Kuwaiba Open Inventory User's Manual

Neotropic SAS

27.07.2016

System Version **1.0**
Visit **kuwaiba.org** for documentation, latest updates and upcoming events

Contents

Document History

Date	Comments
September 28th 2010	First issue shipped with version 0.1.1
November 26th 2010	Update to cover the new features in 0.2
December 26th 2010	Update to cover the new features in 0.2.1
January 18 th 2001	Changes in version 0.3 alpha
February 3 rd 2011	Changes in version 0.3 beta
March 13 th 2011	Changes in version 0.3 beta 2
May 16th 2011	Changes in version 0.3 stable (the clear button in the graphical query editor
May 23rd 2012	Adapted to version 0.4
October 23rd 2012	Adapted to version 0.5
June 4 th 2013	Added documentation for Pools module
June 12 th 2013	Added documentation for Data model Manager module and some other minor changes
January 19 th 2015	Adapted manual for version 0.7
November 6 th 2015	Added documentation about bulk upload, software asset management and detailed physical connections
July 27th 2016	Adapted to Kuwaiba version 1.0. LaTeX is now used instead of LibreOffice to create the documentation.

License

Disclaimer

Introduction

Kuwaiba sees an inventory system as a living entity, not growing only in terms of size, but also in structure and intelligence. The main reason is that business requirements change constantly and therefore, the application must ready to respond to new scenarios. One of the key concepts that can help you unlock the potential of Kuwaiba is the **data model**. It provides a simplified representation of the network and the business from an operational point of view. It can be seen as the skeleton that supports the application, but a skeleton from which you can add, remove and change elements as you go. Later in this document you will be able to see what tools you can use to manage it. For now, just keep in mind that the better you design your data model and the more you get to know it, the more you will take advantage of the application.

Having said that, you will find four types of resources in a typical data model:

- **Physical:** Equipment, pipes, cables, fiber optics, facilities, parts and in general every physical asset from a port to a building.
- **Logical:** These are all the resources related to non-tangible technology assets. In this group fits timeslots, virtual circuits, VLANs, disk space, available bandwidth, etc.
- **Other Non-physical:** mostly software-related assets, such as licenses or virtual machines.
- **Administrative:** These are all those related to administrative tasks, human resources or commercial management. Customers, their services, SLAs (and related parameters like availability or throughput), sales and technical staff assigned to those services, vendors and states belong to this category.

The Kuwaiba desktop client is a set of views (trees, topologies, editors) that allow to put together these elements based on business rules and user-defined models. Kuwaiba extends the concept of **CMDB** (Configuration Management Database, a place where you store objects that can hold configuration information or be subject to configuration themselves -so called Configuration Items- and their relationships) and enables you to perform network design tasks, support capacity management and provisioning workflows and assist field and customer service teams to improve response times.

Kuwaiba helps you model your network according to your needs, no matter if you're an ISP, a carrier or just a guy with a large (or small!) IT infrastructure to manage. It's open source, under active development and new models are added every release. You can contribute to the project by providing technical insight on a particular technology, testing, translating or just sending your feedback through forums[1] and mailing lists[2].

[1] Forums https://sourceforge.net/p/kuwaiba/discussion/
[2] mailing lists https://sourceforge.net/p/kuwaiba/mailman/

Connection to the Server

The first thing you will see when opening the client is the window in the figure 1. The default user and password are **admin/kuwaiba**.

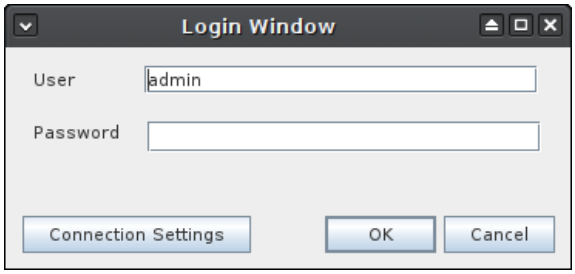


Figure 1: Authentication window

The default connection settings should be enough if the server is running on the same computer the client is. If that's not the case, open the Connection Settings window (figure 2).

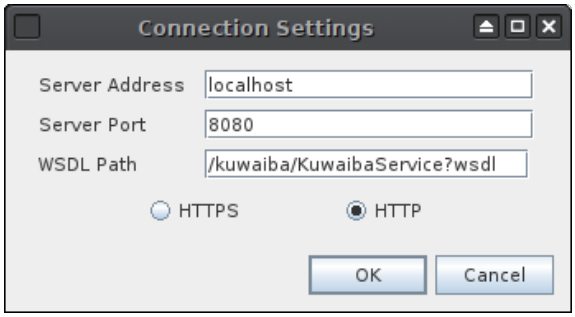


Figure 2: Connection Settings window

- **Server Address** Refers to the server IP address or canonical name.
- **Server Port** is the port Glassfish (the application server) is listening to.
- **WSDL Path** is the path within the application server the web service interface definition can be found. Usually this value should remain unchanged.
- Protocol is the transport protocol to be used. By default is HTTP, but is highly advisable to request your administrator to setup a secure connection, otherwise your credentials will be transmitted in plain text over the network.

Except for the password, the last successful settings will be saved upon clicking OK.

Important

If you are unsure if the server is reachable from your location, open a browser and type the address: **http://[server_address]:[server_port]/[wsdl_location]**

You should see a large XML document.

Troubleshooting

- For a **Can't contact backend** error, check the Administrator's Manual Troubleshooting section.

- If you get a **Connection refused** error, check the connection settings and verify that the server is reachable and there isn't a firewall blocking the traffic to it.

Once you are logged in, you will see only the dashboard page and a toolbar (figure 3).

Figure 3: Main toolbar

The toolbar contains the most frequently used tools. Here is an overview of what cab you do with them:

	Search objects with the Query Manager
	Refresh the current view
	Refresh local cache
	Physical/Rack View of an object
	Create automation tasks (beta version)
	See the changes made to inventory and application objects
	Manage users and groups
	Change the data model
	Manage how objects can be created inside others
	Create new list types
	Freely design network topologies
	Main tree used to explore physical assets
	Create and manage objects that don't fit in the navigation tree
	Manage client, services and resources associated to them

Table 1: Toolbar items

Data Model Manager

One of the key features of Kuwaiba is that it is completely object-oriented[3]. It means that every business (Router, City, Port) and application (users, types) element is represented by an **Object** in the application and these objects are in turn product of an reality abstraction called **Class**. Likewise, every attribute is a **Field** in a class. The set of classes, attributes and relationships between them is called data model. There's a default data model, but you can customize it depending on your needs by adding, removing and modifying classes. To achieve this, use the Data Model Manager module (figure 4). The data model is represented as a tree because it's a hierarchical

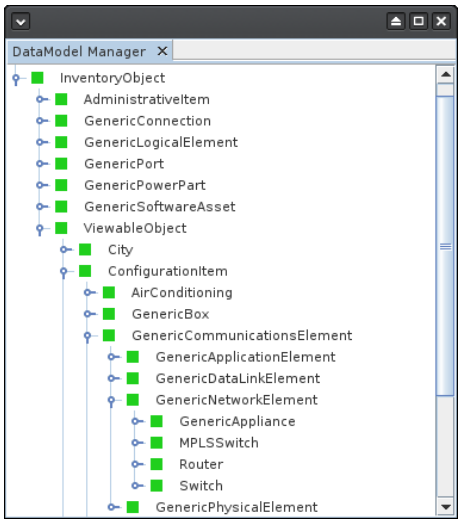


Figure 4: Part of the data model tree

structure. Technically, it's a class hierarchy[4]. The top of the hierarchy (**InventoryObject**) is the most general type of element in the data model and its subclasses represent all the possible elements that will be treated as inventory assets. As you dig deeper into the tree, the classes become more and more specialized and each level inherits the attributes of the parent classes. This kind of structure has two purposes: First, it helps you to organize your classes based on what characteristics they have in common. Secondly, as you will see later in this manual, you can apply operations over top level classes, and they will be propagated to all subclasses. Another root of the data model tree is **GenericObjectList**, and its subclasses are all possible list types (see more details on the subject in the chapter **List Type Manager**).

> **Important**
> The **Properties** window allows you to modify the attributes of a selected object in a tree, list or view. If not already open, it's available from the Windows → Properties menu.

The properties of a class can be edited by using the **Properties** window, selecting the class from the tree (see figure 5). The property sheet is divided in two sections:

- **General:** Contains the intrinsic properties of the class: **name** (can contain only letters and numbers with no special characters or blank spaces). The **display name** of the class, that's how the will be displayed everywhere else (useful for internationalization purposes, for example) and can contain any kind of UTF-8 character. A **description** (useful to document the data model). If the class is **abstract** (abstract classes cannot be instantiated, they're only used to give consistency to the data model). The attribute **countable** is not used currently,

[3]Object-oriented Programming https://en.wikipedia.org/wiki/Object-oriented_programming
[4]Class Hierarchy https://en.wikipedia.org/wiki/Class_hierarchy

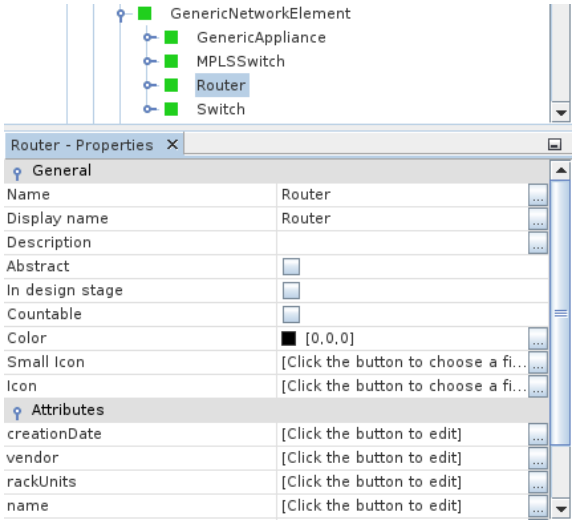


Figure 5: Properties of class **Router**

but it should be used to mark classes whose instances can have graphical representations, but they're not really part of the inventory, such as **Slots**. **In Design Stage** is just a way to mark a class as part of an ongoing data model intervention, and thus, classes with that attribute set to true can not be instantiated. **Color** is the color of the default square icon used to display the object in a tree or view. This icon will be used as long as the **Small Icon** attribute is null. **Small Icon** is the icon that will be used in trees and its size can't exceed 16x16 pixels. **Icon** is the icon used in views, and has a maximum size of 32x32 pixels.

Important

- All user-created classes are set In Design Stage = **true** by default. You won't be able to create objects of these classes until you set it to **false**.
- As a convention, all abstract classes have the prefix **Generic**. Note that a few core classes (like **InventoryObject** or **true**) are abstract are the exception to this rule. You, however, should try to follow this convention as much as possible.

- The second section contains the class fields (attributes). In the figure5, class Router has six attributes: name, state, conditions, vendor, serialNumber and creationDate. Click the button next to the attribute name to customize it (see figure 6).

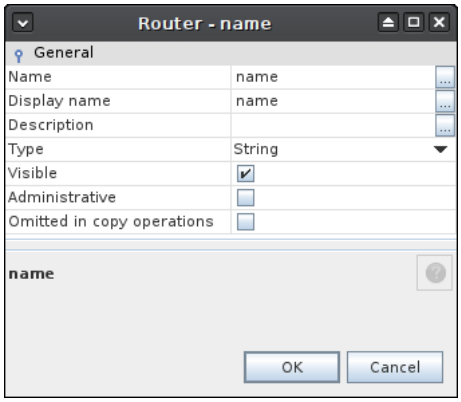


Figure 6: Properties of attribute **name** in class **Router**

In this window, you can modify the attribute's name, display name, description, type (the drop-down list will show you primitive types -String, Integer, Float, Long, etc- and all available non-abstract list types). When you change an attribute's type, all existing instances will be modified to reflect the change, which means that the values of the modified attribute will be converted to the new type if possible (say, from Integers to Strings). If the conversion is not possible, the new value will be set to null. You can also manage the attribute visibility. Attributes marked as "Administrative" will be shown in a separate tab in the object's property sheet. Sometimes, there are attributes that are used only for administrative purposes and might confuse the end user if mixed with the regular attributes. Finally, you can choose what attributes shouldn't be transferred from one object to another in a copy operation.

Important

- You may lose information when changing an attribute's type. make sure the conversion to the new type is possible before you do it.
- Although there's a Cancel button at the bottom of the window, it does not really work. When you perform a change, it's saved immediately.

You can also create and delete classes and attributes by right-clicking a class node (see figure 7) New subclasses inherit the parent class attributes. Classes with instances or subclasses can not

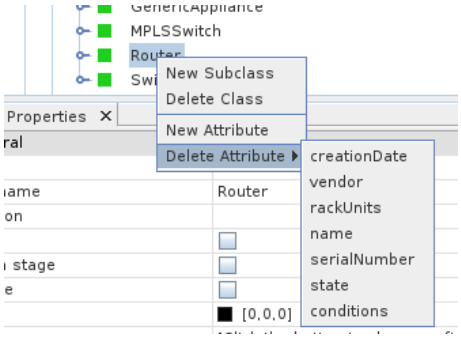


Figure 7: Class **Router** context menu

be deleted (this is a feature to avoid unintended loss of data). Also, attribute **name** can not be deleted.

Important It's highly recommended **NOT** to rename abstract core classes, as some of them are used internally to support many features and renaming them may turn the system unstable.

Containment Manager

Another key concept in Kuwaiba is containment. It consists of the ability to define what kind of objects can be created within others. For example, a **Country** can be inside a **Continent**, but can't be inside a **Rack**. A **Port** is usually within a **Board**, and not inside a **City**. These business rules can be defined using the Containment Manager.

