business rule that will filter out the bottom path (red) based on the names of the red physical connections in the database. Find the routers and their respective locations in the Scratch Pad of the BPI UI, for your convenience.

Reference Data

- Monitor the Neo4j log using the server script **tail_neo4j_log.sh**. Use | grep to filter on your specific log lines such as lines containing „exam:“

```
[bpadmin@bpi-pod03 ~]$ tail_neo4j_log.sh
[bpadmin@bpi-pod03 ~]$ tail_neo4j_log.sh | grep 'exam:'
```

- Use the following import statements in your **ProjectBusinessRuleBackup** class. Add additional import statements if required. VS Code autocomplete can be used to help with imports.

```
import org.neo4j.graphdb.Node;
import com.blueplanet.lab.routing.neo4j.rules.constants.ProjectKeys;
import com.blueplanet.lab.routing.neo4j.rules.debug.Debug;
import com.blueplanet.routing.neo4j.constants.Labels;
import com.blueplanet.routing.neo4j.core.AnalysisInformationEnriched;
import com.blueplanet.routing.neo4j.core.BusinessRule;
import com.blueplanet.routing.neo4j.io.RoutingException;
```

- To access the Neo4j Browser, use the following information
  - URL: **http://neo4j.lab:7474/browser**
    - Connect URL: **bolt:// neo4j.lab:7687**
    - Authentication type: **Username/Password**
    - Username: **neo4j**
    - Password: **drni**

Objectives

3.1. Clone the skeleton structure of your project from the Gitlab repository to your StudentPC from **git@gitlab.lab:studentXX/bpi-final-exam-rps.git** where XX is the number of your pod.

3.2. Clone the repository to your StudentPC, in the local directory **c:\Users\student\Workspaces\3_RPS**. Use git CLI commands or VS Code to achieve this task, then open the cloned repository folder with VS Code.

3.3. Later you can use your Gitlab account to access your repository through the browser and to monitor the CI/CD processes.

3.4. Create a new java class containing Project Keys that you will use in your project. The purpose of this class is to name the properties for use in your project. Add contents to this class at your discretion.

- Create a new folder named **constants** in **src\main\java\com\blueplanet\lab\routing\neo4j\rules**.
- Create a new java class named **ProjectKeys** in the folder **constants**.
- Enter your keys as static members, for example:

```
public static final String deviceName = "deviceName";
```

3.5. Create a new java class for your new business rule, named **ProjectBusinessRuleBackup**. The rule should check that for a given path segment, if there is a physical connection that has the word „BACKUP“ in its name, that route must be dropped from the results. Use the **AnalysisInformationEnriched** end node to achieve this.

- First create a new folder called **exam** in **src\main\java\com\blueplanet\lab\routing\neo4j\rules**.
- Class name should be **ProjectBusinessRuleBackup** in the exam folder.
- The class should extend the existing **BusinessRule** class.
- Override the **isSatisfied** method to complete this task. isSatisfied should take an **AnalysisInformationEnriched** object as a parameter and should throw a **RoutingException** exception.
- Identify physical connections by the node label **PhysicalConnection**.
- Do not manually enter strings as parameters for node properties, instead use keys from the ProjectKeys class that you created.
- To get an insight into the available properties, use the predefined Debug class which will output specific node properties and labels to the log file. Note that not all nodes have the same set of properties. An example of using Debug in your isSatisfied method, where analysisInfo is an example naming of your AnalysisInformationEnriched object:

```
Debug debug = new Debug(this, analysisInfo);
debug.show(true);
```

- Use git to propagate your code changes to your CI/CD instance which will build and deploy the plugin to the BPI server.

3.6. Create the required nodes and relationships in the Neo4j database for your new rule. Use the Neo4j Browser web UI to make changes to the database.

- Create a database metadata node describing your rule, so that the resulting node appears as the following (the identity value may differ):

```
neo4j$ match (r:BusinessRule) where r.rpsId = 113 return r
```