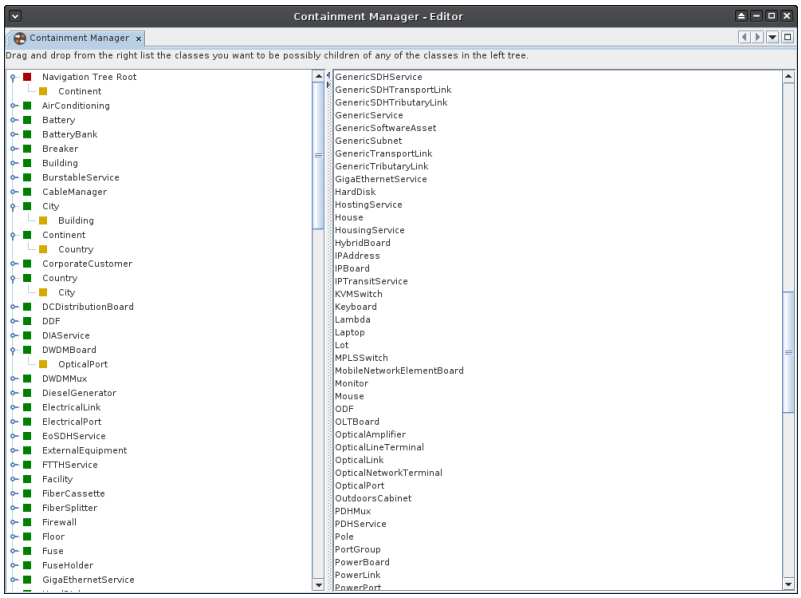


Figure 8: Containment Manager main window. Zoom in the image to see the details

The main window is divided in two panels (see figure 8, zoom in the image to see the details). The one on the left is a tree that holds all the classes plus the **Navigation Tree Root**. The children of the left-side tree node are the possible classes that can be contained. In the figure 8 there are five nodes expanded: **City**, that has one node inside: **Building**. That means that below a given city, you will only be able to add **Building** objects. Likewise, inside a **Continent** you can only create instances of **Country**, and inside those instances, only objects of class **City**. Under the root of the Navigation Tree, only instances of **Continent** are to be created. Finally, only **OpticalPorts** are supported under **DWDMBoards**. If for your operation Continents are not relevant, or if your routers do not have boards, but only ports, simplify the hierarchy as much as you want to meet your needs. To remove a possible children class, just right-click on it and select "Remove", and instances from that class will no longer be available to be added under the parent class, though the objects created already will remain linked to the respective parent objects.

Important

- To avoid adding one by one many classes to a parent, you can use the flexibility of the data model as a hierarchical structure. For example, a **Rack** may contain within many types of equipment (routers, DDFs, switches, battery banks, etc). Instead of adding one by one each of these classes, you can add a common super class and all of them will be added automatically. For this example a common super class for most of those classes could be **GenericCommunicationsElement**.
- To search for a particular class, just select any node in the desired side of the panel and type the first letters of the class name. If there are many occurrences of the term, jump from one to another using the F3 key.
- The changes are applied immediately, however, if you happen to not see them reflected, press the Refresh Cache button  in the main toolbar (see table 1).

Navigation Tree

This module presents in a tree fashion the physical objects of your inventory organized according to the containment hierarchy defined with the tool described in the previous chapter (see **Containment Manager**). Just like the Data Model Manager, the Properties window will display

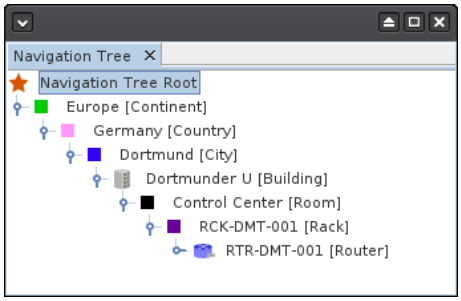


Figure 9: Navigation tree showing objects with default and user-defined icons

the attributes of the object selected in the Navigation Tree. These attributes match the visible attributes defined in the **Containment Manager**. Every change is automatically committed to

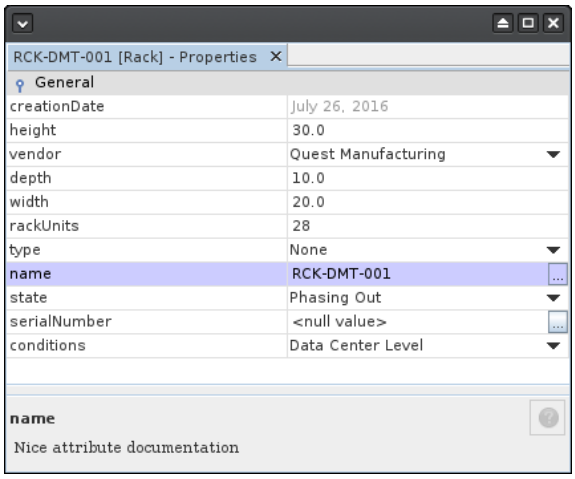


Figure 10: Properties of a selected **Rack** object

the database once you hit the Enter key. When editing dates, you need to select another attribute to commit the changes instead of pressing Enter. In the **Containment Manager** you can also configure what labels will be displayed instead of the actual names of the attributes and the help string in the lowest part of the window.

Every node has a set of actions, some will be active for all objects, some depend on the type of element that is selected. In the figure 11 you can see the actions enabled for a **Rack** object.

- **New Object:** The list of object types that can be contained for the selected element type according to the configured Containment Hierarchy. In this case, a **Rack** can only contain **Routers**.
- **Copy:** A plain copy operation.
- **Paste:** A plain paste operation. You can only paste objects where it is allowed according to the configured Containment Hierarchy.
- **Update:** Update the node information. Useful when a changed has made to the object from a external source (e.g. another user) or if you create a new list type affecting one of the

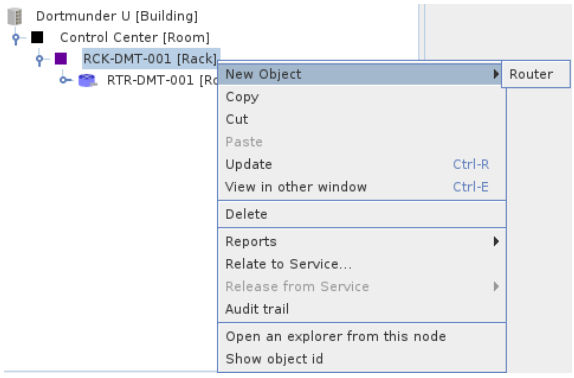


Figure 11: Properties of a selected **Rack** object

attributes of the selected element. In this case, if you, for example, create an instance of **EquipmentVendor** (this will add a new entry to the **vendor** attribute list)

- **Delete:** Deletes the object. Tis will fail if the object has an incoming relationship, for example, a **Port** connected to a cable.
- **Reports:** The reports associate with this class. In this case, **Rack** has a report called **Rack Usage**. If no reports are associated, the option will appear grayed out.
- **Relate to service:** All inventory objects can be associated to an existing service. See more details in the chapter Service Manager.
- **Release from Service:** Removes the association between an object(resource) and a service. If the object is not related to any service, this option will appear grayed out.
- **Audit Trail:** This will display all the audit trail entries for the selected object, that is, all the changes made to the it.
- **Open an Explorer from this Node:** Opens a navigation tree whose root node will be the selected object. Useful when you want to explore an object with a many containment levels below.
- **Show Object Id:** Shows the database id of the selected object. Useful for troubleshooting purposes. It will also show the object's complete containment structure.

Figure 12: Object id action on the selected **Rack** object

Important

- Remember that you can always open the Properties window by selecting the main menu option Windows → Properties.
- You can change the name of an object in-line by pressing F2 on a selected node.

Relationship and Special Children Explorer

Apart from the main navigation tree, there are also two explorers that are very useful to navigate through domain-specific models. Both explorers are located in the Tools → Navigation menu.

- **Relationship Explorer:** Allows to see the special relationships of the selected object. When an object makes part of a domain-specific model (SDH, Physical Connections, MPLS, Software Licensing, etc) there are special bounds to other objects called **relationships** they have names documented on model-basis, and they can be seen using this explorer. In the figure 13, it is depicted an **OpticalPort** with two relationships, one called **endpointA** used in the Physical Connections model and it indicates that this port is the endpoint to a physical connection, probably a fiber optic. It also has a relationship called **uses**, which makes part of the Service management model. It indicates that the service called PDH Service-01 uses that port as a resource.

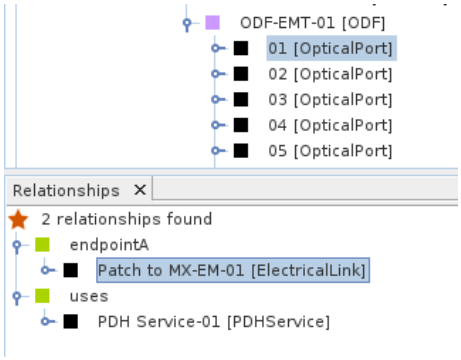


Figure 13: Special relationships of the selected **OpticalPort** object

- **Special Children Explorer:** The special children are children as in the containment hierarchy concept, but used in domain-specific models, which gives them particular behavior depending on the situation (that is, they can't be handled as simple objects in the navigation tree, because, for example, deleting them may require to perform other tasks but just removing the object from the database as they make part of a complex workflow). This is the case of the cables inside a conduit connecting two buildings. You can find more details about this scenario in the chapter **Physical Connections**.

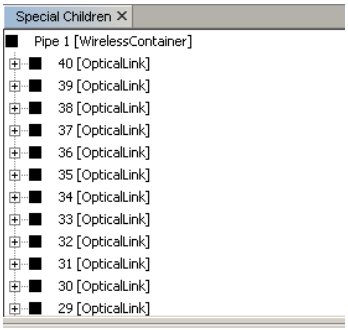


Figure 14: Fibers inside a container between two buildings

List Type Manager

Most of the attributes are primitive types (String, Integer, Booleans, etc), however, there are some more complex that point to another object in the database. This is the case of attributes such as *vendor*, which points to an object holding the information about the vendor of that equipment (support lines, account manager, etc) or *state*, that refers to the current operational state of the equipment (Working, Not Working, Stored, etc) and the state itself is an object, because it may hold information about what's the next allowed states, for example. Many objects in the database will have the same *vendor*, and many other will have the same *state*. In short, list types are those kind of attributes that point to an element in a limited set of objects. In terms of relational databases, you can see it as a many-to-one kind of relationships. To manage the existing list types and its instances, use the **Data Model Manager** and the **List Type Manager**.

To add new list types, add a subclass under GenericObjectList directly or any of its utility subclasses. List types are like any other class, you can customize them as needed. To create list type

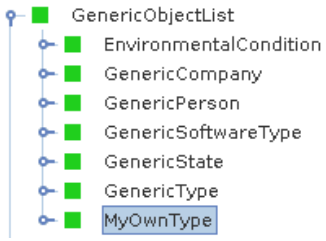


Figure 15: Custom list type

items, use the button  or the menu option Tools → List Type Manager. This module consists of a simple navigation tree similar to the one seen in the past chapter.

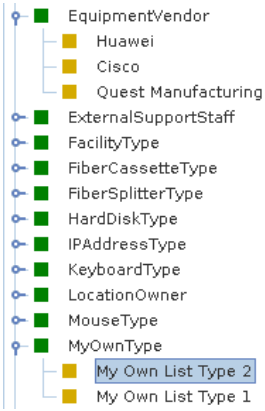


Figure 16: Custom list type items

By right-clicking and choosing *New* on a selected list type, you can create new items. The details for every item can be edited using the standard Properties Window as seen in the past chapter.

Important
If the changes are not immediately reflected when editing an object, use the Refresh Cache button and update the object (Right-click the object node and select the option *Update*).

In the figure 17 you can see the properties of an object of class **Router** modified to have an attribute called *myOwnAttribute* of type **MyOwnListType**.

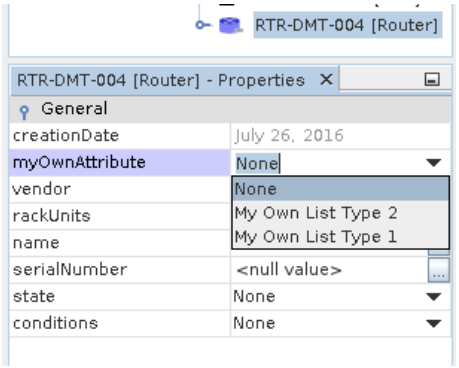


Figure 17: Object of class **Router** with a custom listy type attribute

Default and Rack Views

Physical View

A view is a graphical representation of an object. There are many types of views, because an object has different perspectives, for example, a service object may have a view showing all the resources associated to it, a second view showing how all those resources are connected and a third showing statistics about such service. All instances (objects) of subclasses of **ViewableObject** have a Physical View that displays the direct children of the selected node. Most objects, except logical and administrative assets and a few physical ones such as slots and ports are subclasses of **ViewableObject**. To access this view, open the Physical View window (see Toolbar items) and select a node in the navigation tree.

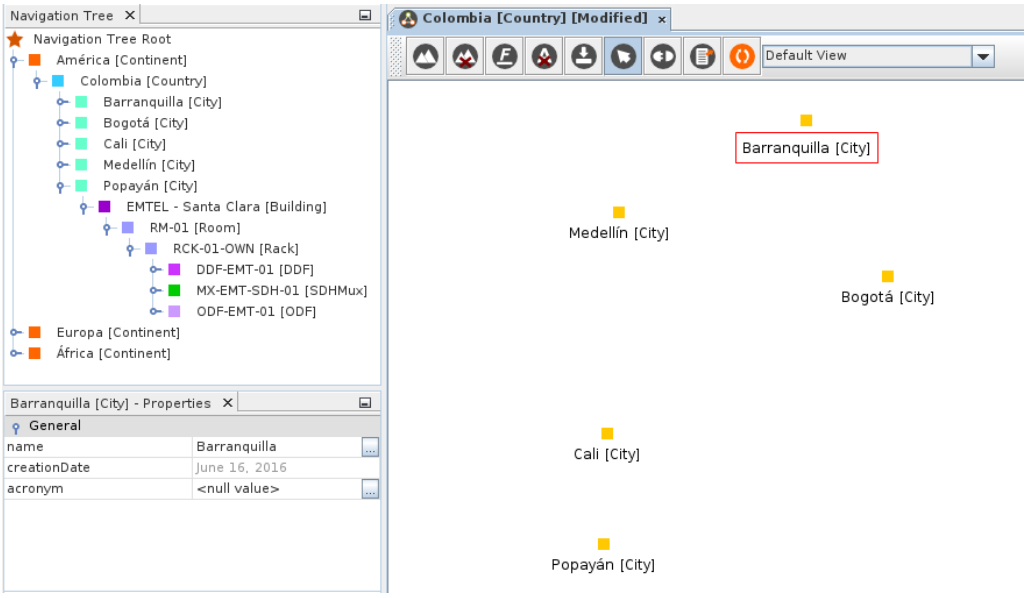


Figure 18: Physical View of the selected **Country** object

In the Physical View, you can move and connect the nodes, add a background and export the view. It is particularly useful at Room and City levels, because it will allow you to see how the children elements are located geographically (e.g. buildings) or in an enclosed space (racks in a data center as seen in figure 19).

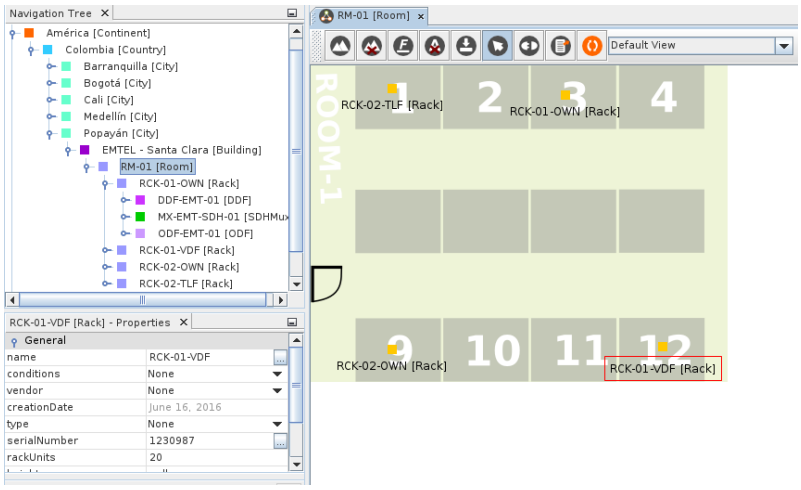


Figure 19: Physical View of the selected **Room** object

The figure 20 shows all general purpose tools available in this window.

Figure 20: Physical View toolbar

	Add background
	Remove background
	Format text
	Add background
	Hide/show the labels under the nodes
	Save view
	Add background
	Select/Move tool (selected by default)
	Connect tool (See **Physical Connections** for more details on how to use it)
	Export to image file
	Refresh view
	View type. Currently, only the Physical (default) and Rack view are available (see **Rack View** for more details on the latter)

Rack View

This view works only with objects of class **Rack** or its subclasses. It shows how the elements contained within the selected object are organized, based on their positions and number of rack units used.

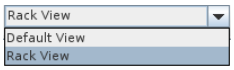


Figure 21: How to find the rack view in the toolbar

For this view to be correctly built, two conditions must be met:

- The rack must have its **rackUnits** attribute set to a valid integer value. This attribute stores the total number rack units supported by the rack. Typical values are 20, 28, 34, 40 or 45.

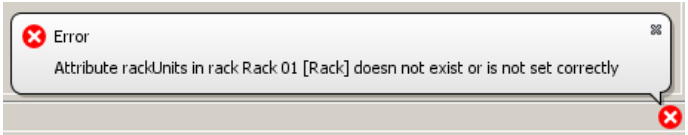


Figure 22: Error displayed when the rack does not have a valid value of rack units

- The attributes **rackUnits** and **position** must exist and set to valid values in the elements contained inside the rack. **rackUnits** in this case, makes reference to the number of rack units occupied by the contained element, while **position** is the start position of the contained element, 1-based, numbered from top to bottom. Your equipment vendor usually provides this value.

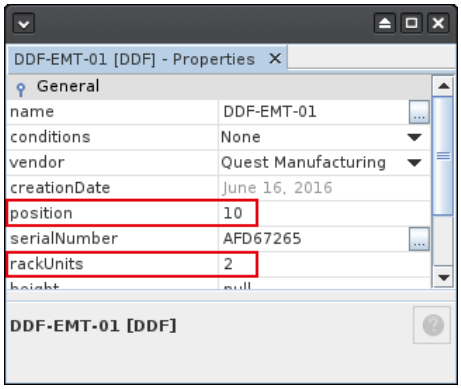


Figure 23: Attributes correctly set in a DDF contained inside a rack

Important

- The attribute names (**rackUnits**, **position**) are case sensitive and must be integers.
- Some elements you want to create inside a rack **might not** have those attributes by default, so you will need to create them using the Data Model Manager.

If all values are correct, the view should look like this:

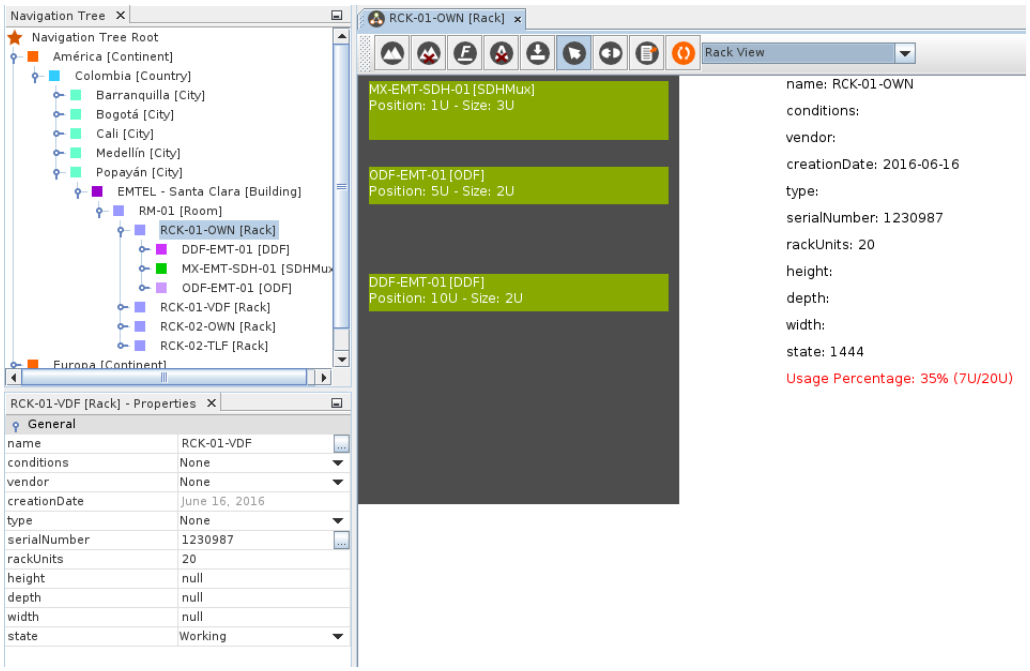


Figure 24: Sample rack view

Physical Connections

The Physical Connections toolkit is tightly integrated to the Physical View module. With Kuwaiba you can create physical layer connections using cables, fiber optics or radio links very easily, navigate through the connections and inspect the resources in use.

Before presenting the tools provided by the application, let's first clarify and introduce some concepts. This module deals solely with L1 topologies. It's only about cables, ports, etc. The data model provides four types of entities to represent physical layer elements:

- **Connections:** These are all point-to-point physical links. In the current data model, there are three types of connections: **ElectricalLink** (for electrical connections like coaxial, twisted pairs and the like), **OpticalLink** (for fiber optics) and **RadioLink** (for radio links -Microwave links, mostly-).
- **Containers:** All objects that can be used to contain, wrap and protect connections (understanding *Connection* as defined above). There are two types of containers: **WireContainer** (used to contain all kind of cables -wires and fibers-, like pipes, conduits, ditches, etc) and **WirelessContainer** (used to contain radio channels or carriers).
- **Nodes:** These objects are endpoints to *Containers*. In the default data model you will find classes like: **Tower**, **Warehouse**, **Facility**, **Shelter**, **Building**, **Floor** and **Room**, but in general, any subclass of **GenericPhysicalNode** can be a so called *Node*. **Endpoints:** These objects are endpoints to *Connections*. In practice, they're always some kind of **Port** (in the data model context, this means all subclasses of **GenericPort**).

In summary, you can only connect *Nodes* using *Containers* and *Endpoints* using *Connections*. To create new connections, select in the **Navigation Tree** an element and try to connected the desired children using the Physical View. For example, if you want to connect two buildings, select the city which is the parent for both. Likewise, if you want to connect a router to an ODF, you probably will have to select the room or the rack they are located. In short, select the nearest common parent between the two elements you want to connect.

> **Important**
> You can also create power connections with this approach.

	Select the Connection tool
	Create a wire container
	Create a wireless container
	Create an electrical link
	Create an optical link
	Create a wireless link

Table 2: Connection tools

Let's put together all this concepts with some examples.

Example 1

In this example, we will create a direct connection between two routers in the same room, but in different racks using a CAT-5 patch as shown in the figure 25.

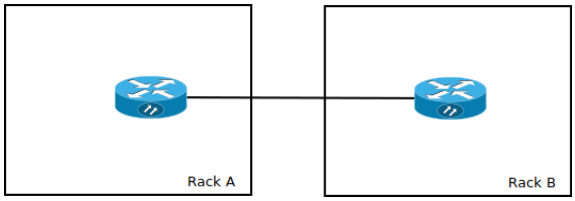


Figure 25: Connection diagram for example 1

Let's consider the layout in figure 26

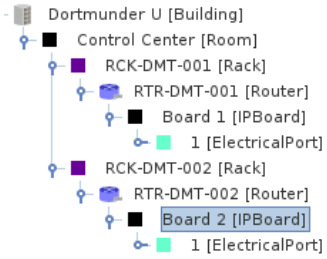


Figure 26: Containment layout for example 1

1. Select the nearest common parent in the navigation tree (in this case the **Room** called Control Center), and using the select tool, change the default position of the nodes

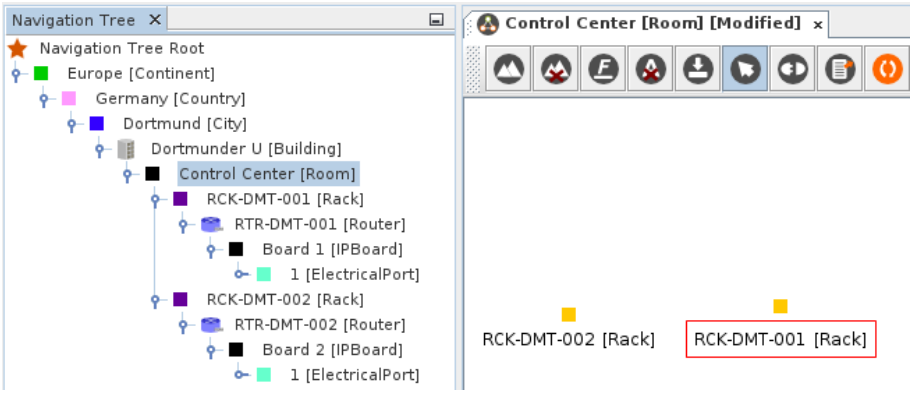


Figure 27: Racks in the room

2. Using the **List Type Manager**, create a list type called *CAT-5* under **ElectricalLinkType**. Following a similar procedure, you can create the different types of electrical connections depending on your network (POTS, coaxial, etc). We will use this later.

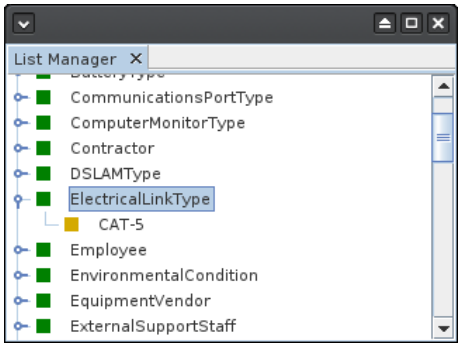


Figure 28: Creating the link type

3. Activating the connection tool, click on one node and hold, dragging the mouse until you reach the second node. Make sure you also select the type of connection you want to create, in this case, an **ElectricalLink** (a patch cable).

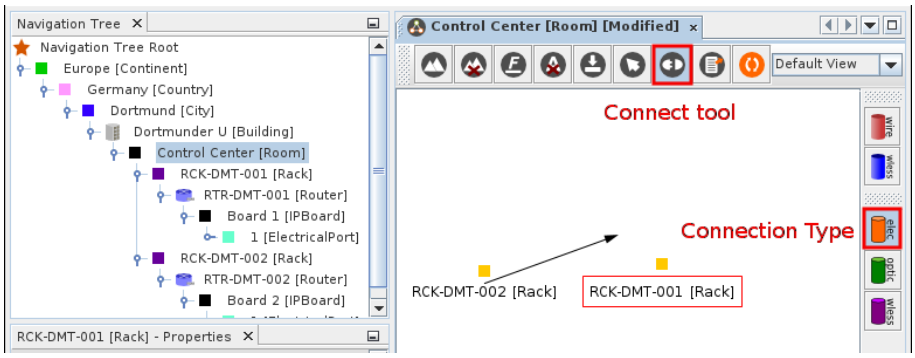


Figure 29: Creating the connection

4. This will open a wizard where you should select the endpoints of the connection (the end ports)

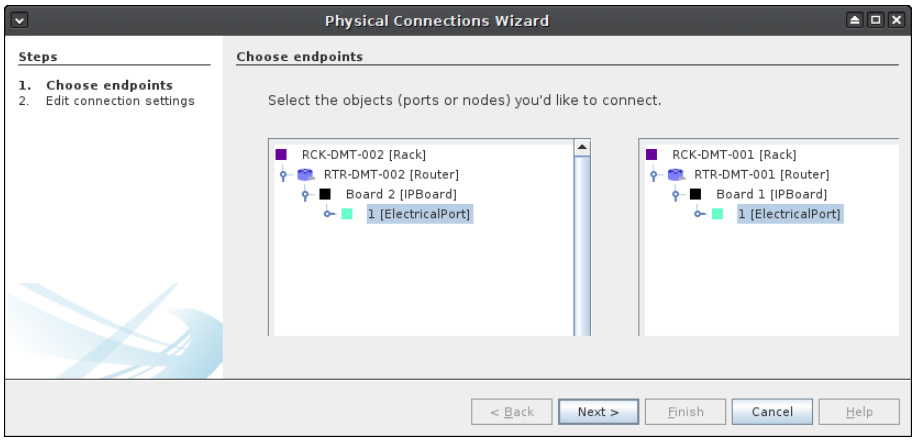


Figure 30: Connection wizard, step 1

5. In the next step of the wizard, you have to fill in the basic information about the connection: its name and type. Use the type we just created in step 2.

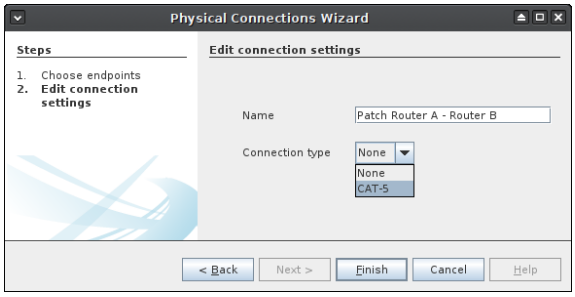


Figure 31: Connection wizard, step 2

And that's it. Double-clicking the connection will add a control point to it. You can add as many as you want, and control its route. The Properties window will get updated accordingly if you select the link or a node.