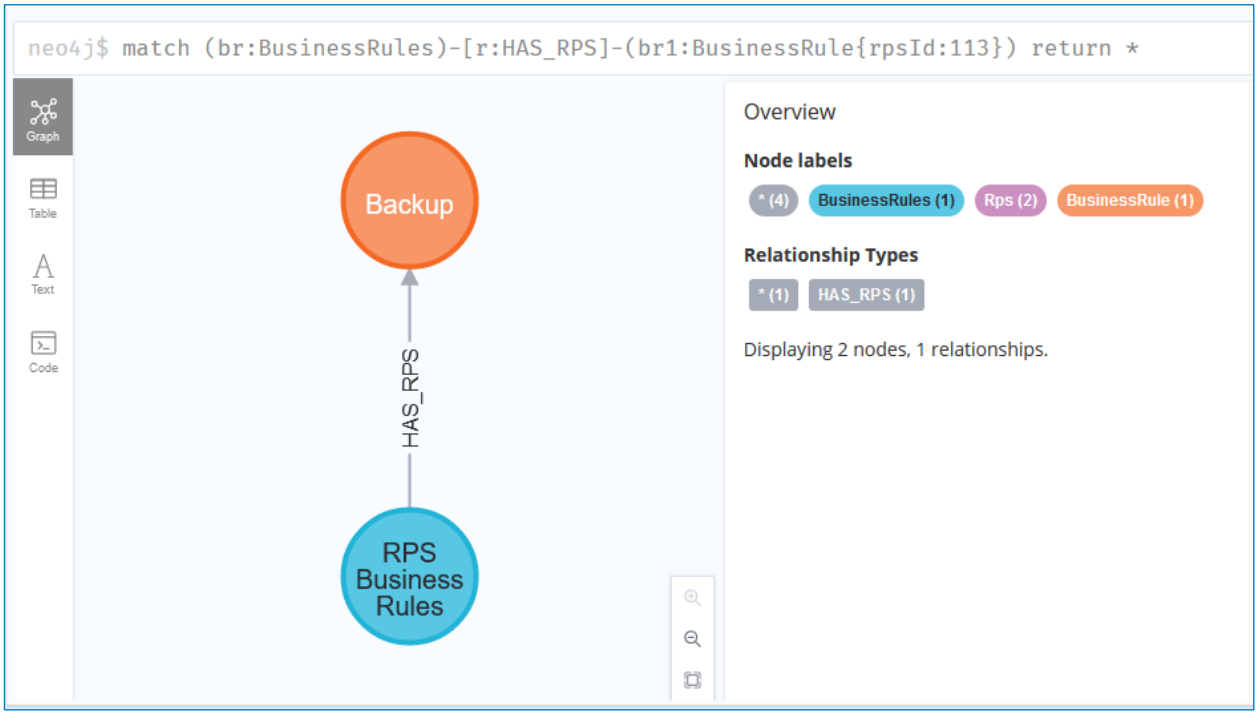
Graph | Table | Text | Code

r

```
1
{
  "identity": 50269,
  "labels": [
    "Rps",
    "BusinessRule"
  ],
  "properties": {
"commentsOnNok": "Backup router used",
"name": "Backup",
"weight": 50,
"className": "com.blueplanet.lab.routing.neo4j.rules.exam.ProjectBusinessRuleBackup",
"rpsId": 113,
"key": "Backup"
  }
}
```

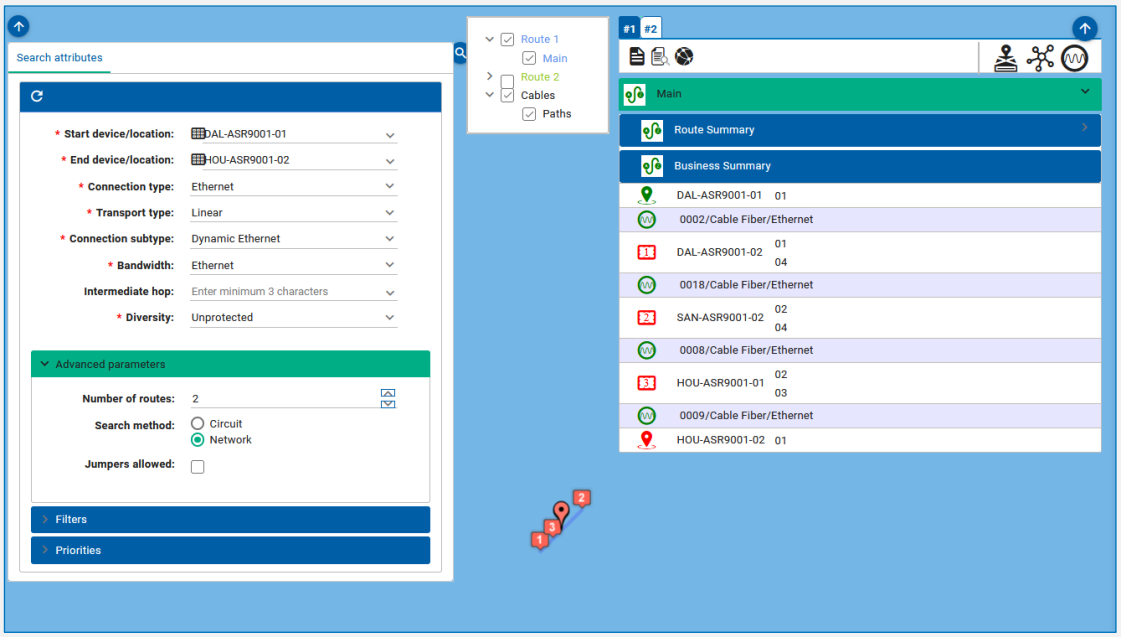
Started streaming 1 records in less than 1 ms and completed after 1 ms.

- Create a HAS_RPS relationship between your new rule and the BusinessRules node (rpsId=10), so that it appears as the following:



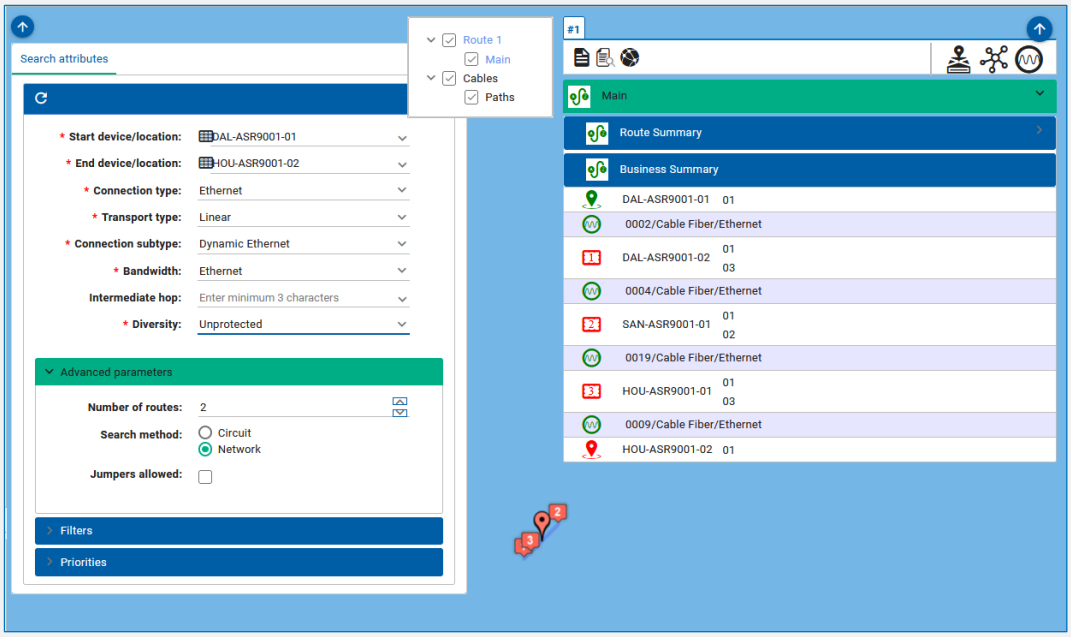
Verification

- Execute a search for Ethernet connections between DAL-ASR9001-01 and HOU-ASR9001-02 with a number of routes 2.



Note how RPS found two routes between the defined devices. One route goes over SAN ASR9001-01 and the other through SAN-ASR9001-02.

- Now from the BPI UI, change the name of both physical connections on the SAN-ASR9001-02 router in a way to add „BACKUP“ at the end of the connection name. For example: „0008/Cable Fiber“ to „0008/Cable Fiber BACKUP“.
- Repeat the same search. The result shows only one path, the path going through SAN-ASR9001-01.



Topic 4: TMF APIs

Introduction

In this task, you create API requests to the TMF639 Resource Inventory Management API to create new objects and relationships. You also create a new notification hub to monitor the notification events.

Reference Data

You can access the TMF API Swagger documents at *https://bpi.lab/blueplanet-inventory-tmf-api/swagger-ui/index.html*.

You can also reference the TMF REST API Technical Guide for sample payloads if needed.

You also might use the approach you used in the trainings, where you use TMF API requests to retrieve existing inventory objects, and use the response body as the template for your create/update requests. Make sure you consider which keys you need to exclude from that payloads.

Objectives

4.1. Create a new Postman Collection with the name **Exam TMF API**. Add all the requests that you create to that Collection.

4.2. Create an authentication request to collect the authentication token with read/write permissions. Provide the correct username, password, and other parameters needed for the authentication request. You can refer to the TMF API Technical Guide for the required parameters. Use the retrieved token for authenticating your next requests.

4.3. Use a TMF API request to create an internal REST hub for the ResourceCreationNotification and ResourceAttributeValueChangeNotification event types.

4.4. Use a TMF API request to create a new Room in the **DAL-LAB-LOC-01** location. Name the room **ROOM-EXAM-1**.

4.5. Use a TMF API request to place the existing device **ASR9001-EXAM-1** to **ROOM-EXAM-1**.

4.6. Use a TMF API request to create a Cable Fiber connection between any of the ports on the **ASR9001-EXAM-1** router and the **LRPP-EXAM-1** long-reach patch panel in your lab. Name the connection **CF-EXAM**.

Verification

- Verify that the room and the connection you created can be seen in the Graph DB or in the UI. Make sure you consider which views the objects you created will be visible in.

Topic 5: Data Ingestion Framework

Introduction

In this task, your goal is to create two Cable Fiber connections in a planned perspective using Data Ingestion Framework. To achieve that, you will need to write and deploy a custom DROOLS rule which you will use in the Data Ingestion Source configuration to process the event messages from the source.

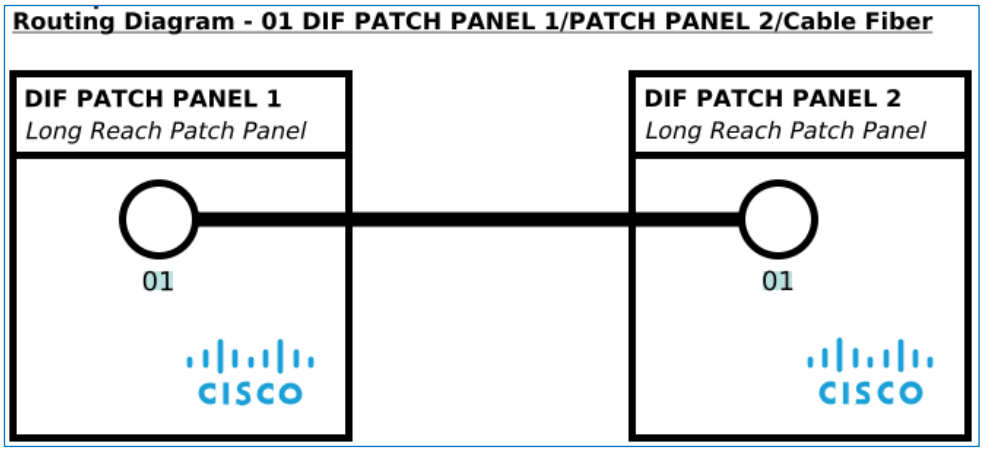
Clone the git repository **git@gitlab.lab:studentXX/bpi-final-exam-dif.git** where XX is the number of your pod. Clone the repository to your StudentPC, in the local directory **c:\Users\student\Workspaces\5_DIF**. Use git CLI commands or VS Code to achieve this task, then open the cloned repository folder with VS Code. You will find three files in the repository:

- **CreateCableFiberConnection.drl** – DRL file for the custom rule that needs to be implemented. You will need to modify this file to provide the correct implementation for the rule. (Instructions on how to modify this file will be provided in the following steps).
- **ExamDIF.postman_collection.json** – Postman collection which you will use to create an authentication token, create a Data Ingestion Source configuration, and send event messages to test the implemented changes.
- **ExamDIF.postman_environment.json** – Used to import environment variables.