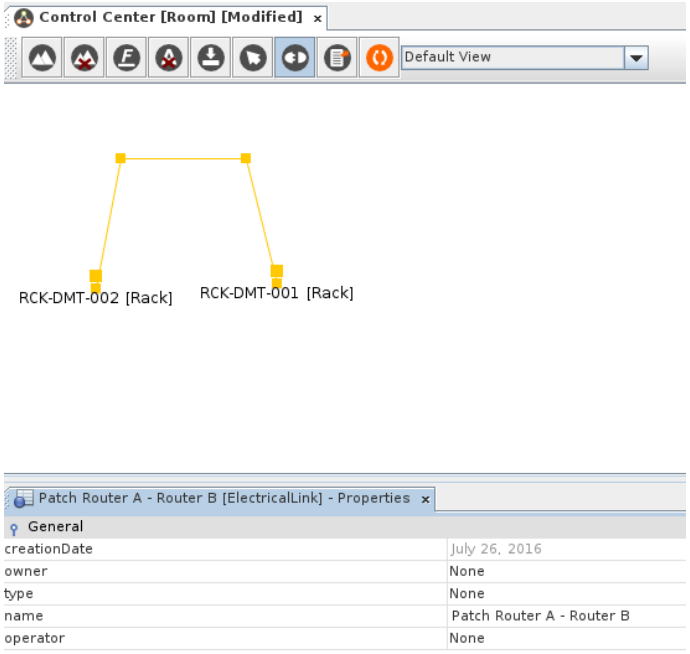


Figure 32: Final result

Important

Don't forget to save the view using the icon  in the toolbar, so your positioning changes are stored.

Example 2

The second example is more complex. We will connect two buildings with a conduit that contains fibers. In each building there's a router and an ODF, one of the fiber pairs connecting the buildings, will be linking the two ODFs, and then, from the ODF, we'll patch our way the routers.

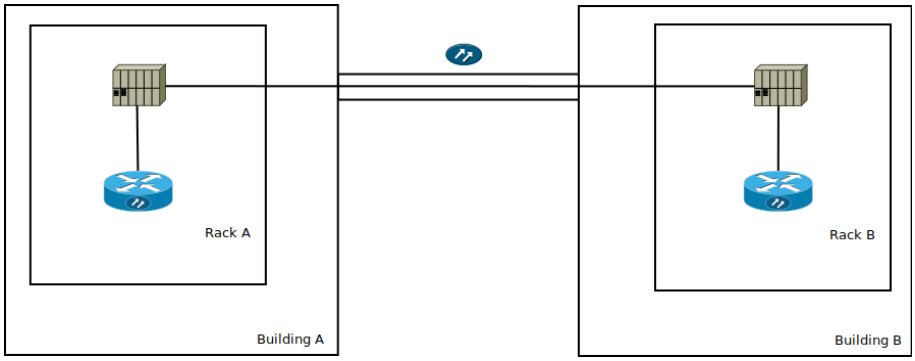


Figure 33: Connection diagram for example 2

We will use a similar containment layout for this example, just adding a couple ODFs.

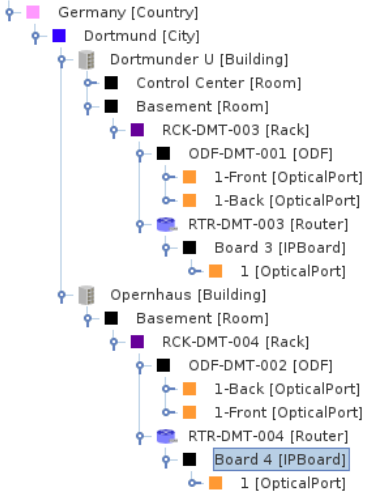


Figure 34: Containment layout for example 2

1. First, we must make the initial arrangements. In this case, we will use a Google maps image as background for the city (the nearest common parent) and locate the buildings. We also select the connection tool and the type of connection, which will be a **WireContainer** this time.

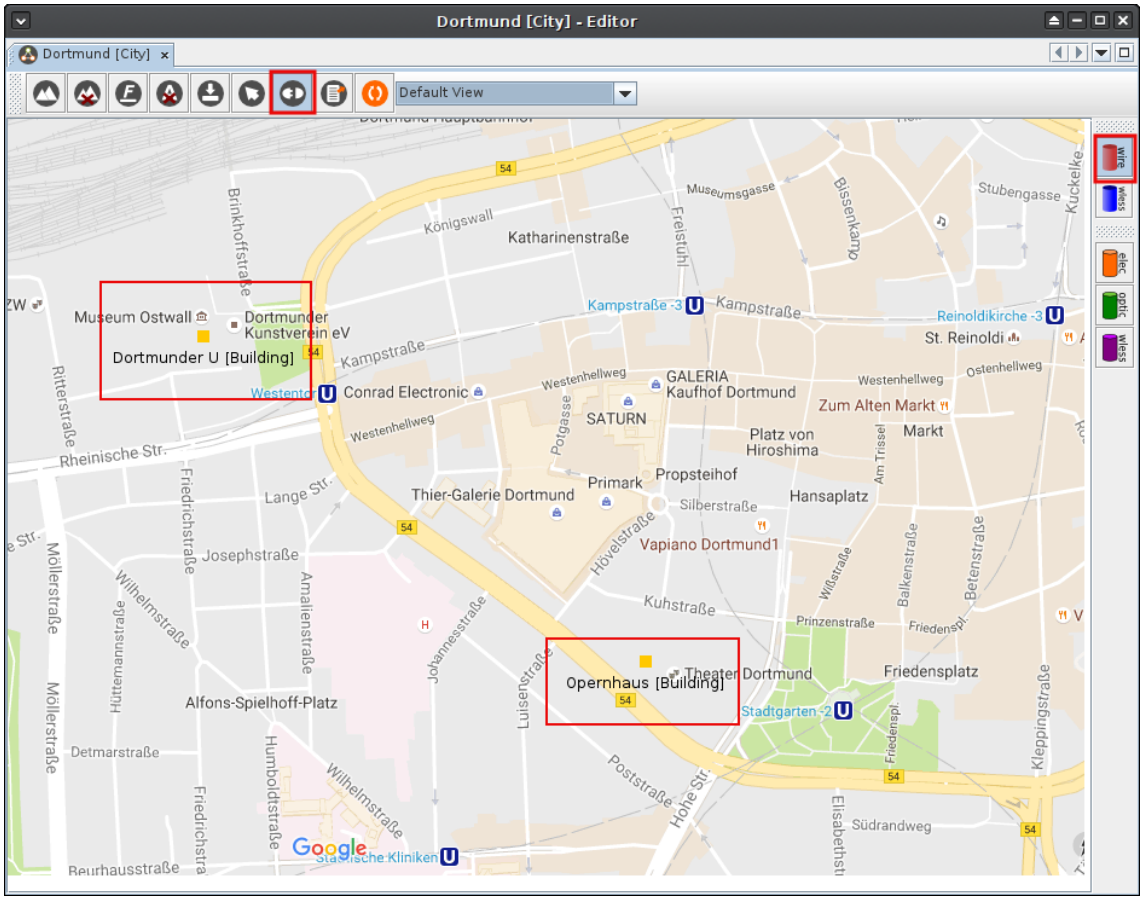


Figure 35: Buildings in the city

2. Since this time we are creating a **WireContainer**, the list type for it must be created under **WireContainerType**. Using the List Type Manager, add this entry:

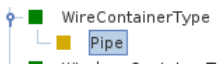


Figure 36: reating a connection type

3. Just like we did in the past example, let's start the connection wizard, but this time the endpoints will be the buildings instead of the ports (remember that we are creating a container here).

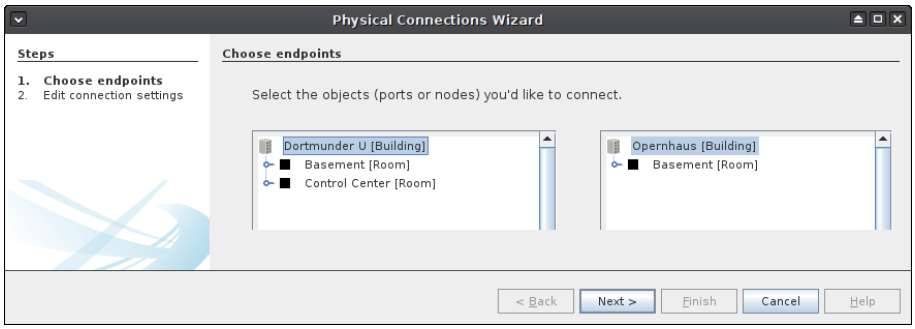


Figure 37: Connection wizard, step 1

4. In the second step, we fill in the basic fields, but in addition to what we had seen already, it's also necessary to provide information about what's gonna be contained inside the **WireContainer** and how many of those elements are to be created. Since this example is about connecting fibers, we select **OpticalLink** in the field *Children type* and 20 in *Num of children* just for the sake of the example.

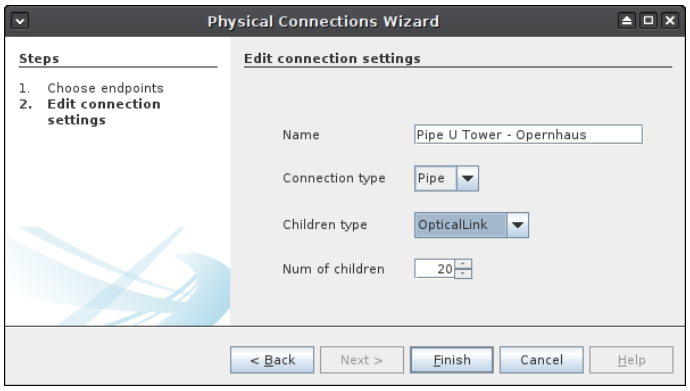


Figure 38: Connection wizard, step 2

5. Using the Select tool, modify the route of the connection. Now the container has been created successfully.

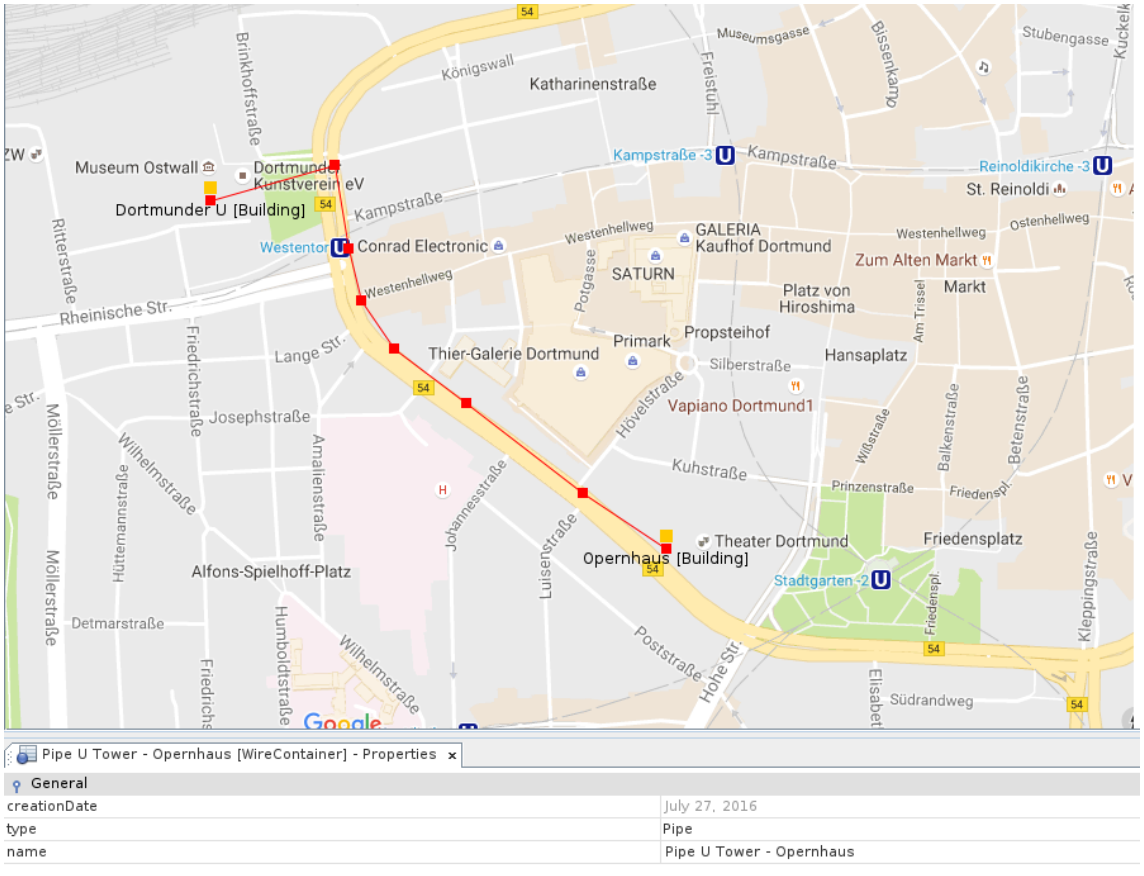


Figure 39: Final result

6. Once the container has been created, we can now connect a pair of fibers to each side's ODF back port (note the naming used for the ODF ports in the figure 34). To start a wizard to perform this operation, right-click the container and select the option *Connect links...*

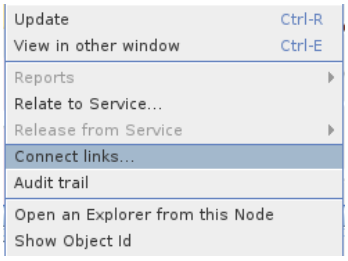


Figure 40: Container's contextual menu

7. This open a window divided in three panels. The side panels let you choose what ports do you want to connect to the fibers in the container. In this example, we will connect the first pair to the ports labeled as *Back* in the ODFs. You can select multiple ports and fibers and connect them at the same time (use the SHIFT key for multiple selection). Note that it's not necessary to connect both sides. Push the *Connect* button on the lower part of the central panel to finish the procedure.

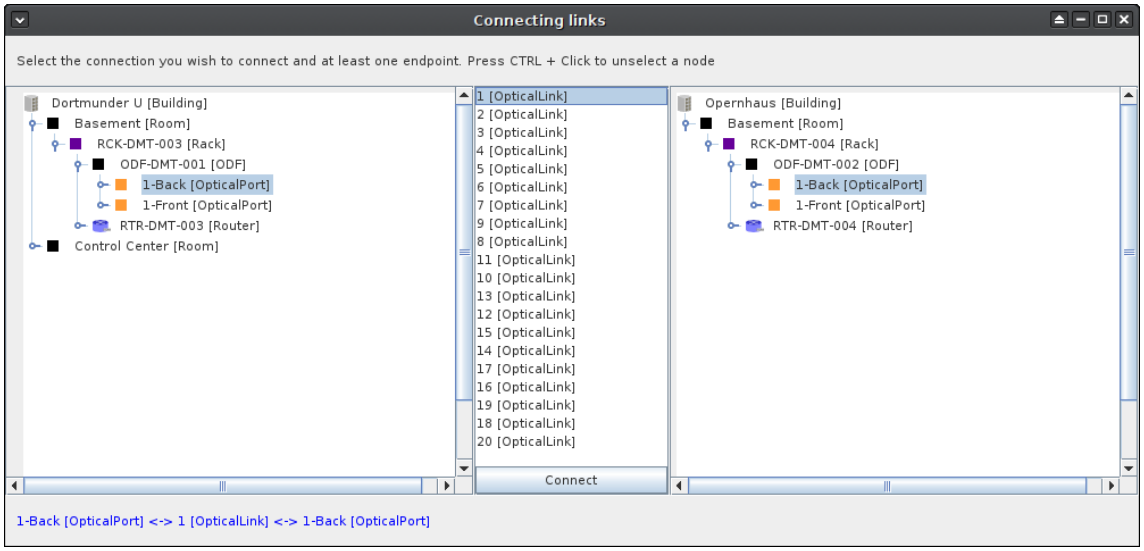


Figure 41: Connecting links

> **Important**
>
> - In optical connections, Rx and Tx ports are treated as a single port and a pair of fibers, are actually represented as a single **OpticalLink**.

It's possible to explore the contents of a container and see to what fiber the ports are connected to using the explorers in the menu Tools → Navigation. In the figure 42 you will see the Special Children explorer when the container is selected in the view. In the figure 43, the relationship explorer shows the connections of one of the ports in the ODF when selected in the Navigation Tree.

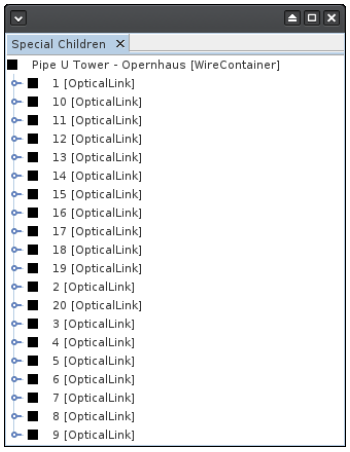


Figure 42: Exploring the container's contents

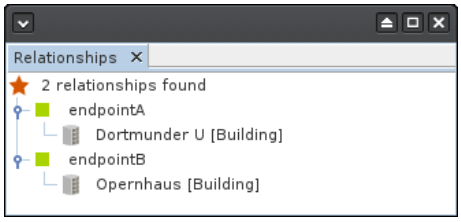


Figure 43: Exploring the container's relationships

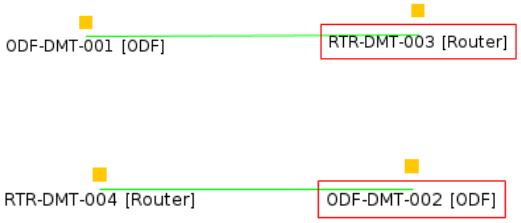


Figure 44: Optical patches between routers and ODFs

8. Now we will create an optical patch on each building between the routers and the 1-Front ports of their respective ODFs in a similar way we did in the **Example 1**.

> **Important**
>
> Remember to use an **OpticalLink** instead of an **ElectricalLink**.

9. There's still one last step: The ports 1-Front and 1-Back in each ODF are not yet bridged. To do this, we just have to right-click any of the two ports on each ODF and select the option *Connect Mirror Port...*. A mirror port is basically a direct connection between two ports. A cable is a separate object in Kuwaiba, a mirror connection is not even an object, is just a relationship between a port an another with the same parent (that is, a sibling). You can see it as a backplane connection.

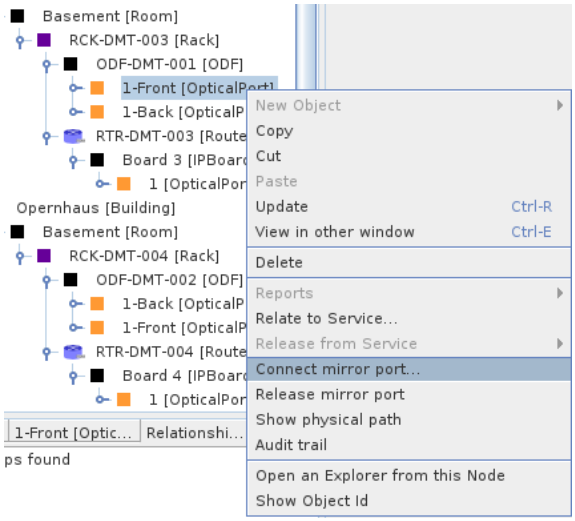


Figure 45: Mirror connection menu

This will open a window where you can select what port do you want to be mirror of the selected one. Since 1-Front ports only have one sibling (1-Back), that's the only option available.

Figure 46: Mirror connection details

Once the port is mirrored, you can see a new relationship in the relationship explorer when selecting any of the ODF ports in the **Navigation Tree**.

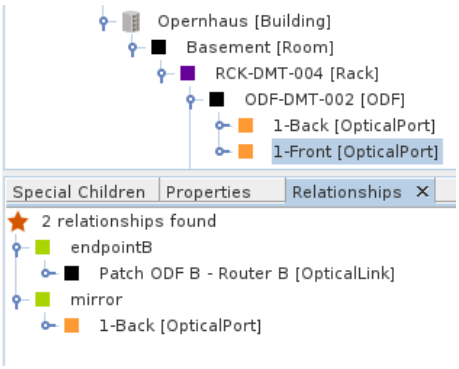


Figure 47: Mirror connection relationship

To release a mirror port, select the option from the context port's context menu.

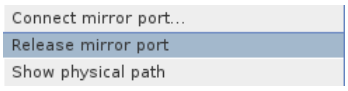


Figure 48: Release mirror connection menu option

A nice feature in Kuwaiba is the ability to see the path of a physical connection as long as there's continuity. In our example, there is continuity between the port in the router A (RTR-DMT-003) and the port in the Router B (RTR-DMT-004), that is, there are no active elements in between. To see the trace, right-click on any of the ports involved in the connection and select the option *Show physical path*. The figure 50 show the physical path between Router A's port and Router B's

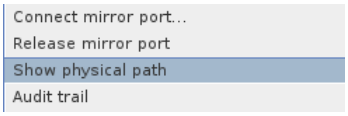


Figure 49: Physical path menu option

port. The port in red is where the trace begins.

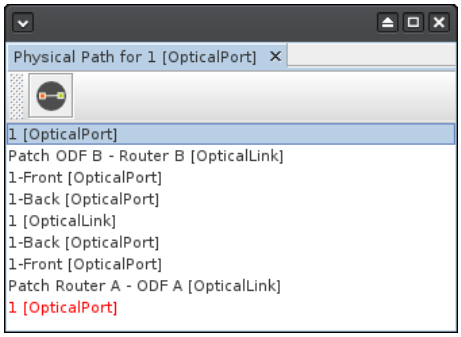


Figure 50: Physical path details

And pressing the button  you will get a graphical representation, easier to understand. The

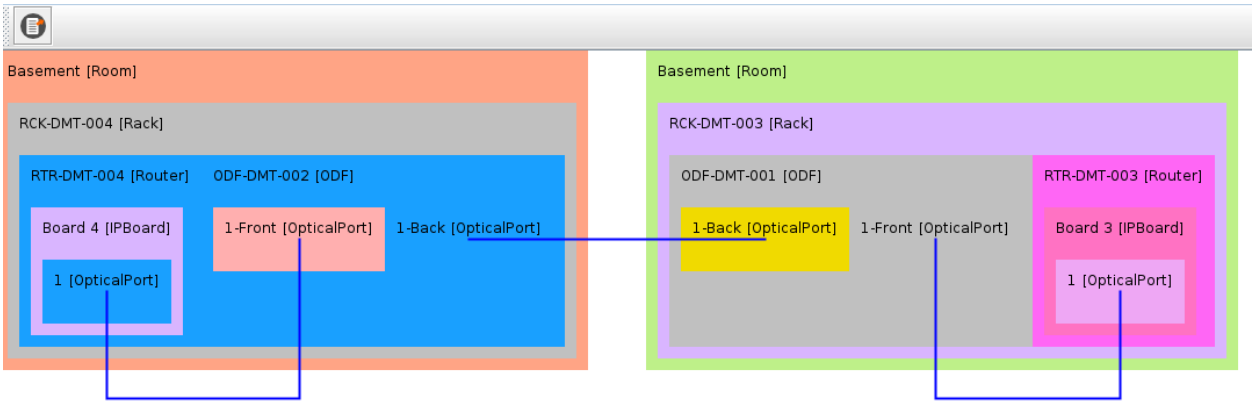
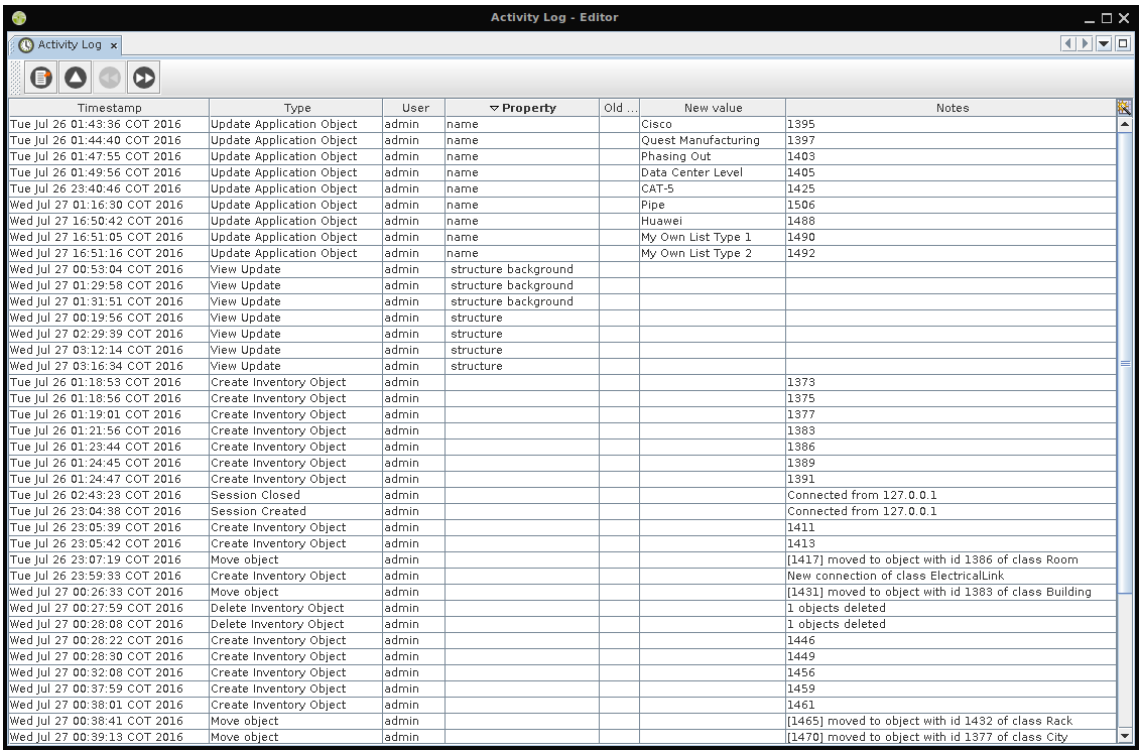


Figure 51: Physical path graphical representation

Properties window will be updated as you select any of the blocks. This view can also be updated.

Audit Trail

Kuwaiba is capable of tracking the changes performed by the users in the database for audit purposes. These changes can be made to inventory objects (equipment, locations, etc) or application objects (pools, tasks, user properties). There are two types of events that are logged: **General events**, that is, those that are not related to any object in particular, like new logins or creation of application objects. **Object-related events**, like property changes or move operations.



Timestamp	Type	User	Property	Old ...	New value	Notes
Tue Jul 26 01:43:36 COT 2016	Update Application Object	admin	name		Cisco	1395
Tue Jul 26 01:44:40 COT 2016	Update Application Object	admin	name		Quest Manufacturing	1397
Tue Jul 26 01:47:55 COT 2016	Update Application Object	admin	name		Phasing Out	1403
Tue Jul 26 01:49:56 COT 2016	Update Application Object	admin	name		Data Center Level	1405
Tue Jul 26 23:40:46 COT 2016	Update Application Object	admin	name		CAT-5	1425
Wed Jul 27 01:16:30 COT 2016	Update Application Object	admin	name		Pipe	1506
Wed Jul 27 16:50:42 COT 2016	Update Application Object	admin	name		Huawei	1488
Wed Jul 27 16:51:05 COT 2016	Update Application Object	admin	name		My Own List Type 1	1490
Wed Jul 27 16:51:16 COT 2016	Update Application Object	admin	name		My Own List Type 2	1492
Wed Jul 27 00:53:04 COT 2016	View Update	admin	structure background			
Wed Jul 27 01:29:58 COT 2016	View Update	admin	structure background			
Wed Jul 27 01:31:51 COT 2016	View Update	admin	structure background			
Wed Jul 27 00:19:56 COT 2016	View Update	admin	structure			
Wed Jul 27 02:29:39 COT 2016	View Update	admin	structure			
Wed Jul 27 03:12:14 COT 2016	View Update	admin	structure			
Wed Jul 27 03:16:34 COT 2016	View Update	admin	structure			
Tue Jul 26 01:18:53 COT 2016	Create Inventory Object	admin				1373
Tue Jul 26 01:18:56 COT 2016	Create Inventory Object	admin				1375
Tue Jul 26 01:19:01 COT 2016	Create Inventory Object	admin				1377
Tue Jul 26 01:21:56 COT 2016	Create Inventory Object	admin				1383
Tue Jul 26 01:23:44 COT 2016	Create Inventory Object	admin				1386
Tue Jul 26 01:24:45 COT 2016	Create Inventory Object	admin				1389
Tue Jul 26 01:24:47 COT 2016	Create Inventory Object	admin				1391
Tue Jul 26 02:43:23 COT 2016	Session Closed	admin				Connected from 127.0.0.1
Tue Jul 26 23:04:38 COT 2016	Session Created	admin				Connected from 127.0.0.1
Tue Jul 26 23:05:39 COT 2016	Create Inventory Object	admin				1411
Tue Jul 26 23:05:42 COT 2016	Create Inventory Object	admin				1413
Tue Jul 26 23:07:19 COT 2016	Move object	admin				[1417] moved to object with id 1386 of class Room
Tue Jul 26 23:59:33 COT 2016	Create Inventory Object	admin				New connection of class ElectricalLink
Wed Jul 27 00:26:33 COT 2016	Move object	admin				[1431] moved to object with id 1383 of class Building
Wed Jul 27 00:27:59 COT 2016	Delete Inventory Object	admin				1 objects deleted
Wed Jul 27 00:28:08 COT 2016	Delete Inventory Object	admin				1 objects deleted
Wed Jul 27 00:28:22 COT 2016	Create Inventory Object	admin				1446
Wed Jul 27 00:28:30 COT 2016	Create Inventory Object	admin				1449
Wed Jul 27 00:32:08 COT 2016	Create Inventory Object	admin				1456
Wed Jul 27 00:37:59 COT 2016	Create Inventory Object	admin				1459
Wed Jul 27 00:38:01 COT 2016	Create Inventory Object	admin				1461
Wed Jul 27 00:38:41 COT 2016	Move object	admin				[1465] moved to object with id 1432 of class Rack
Wed Jul 27 00:39:13 COT 2016	Move object	admin				[1470] moved to object with id 1377 of class City

Figure 52: General events window

The figure 52 shows the window displayed after pressing the **Audit Trail** button  (or selecting the menu option Tools → Audit Trail). It shows the general events, paginated and chronologically sorted from the oldest to the newest event.