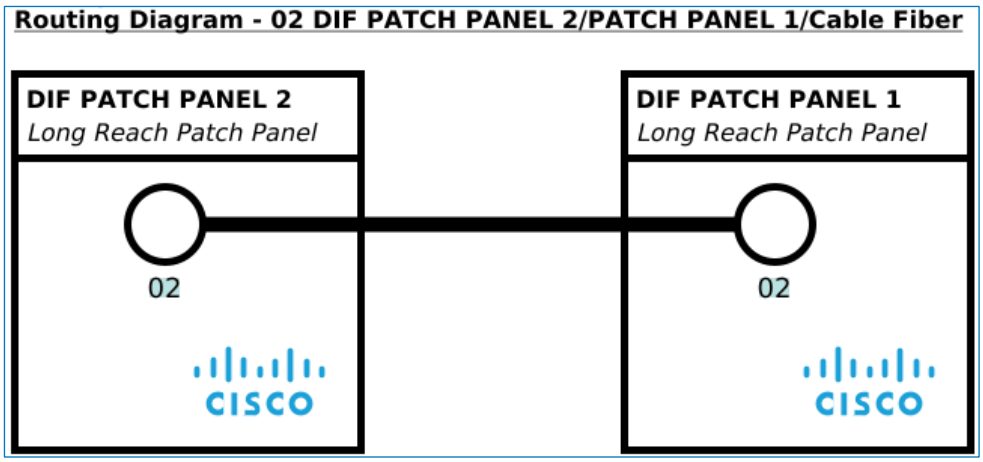
As part of the task, you will be asked to modify the collection and as part of the solution, you will need to export the modified postman collection and replace the old file with the newly exported file.

As part of your task, you have to modify the DROOLS rule to enable the creation of Cable Fiber connections for Long Reach Patch Panels through the Data Ingestion Framework. In the provided Postman collection, there are two requests that, when executed, should start the process to create the Cable Fiber connections displayed on the following routing diagrams.

- **Create Cable Fiber Connection 1**:



- **Create Cable Fiber Connection 2:**



As a result, you will need to execute those two requests to create the Cable Fiber connections.

NOTE: For testing purposes, you can use the request **Cable Fiber TEST.** This request uses different devices (names of the devices can be found in the request) and modifications regarding those devices will not be counted toward the final scoring.

For the purpose of the exam, you can assume that the names of the devices and connections are unique.

Objectives

5.1. Import the postman collection **ExamDIF.postman_collection.json** and postman environment configuration **ExamDIF.postman_environment.json** into the Postman desktop application. Set the **ExamDIF** environment as the current working environment.

5.2. Open the request **Ingestion Token Request**. The URL for this request is empty. Type in the correct URL and execute the request. The base URL is ***https://bpi.lab***.

NOTE: When the request executes correctly, the token should be saved as an environment variable. Check to make sure that it is saved correctly.

5.3. Create a new Data Ingestion Source using the **Create DIF Source** request. You will need to modify the body of the request to configure the following criteria:

- Data Ingestion Source name: **ExamDataSource**
- Applied to perspectives: **2 (Planned perspective)**
- Uses DROOLS rule: **Create Cable Fiber Connection**

5.4. Execute the request.

NOTE: If you executed a request but the configuration was wrong and you would want to change it, you can generate a new Data Ingestion Source with a different name and make sure that the proper URL is used in other requests which are used to send event messages through that Data Ingestion Source. Also, make sure that the configuration with the new name is properly saved before exporting the collection.

5.5. Open BPI UI and create a new network order with the sub-type Create Connection. Once the order is created, copy the drnId of the order and save it in the postman ExamDIF environment variable **orderRef.**

5.6. Save the changes made to the collection and export the ExamDIF collection and the ExamDIF environment.

5.7. Replace the files in the git repository with newly exported files and push the changes to the remote git repository.

NOTE: If you have already cloned the bpi area in previous tasks, use that area. If you have not yet cloned the bpi area, follow the next step.

5.8. From the Blue Planet extension, BP systems, clone the BPI area from the bpi.lab server to your Student PC. If the server is not listed in Blue Planet Systems, add the server with the name bpi.lab and DNS bpi.lab. Use **exam** as the branch name. If your cloning fails, check that the SSH keys are properly set up. If required, exchange SSH keys with the server.

5.9. Copy the **CreateCableFiberConnection.drl** file into the appropriate folder in the cloned area.

5.10. Open the file in VS Code. The skeleton for the Drools rule is already provided.

NOTE: For each of the following steps that ask you to modify the DROOLS rule, there is a section in the provided file where you need to write the code marked with START/END comments. Replace the /*code here*/ comments in the file with the appropriate code or the appropriate values.

5.11. Write the condition part of the rule which will match only the events that create the resource com.bp.inv.metamodel.PhysicalConnection.

5.12. In the consequence part of the rule, write the code to extract the JSON object **resource** from the event message into the variable **connectionData**.

5.13. In the code, when creating a connection node, set the query and query parameters for the matching connection node in the database. For the purpose of the exam, you can assume that the name of the connection is unique.

5.14. In the code, when creating a connection node, extract the archetype name from the event message and set the query to the return archetype instance drnId value based on the archetype name.

5.15. In the code, when creating a connection relationship, set the necessary query and query parameters to match the connection node as the source node of the relationship.

5.16. In the code, when creating a connection relationship, set the necessary query parameters to match the target node of the relationship.