	Export to CSV file
	Retrieve all log entries (may take some time depending on the number of records)
	Previous page
	Next page

Table 3: Audit trail toolbar icons

The object-related audit trail window is very similar, but contains only the events related to the selected node. To see the audit trail, select a node in a tree or view, right-click on it and select the *Audit Trail* option.

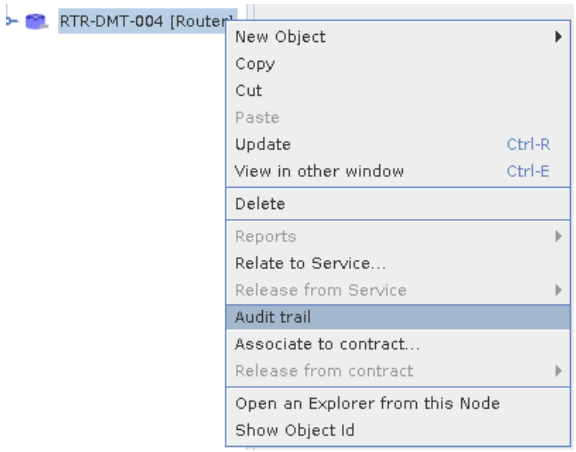
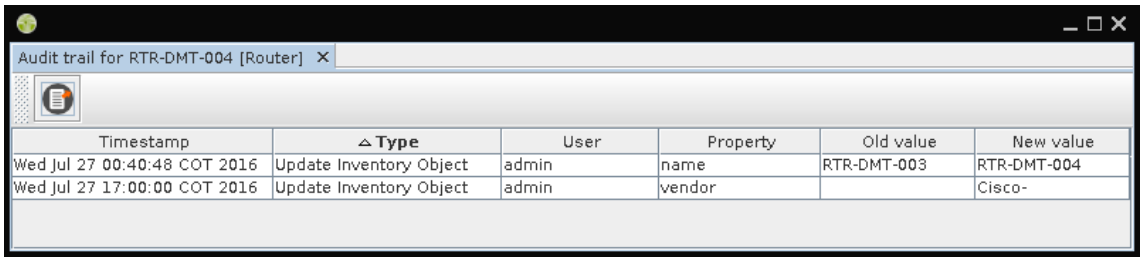


Figure 53: Audit trail menu option

The result is not paginated, but can be exported to a CSV file.



Timestamp	Type	User	Property	Old value	New value
Wed Jul 27 00:40:48 COT 2016	Update Inventory Object	admin	name	RTR-DMT-003	RTR-DMT-004
Wed Jul 27 17:00:00 COT 2016	Update Inventory Object	admin	vendor		Cisco-

Figure 54: **Router** object audit trail

Topology Designer

Use this module to make sketches of your network, but using the equipment already created in the inventory. Most people use applications like MS Power Point™, MS Visio™or a CAD program for this, but here it is already integrated. Use the button in the main toolbar to open the designer.

	New topology
	Open topology
	Save topology
	Delete topology/Clear canvas
	Export topology to image file
	Select tool
	Connect tool
	Insert cloud
	Insert frame
	Add background
	Remove background

Table 4: Topology Designer toolbar icons

> **Important**
> The connections made using this module are not actual inventory objects (unlike those seen in the chapter **Physical Connections**), simply lines linking nodes.

For starters, drag objects from the **Navigation Tree** onto the canvas of the view. You can add equipment (routers, switches, multiplexers) or locations (buildings, outdoor cabinets, poles, etc). If you have previously set the *icon* of the classes the object you have dragged are instance of, it will be used, otherwise, a colored squared will be used instead to represent the nodes. In the figure 55 there are two routers and an object of class **ExternalEquipment**.

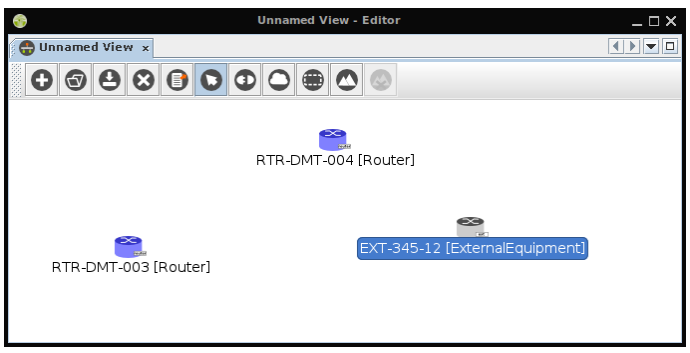


Figure 55: Sample topology without connections

You can add nodes that are not inventory objects, but that can help to decorate the topology. If you click the cloud button  , a cloud will be added to the topology. If the Selection tool is activated  , drag the nodes to position them as needed. You can also add a frame by clicking the frame button  . The name of the nodes can be changed by double-clicking them. The label of a frame can also be changed by double-clicking its border.

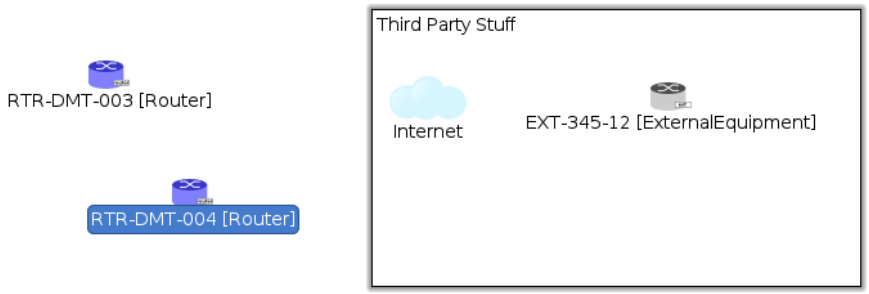


Figure 56: Same topology, but adding a cloud and a frame

To resize a frame, click on a corner and drag until you get the desired size. To create connections, select the Connection tool and link two nodes as you did in the **Physical Connections** chapter.

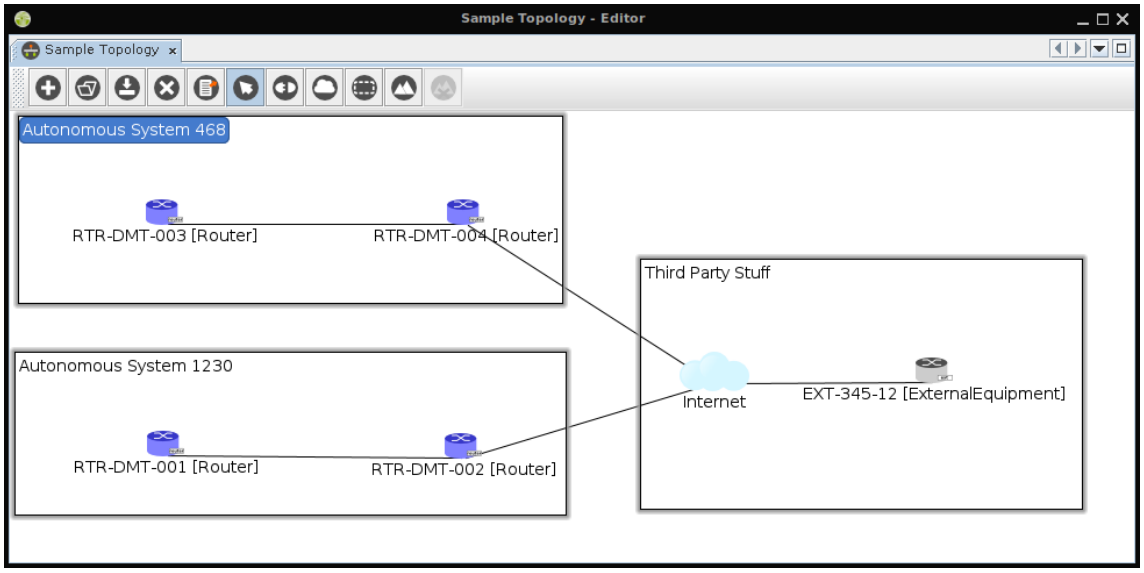


Figure 57: Topology with connections

Don't forget to save the topology with pressing the Save button  . If saved, this topology can be loaded later using the Open button 

Service Manager

Contract Manager

This small module allows you to organize your contracts in pools and keep track of the expiration dates and service providers. To open it, use the menu option Tools → Advanced → Contract Manager.

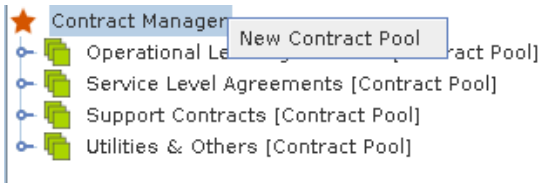


Figure 58: Creating a Contract Pool

The contracts are stored in pools, so first, you need to create your contract structure. Right click on the root node and select *New Contract Pool*. Then, inside the pools create the contracts you want, and set their properties.

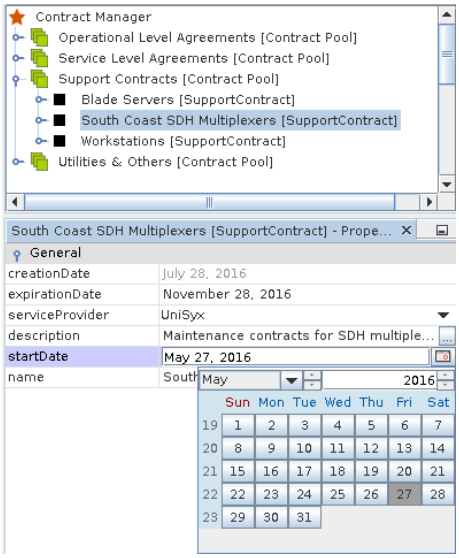


Figure 59: Editing a contract

Important
If you are editing a date, so it can be set correctly, instead of just hitting the Enter key, change the focus to other attribute in the property sheet (that is, select another property with the mouse).

To create a service provider for the contracts, go to the **List Type Manager** and create an entry under **ServiceProvider**.

Figure 60: New service provider

Any inventory object can be subject to a contract. You can associate an element to an existing contract, right click on it ans select the option *Associate to contract...* and select the desired one as shown in the figure 62.

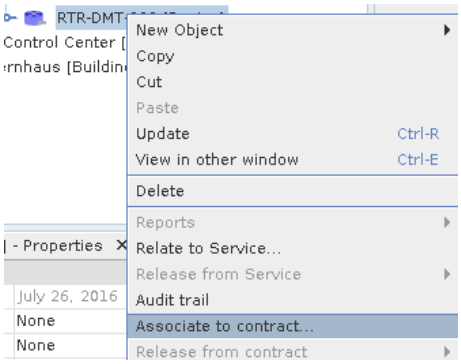


Figure 61: Associate to contract option

Figure 62: Contract list

The new elements associated to a contract are also available in the Contract Manager window.

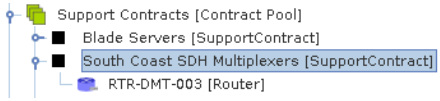


Figure 63: Elements associated to a contract

This association can be undone either from the **Navigation Tree** or the **Contract Manager** using the node's context menu.

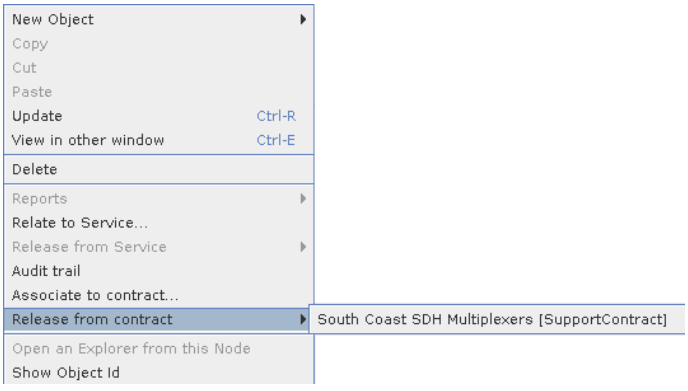


Figure 64: Removing the association

Task Manager

The Task Manager is the early release of a complete automation and notification module. At this stage, you can create tasks, edit a script that will be executed and execute this task either on demand or associated to a Dashboard widget that will be loaded at user login time. To open the Task Manager, select the menu option Tools → Task Manager or press the button in the main toolbar. Right-clicking the root node of the task tree will present a menu to create tasks. A task

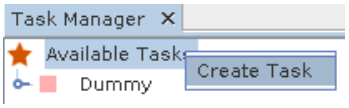


Figure 65: Creating a new task

object has the following attributes

name	The name of the task.
description	The description of the task.
enabled	Is this task enabled?
script	The Groovy script to be executed by this task.
startTime	The exact time and date this task shoud start its execution. Applicable only to tasks with *executionType* set to Loop. **Not used yet**.
everyXMinutes	After the start time, every how many minutes should this task be executed? Applicable only to tasks with *executionType* set to Loop. **Not used yet**.
executionType	How should the task be executed. • On Demand: Run only if the user executes it manually. • User Login: Run upon subscribed users login. • Loop: Run on a periodic basis, using the parameters described above. • System Startup: Run when the system goes up. Currently, only *On Demand* and *User Login* are used
email	The email of the person or group that will receive the notification. Optional. **Not used yet**.
notificationType	What type of notification should the subscribed users receive? • No Notification: No notification. • Client Managed: The client that executes the task decides how to notify. • Email: Send an email to the address described above. **Not used yet**.

It is possible to subscribe users, so that they receive notifications with the result of the execution. Right-click on the task node, select the option *Subscribe users...* and pick a user from the list.
It is out of the scope of this document to teach how to code scripts, however, in the directory **scripts** of the client installer, you will find some samples to guide you through. It's important to note, though, that most of the times, the scripts will require input parameters. This parameters

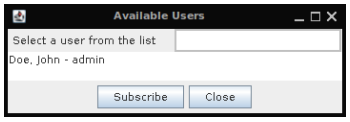


Figure 66: Subscribing a user

can be easily added to the task so the user can fill them in before to run the task. To create parameters, right-click the task node and select the option *Add Parameter...*

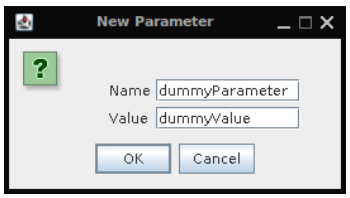


Figure 67: Adding a parameter

Once added, it will be immediately available in the property sheet, under the category *Task Parameters*.

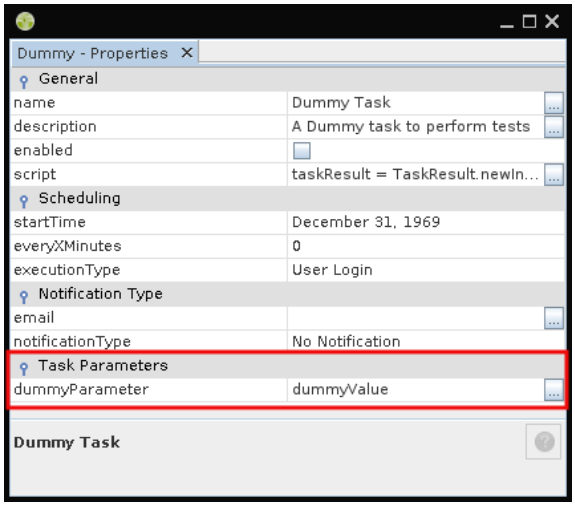


Figure 68: Subscribing a user

Tasks can also be executed on-demand. To do that, select the *Execute Task* option from the task node's context menu.

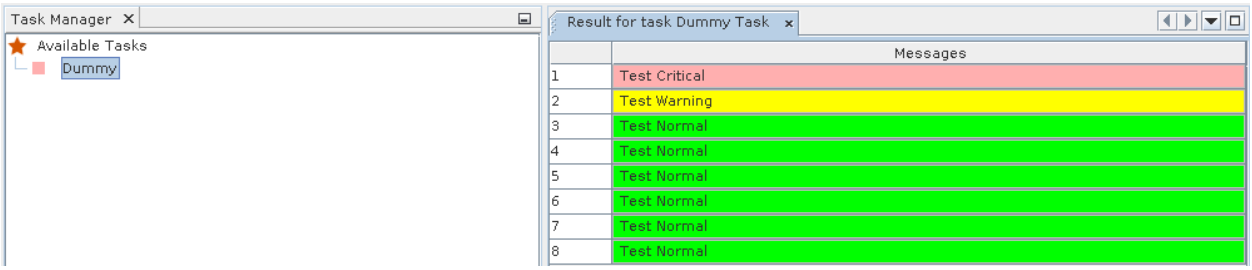


Figure 69: Task execution

Important
The full set of methods available in a script can be found at
http://kuwaiba.org/javadoc/persistence/org/kuwaiba/apis/persistence/interfaces/ApplicationEntityManager.html

SDH Networks

This module allows to create end-to-end virtual circuits and all other logical entities required by the SDH protocol[5]. Let's introduce some concepts first. See figure 70 for sample connection that illustrate the following terms.

1. **TransportLink:** A point-to-point logical connection that represents a single STMX. By default, the **TransportLinks** included in Kuwaiba are SMT1, STM4, STM16, STM64 and STM256.

2. **ContainerLink:** It's a logical connection that may or may not have multiple hops and connects ports within the boundaries of the transport network. A **ContainerLink** ends where at least one **TransporLink** ends, but in its path, the **ContainerLink** may have used many **TransportLinks**. In simple terms, a ContainerLink is an SDH virtual circuit. By default there are defined 5 types of ContainerLinks: VC12, VC3, VC4, VC4x4 and VC4x16. The latter two provide support for concatenation.

3. **TributaryLink:** Objects of this types are not circuits themselves, they rather use actual virtual circuits (ContainerLinks) and end in the ports where the SDH service is delivered to the customer (usually tributary ports).

In short, Transport and ContainerLinks provide the foundations for the hierarchy, while the TributaryLinks provide information about where the circuits are delivered (that is, what are the endpoints of the connections). Detailed information about the data model portion devoted to SDH, read the document *SDH Model and Tools*.

[5]SDH Graphical Overview http://www.cisco.com/c/en/us/support/docs/optical/synchronous-digital-hierarchy-sdh/28327-sdh-28327.html

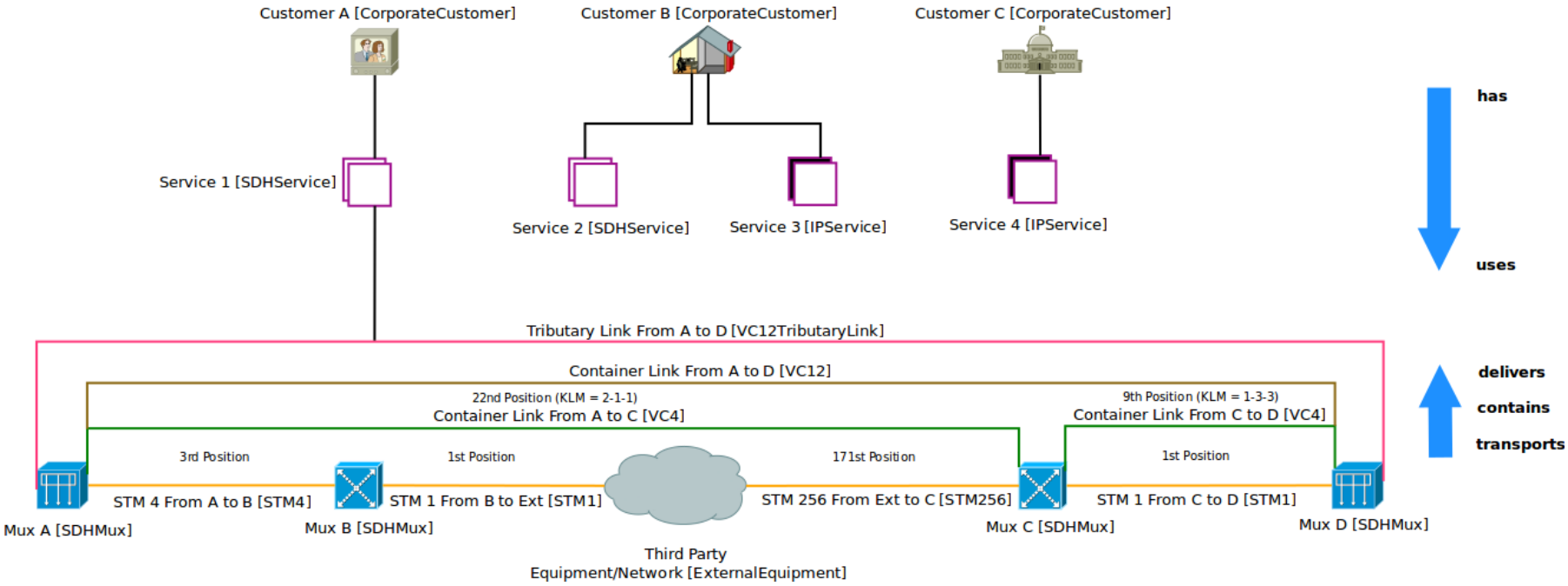


Figure 70: Sample multi-hop SDH circuit and its relationships with services, customers and physical resources

To create an SDH topology, go to the menu option Tols → Advanced → SDH Networks. This will open an empty canvas similar to the Topology Designer.

	New topology
	Open topology
	Save topology
	Delete topology/Clear canvas
	Export topology to image file
	Select tool
	Connect tool
	Create a **TransportLink**
	Create a **ContainerLink**
	Create **TributaryLink**

Table 5: SDH module toolbar icons

Drag and drop the equipment you need from the Navigation Tree and place them on the canvas.

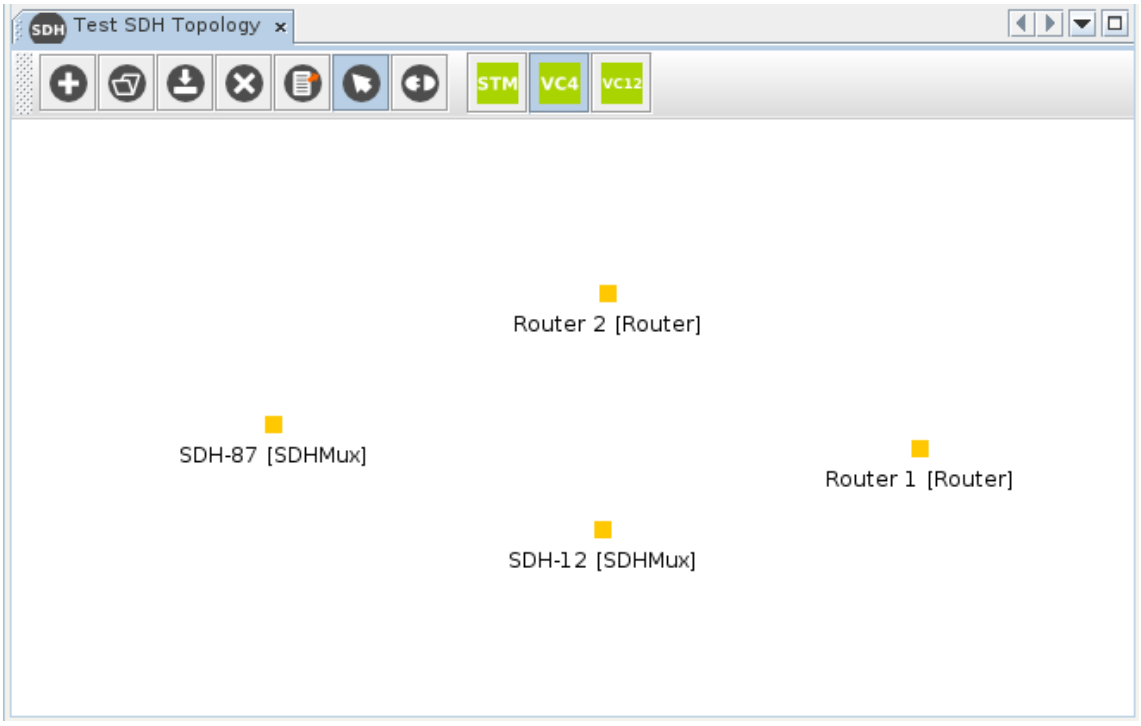


Figure 71: Some routers and multiplexers in an SDH topology

Let's consider this 4 equipment topology. What we are going to do, is to create a ring using STM-4 on one branch and SMT-1 on the other. We will create a VC12 between **SDH-87** and **Router 1**. Each equipment has the same board/port layout: Two boards, a tributary and an aggregate one, with two ports each.

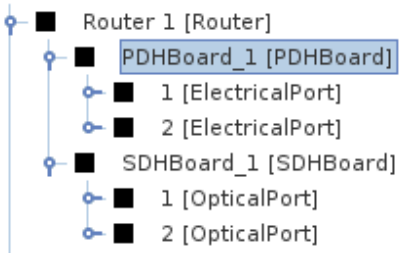


Figure 72: Port/board layout

First, we have to create the most basic support, that is the TransportLinks (STMX). Use the Connect tool and press the Create TransportLink button STM. Then, just like the Topology Designer and the Physical Connections, drag a line from one equipment to the other. That will start a wizard.

To change the connection color, use the Data Model Manager. Select the class corresponding to the desired TransportLink and change the attribute *color*.

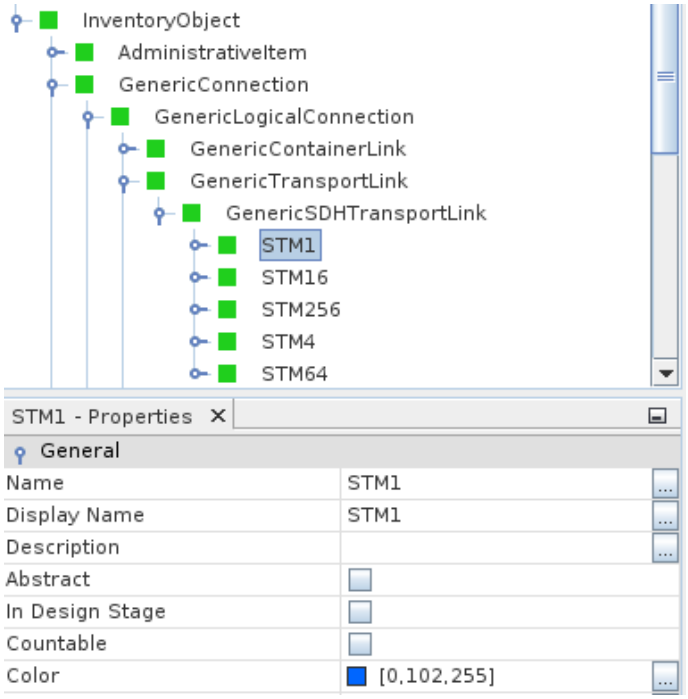


Figure 73: Changing the color for STM1 links

The first step of the wizard will request for the basic information: name and type

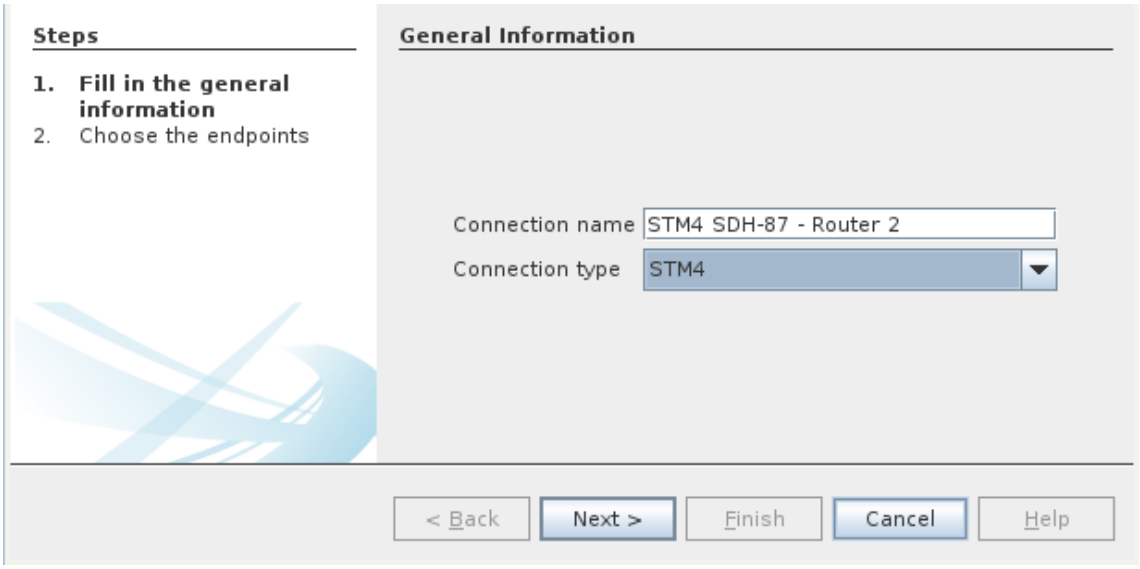


Figure 74: Creating a TransporLink, step 1

The second will request for the the connection endpoints, which will always be aggregate ports.