5.17. In the code, when creating a connection relationship, set the correct relationship type.

5.18. In the code, when creating a connection relationship, set correct relationship properties.

5.19. Save the changes and onboard the rule to the database. Reset the DB context and the DROOLS session.

5.20. Execute the requests **Create Cable Fiber Connection 1** and **Create Cable Fiber Connection 2** from the postman collection.

5.21. Replace the files in the git repository(from the introduction of this topic) with the modified files:

5.21.1. Replace the old file **CreateCableFiberConnection.drl** from that repository with the modified file **CreateCableFiberConnection.drl** that contains the solution

5.21.2. In the Postman applications, save all the changes to the requests in the ExamDIF collection. Export the ExamDIF collection as Collection v2.1 and name the collection **ExamDIF.postman_collection.json**. Replace the old **ExamDIF.postman_collection.json** file in the git repository with the newly exported collection file.

5.21.3. In the Postman application, export the ExamDIF environment (the option to export the environment is only available from the tab of the opened environment). Name the exported file **ExamDIF.postman_environment.json.** Replace the old

ExamDIF.postman_environment.json file from the git repository with newly exported environment file.

5.22. Commit the changes to the main branch and push the changes to the remote repository.

Verification

- Execute the following query in Neo4j to search for the Data Ingestion Source node:

```
MATCH (n:IngestionDataSource) WHERE n.name="ExamDataSource" RETURN n
```

- Execute the following query in Neo4j to check if the Cable Fiber connections were created successfully:

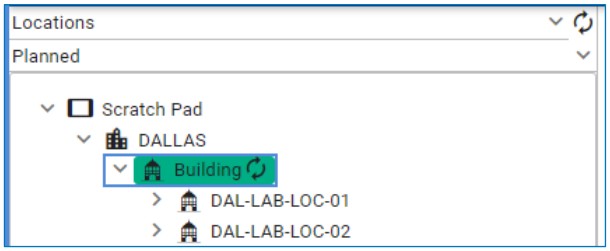
```
MATCH g=(n:PhysicalConnection)-[:HAS_CONNECTION_COMPONENT]-
   () where n.name="01 DIF PATCH PANEL 1/PATCH PANEL 2/Cable Fiber"  OR
       n.name="02 DIF PATCH PANEL 2/PATCH PANEL 1/Cable Fiber" RETURN g
```

Topic 6: Guided Operations

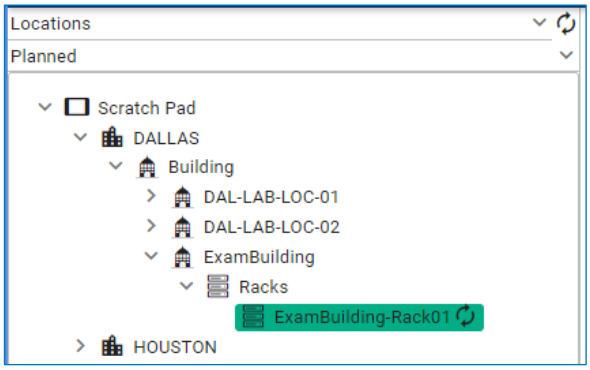
Introduction

For this exercise, you will use an existing BPMN workflow definition that you will find on your StudentPC. The workflow has an accompanying service task, implemented as a Groovy script in through the VS Code IDE Blue Planet Plugin. The service task creates a new building with a name of your choice, in a city of your choice. Your tasks will be to first deploy the workflow definition to your server instance and then modify the existing service task to implement additional functionality, namely to programmatically create a new rack in the newly created building.

The initial state of the buildings in, for example, Dallas, TX is the following:



The unmodified groovy script will simply create the building in the given city. However, after you put in place the code modifications from this exercise, a new rack will also be created. While executing the workflow from the UI, if you choose Dallas as your city and enter the name **ExamBuilding**, a rack with the name **buildingName + '-Rack01'** should be created, so your tree should look like the one in the following figure.



Reference Data

- Monitor the **catalina.out** log file using the server script **tail_web_log.sh**. Use „**| grep**“ to filter on your specific log lines such as lines containing „**exam:**“

```
[bpadmin@bpi-pod03 ~]$ tail_web_log.sh
[bpadmin@bpi-pod03 ~]$ tail_web_log.sh | grep 'exam:'
```

This is an example of a debug output from the catalina.out log file. This assumes that you implemented such log lines in proper places in your **execute** and **createRacksInBuilding** methods:

```
[bpadmin@bpi-pod03 ~]$ tail_web_log.sh | grep exam
ERROR: 2023-01-30 11:07:31,176 : TopicSubscriptionManager bpi_tasks 1 : bpi.appserver.extensions.ExamServiceTask : exam: Executing Create Building & Racks Service Task
ERROR: 2023-01-30 11:07:32,764 : TopicSubscriptionManager bpi_tasks 1 : bpi.appserver.extensions.ExamServiceTask : exam: Created building with name: ExamBuilding
ERROR: 2023-01-30 11:07:32,766 : TopicSubscriptionManager bpi_tasks 1 : bpi.appserver.extensions.ExamServiceTask : exam: Starting creation of racks
ERROR: 2023-01-30 11:07:32,816 : TopicSubscriptionManager bpi_tasks 1 : bpi.appserver.extensions.ExamServiceTask : exam: creating a rack with: RackVO [rack=null, rackName=ExamBuilding
-Rack01, rackType=UIEnumeration [isDefault=false, label=42 RU 19" Rack, value=1021], aisle=null, bay=null, title=null, modelNumber=null, location=UIInventoryObject [label=ExamBuilding
, main=com.blueplanet.inventory.rest.model.common.BaseIdentifiableObjectWithTimestampImpl@46a46a6[objectTimestamp=<null>,objectId=258175176744826213,objectClassId=101], today=null, st
atus=null, toolTip=null, icon=null] , additionalRackAttrsRelsVO = , frontAccessOnly= false, backAccessOnly= false]
ERROR: 2023-01-30 11:07:37,203 : TopicSubscriptionManager bpi_tasks 1 : bpi.appserver.extensions.ExamServiceTask : exam: Rack created.
```

- To get to the shell of the web-0 container, use the custom **enter** script:

```
[bpadmin@bpi-pod03 ~]$ enter web-0
root@web-0:/dev/shm#
```

Objectives

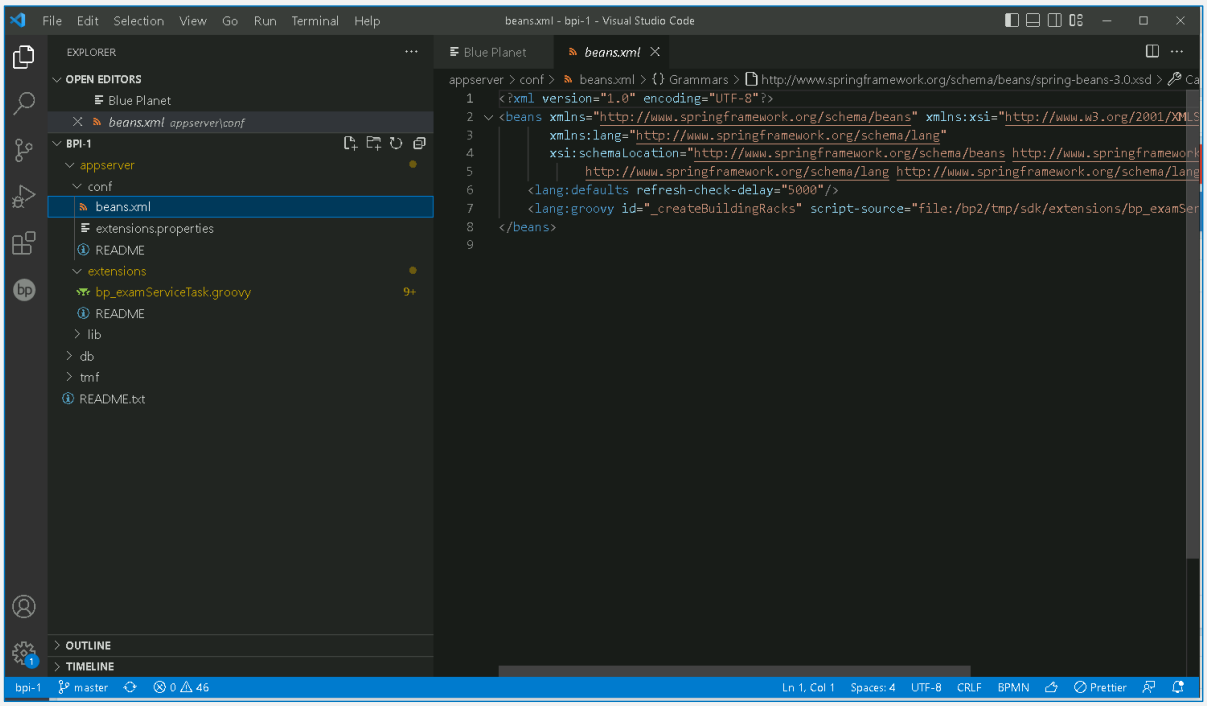
7.1. From your Student PC, open a new VS Code window. Make sure all other VS Code windows are closed and you have only one VS Code instance open.

NOTE: If you have already cloned the bpi area in previous tasks, use that area. If you have not yet cloned the bpi area, follow the next step.

7.2. From the Blue Planet extension, BP systems, clone the BPI area from the bpi.lab server to your Student PC. If the server is not listed in Blue Planet Systems, add the server with the name bpi.lab and DNS bpi.lab. Use **exam** as the branch name. If your cloning fails, check that the SSH keys are properly set up. If required, exchange SSH keys with the server.

Verification

Your cloned area should look as in the following figure.

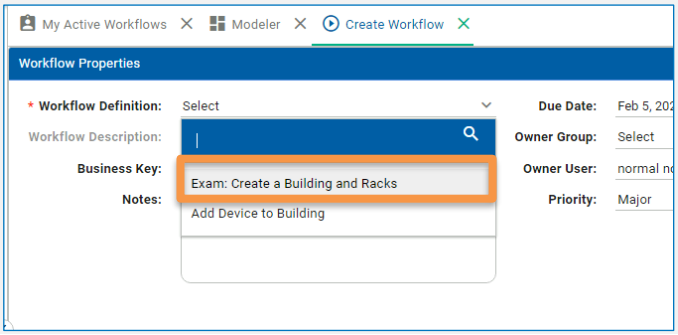


NOTE: You should have the **beans.xml** and **bp_examServiceTask.groovy** files in place.

7.3. Deploy the prepared BPMN workflow definition to your BPI server instance. The method of deployment is arbitrary. The workflow definition is stored in your Student PC, in **C:\Users\student\Workspaces\6_GO\bp_examCreateBuildingRacks.bpmn.**

Verification

When the workflow definition is deployed, you should be able to see it in **Guided Operations > Create Workflow**.



7.4. The provided file **beans.xml** is misconfigured. Find the error and correct it. Consult the groovy script to find out what might be misconfigured. In the current state, your service task would not execute properly.

NOTE: Remember to save file changes in VS Code before onboarding the files to the server.

7.5. In the **bp_examServiceTask.groovy** file, add your code additions and modifications that will enable the creation of a single rack in the building. If you examine the structure of the class within the file, you will notice that the execute method is the place where the new building is created. Place your **createRacksInBuilding** method call at the end of the execute method.

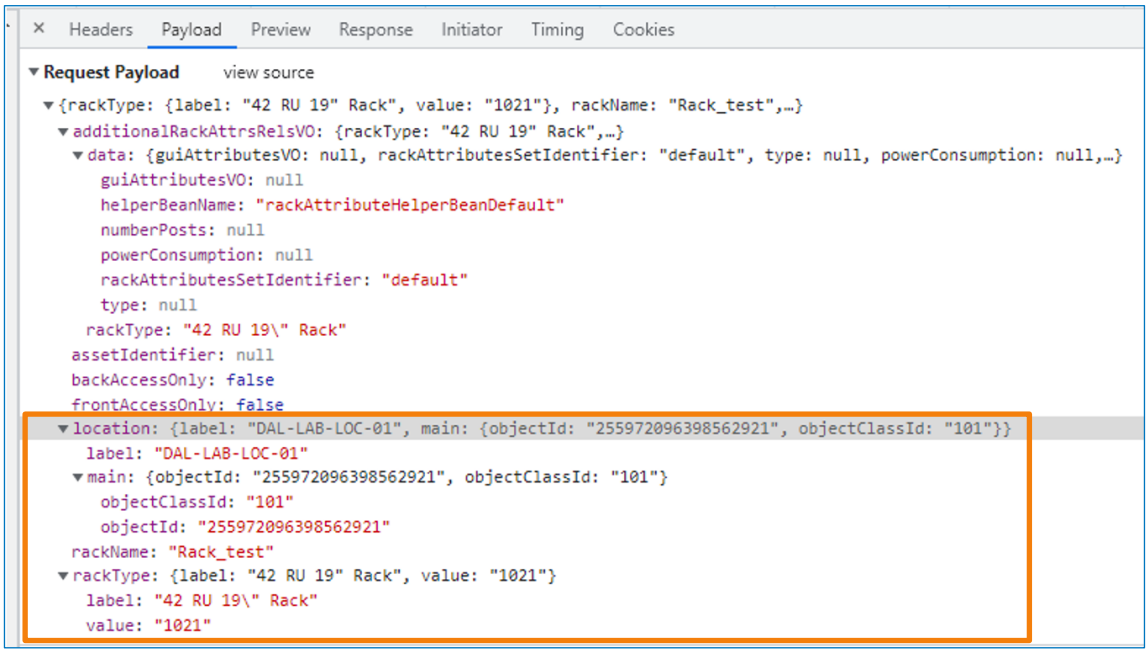
- Uncomment the following lines of code before you implement the **createRacksInBuilding** method.

```
            //exam task: uncomment the following 2 lines and create the required private method
            //logger.error("exam: Starting creation of racks");
            //createRacksInBuilding(buildingName, buildingId, sessionContext)



        } catch(Exception e) {
            throw new BpwmTaskException("exam: Failed to execute task : " + externalTask.getActivityId(), e);
        }
        return taskOutputParams;
    }
    //private void createRacksInBuilding(String buildingName, BigDecimal buildingId, SessionContext sessionContext) {
    //}

```

- Implement the **createRacksInBuilding** method. Use the predefined **rackBusiness** object to create the rack but before you can execute the **rackBusiness.createRack** method, you will have to build several other prerequisite objects, namely:
    a. RackVO
    b. UIInventoryObject
    c. BaseIdentifiableObjectWithTimestamp
    d. UIEnumeration

- The objects should have, at minimum, the following information, to be able to create a rack:
    a. rackName = (the buildingName provided in the UI form)+“-Rack01“
    b. rackType = "42 RU 19\" Rack", value = 1021
    c. location = label: {buildingName}, main: {objectId: (Id of the building), objectClassId: 101}

NOTE: Observe the following example output of one successful rack creation from the browser developer tools to get an idea of what properties should be provided. For more information on the required objects, refer to the javadocs for RackBusiness and RackVO classes.

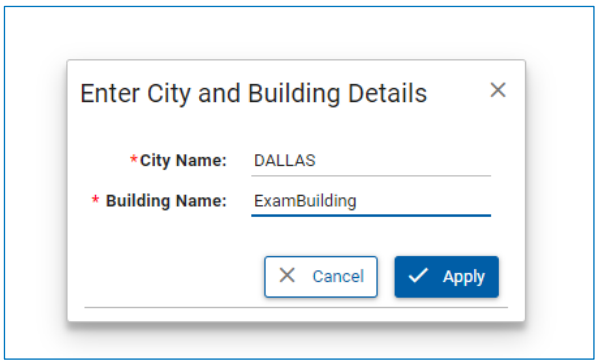


- When finished with the code modifications, onboard the changes to the server.

NOTE: If major changes were introduced, reset the web-0 container at your discretion. Observe the **catalina.out** log for compilation or other errors with regard to your new code.

7.6. Create a new workflow instance from the server UI and complete it so that your service tasks are triggered and executed.

- Create an instance of **Exam: Create a Building and Racks**
    a. For Business Key enter **exam**
- Claim the user task, run the task, and enter this exact information (the city in capital letters):

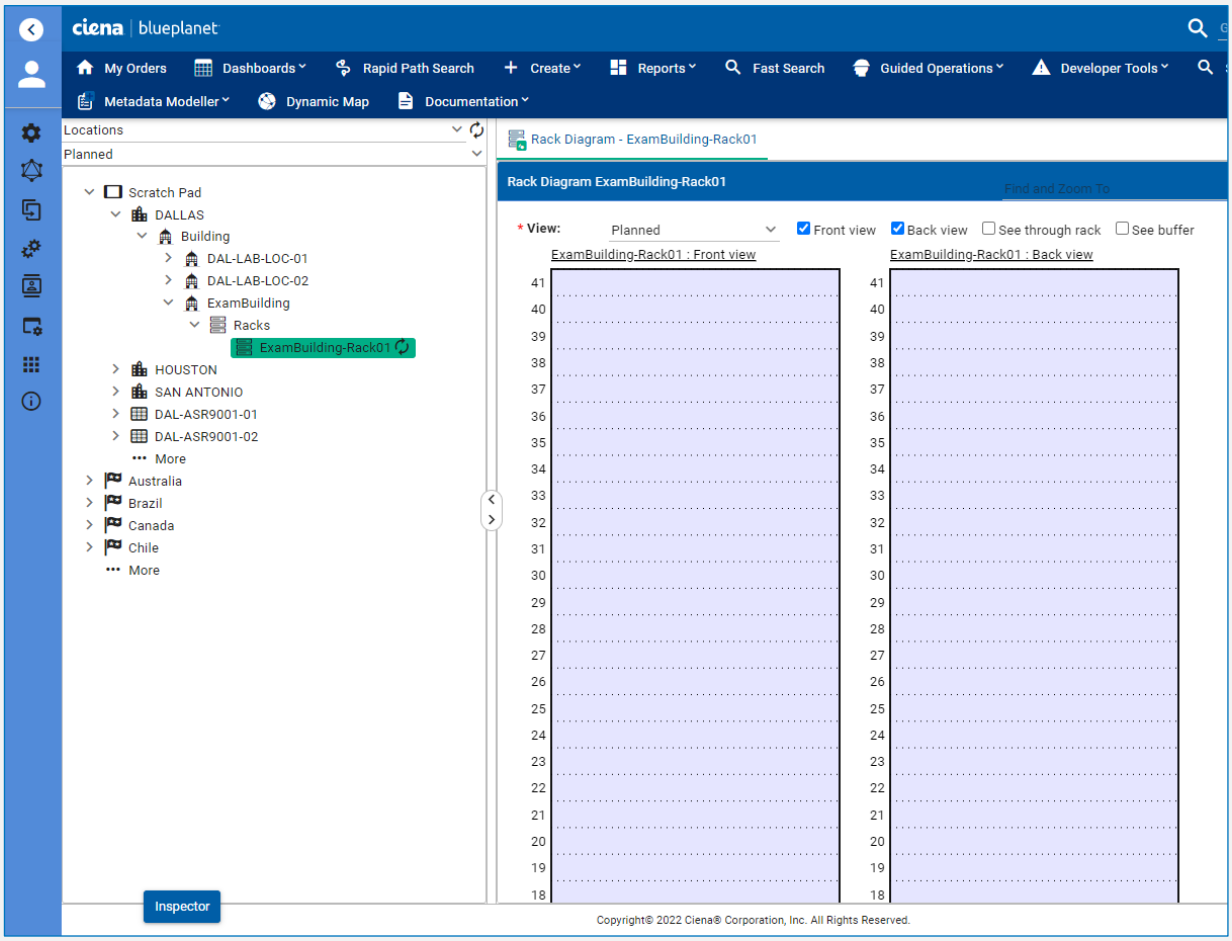


- Complete the user task. This will trigger your service task.

NOTE: This is a good time to monitor the log file.

Verification

- After the workflow finishes successfully, you should have the new building and the new rack created in the Dallas location.



Topic 7: Connection Naming

Introduction

In this task, you extend the business logic for naming connections. You deploy a new Groovy script, which will update the connection naming logic to include the connection type and the device ID in the connection name, for example 0001/Ethernet-10GE/CLLI-A/CLLI-Z.

Objectives

NOTE: If you have already cloned the bpi area in previous tasks, use that area. If you have not yet cloned the bpi area, follow the next step.

6.1. From the Blue Planet extension, BP systems, clone the BPI area from the bpi.lab server to your Student PC. If the server is not listed in Blue Planet Systems, add the server with the name bpi.lab and DNS bpi.lab. Use **exam** as the branch name. If your cloning fails, check that the SSH keys are properly set up. If required, exchange SSH keys with the server.

6.2. Create a new **ConnectionNaming.groovy** script, and place it in a proper folder so that the Groovy script will extend the current naming logic.

6.3. Update the **getIdValue** method, so that it follows the following logic:

- Check the value of the **equipmentClli** property of a node.
- If the property value is not empty, return that value.
- If the property does not exist, check the value of the **deviceName** property.

6.4. Update the **generateName** method to retrieve the connection type from the connectionVO. Refer to the javadocs for the connectionVO class to find the method that achieves this.

6.5. Update the **generateConnectionName** method to accept the connection type and add it to the generated name string.

6.6. The connection name should be generated in the following form:

- [****]/[connection type-connection bandwidth]/[A-end device ID]/[Z-end device ID]
    - Example: 0001/Ethernet-10GE/CLLI-A/CLLI-Z

6.7. Make sure to configure the appropriate file to register the connectionNaming bean.

6.8. Save all changes and push them to the Asset Manager.

Verification

- From the BPI UI, create a new 10 GB Ethernet connection between devices ASR9001-EXAM-1 and ASR9001-EXAM-2. The generated connection name should be **0001/Ethernet-10GE/CLLIA/ASR9001-EXAM-2** (the prefix number can be different, depending on the number of connections you created). You can ignore any errors received in the UI. If the connection name matches the objective, you completed the task successfully.