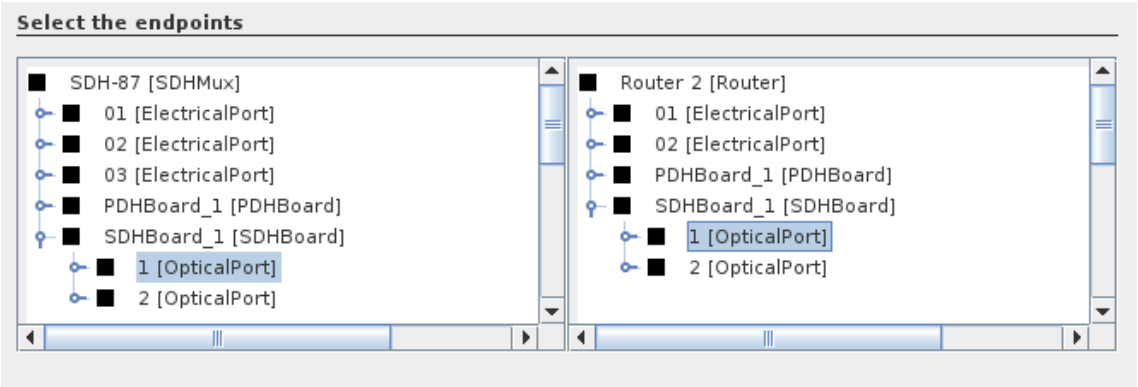


Figure 75: Creating a TransporLink, step 2

After repeating the procedure three more time, the ring will be finished. In the figure 76, the SMT4 links are colored light purple, while the SMT1 ones are blue.

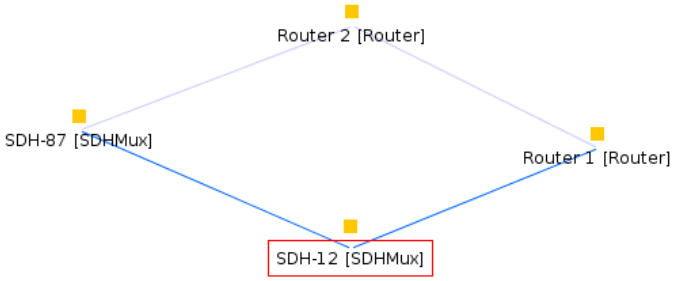


Figure 76: SDH ring

The next level in the hierarchy is the set of structured high-order circuits (VC4) that will contain the E1s that we will add later. We will create three VC4: One long, **from SDH-87** to **Router 1**, other short, from **SDH-87** to **Router 2**, and finally another short between **Router 2** and **Router 1**. This will provide two paths to route the E1s later on. To create the VC4 connections (a.k.a. ContainerLinks), we will leave the Connection tool selected and press the *Create ContainerLink* button  . To create the first ContainerLink, let's drag a line between the **SDH-87** and the **Router 1**. That will start a wizard.

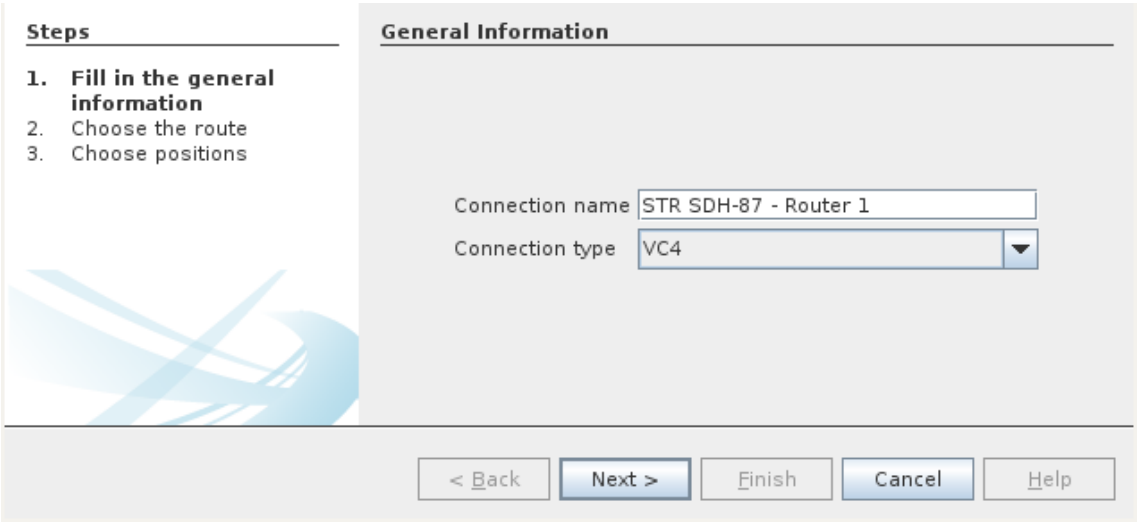


Figure 77: New ContainerLink, step 1

Again, the first step is about basic information: Container name and type. We will create a simple VC4. The next step is about routes. You will find two routes. In the center panel, the detail of the chosen route will be displayed.

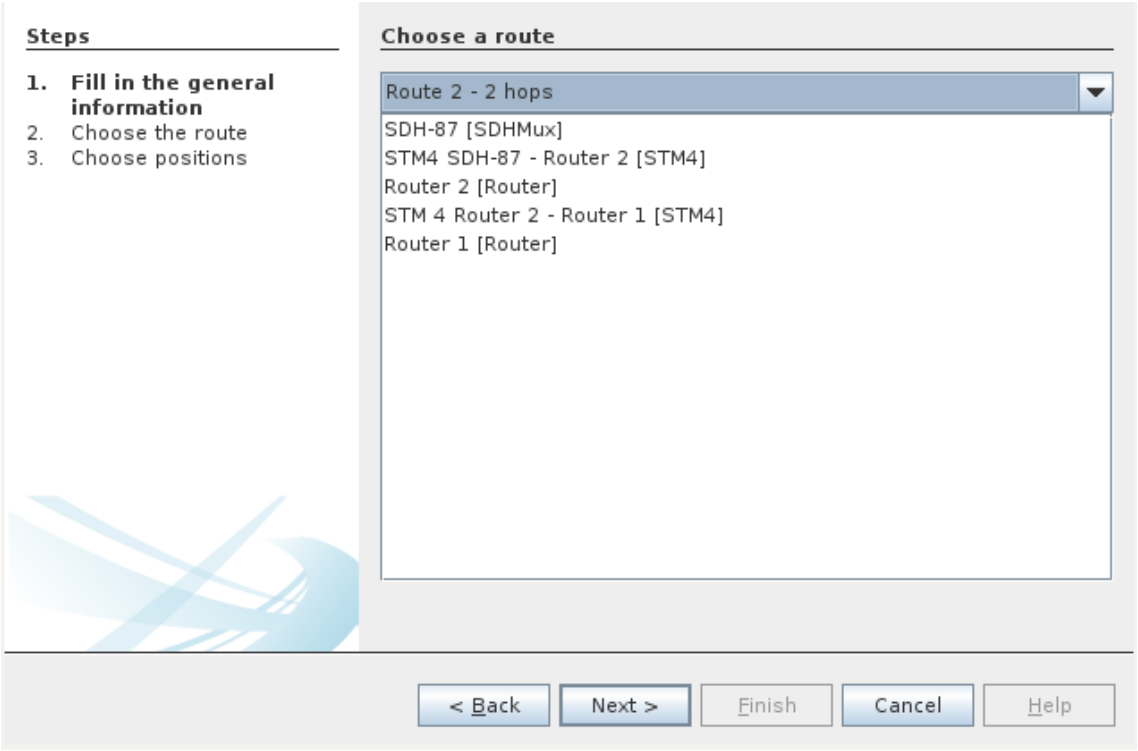


Figure 78: New ContainerLink, step 2

For this example, we're going to use the lower branch composed by the STM1 TransportLinks. The next step will ask you to define what timeslots will you use from each TransportLink in the route(figure 79). Since the TransportLinks are STM1, there will be only one slot available. To select the timeslot, double-click the desired STM, and select the timeslot from the list (figure 80).

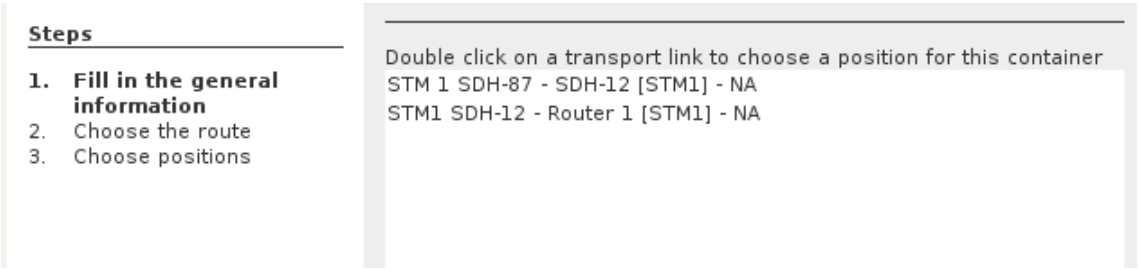


Figure 79: New ContainerLink, step 3

This list will be populated with the available timeslots, but besides, it will inform you what timelots are already in use and who's using them.

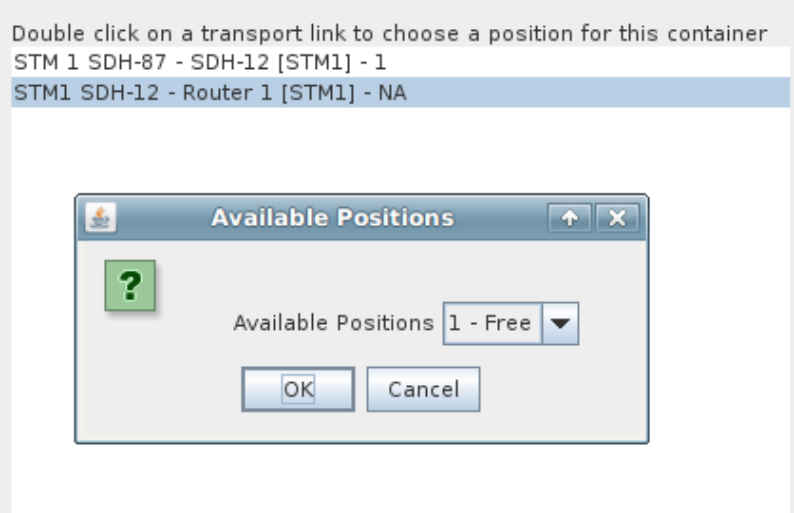


Figure 80: New ContainerLink, step 2

Note that after choosing a timeslot the label next to the STM's name will change from *NA* to the number you chose. It's not necessary to select any endpoint ports, since they're taken from the underlying TransportLinks. Once created, you won't see anything different in the view, since the only connections to be displayed are the TransportLinks.

The last step is to create a TributaryLink, that is, the actual E1 to be delivered. For that, we have to press the Create Tributary button  . As usual, drag a line between the end equipment, in this case, SDH-87 and Router 1. This will open another wizard.

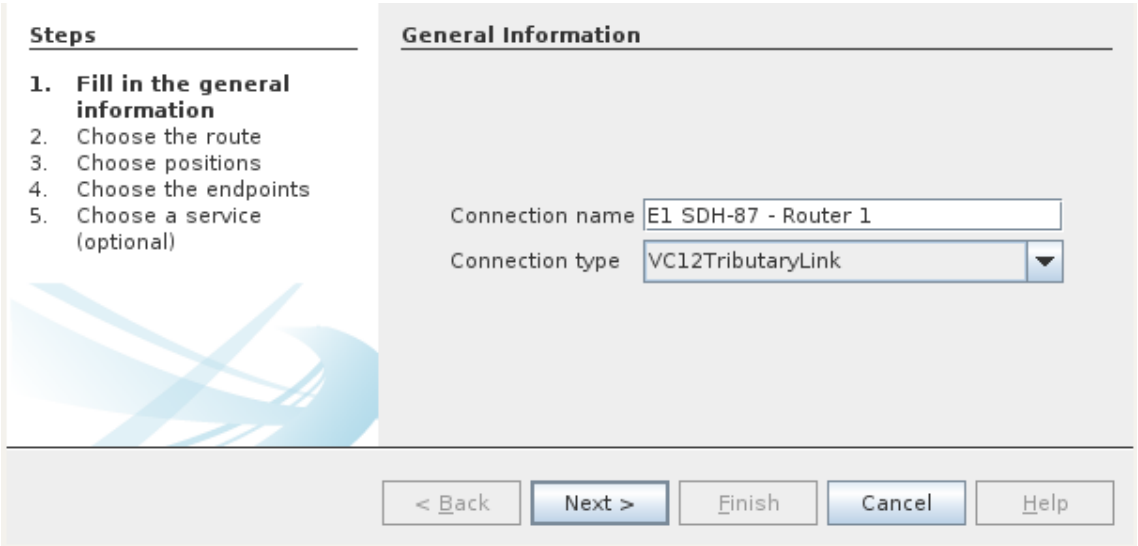


Figure 81: New TributaryLink, step 1

In the next step, it's requested to pick a route. Note that the STM4-based route won't be offered, as it doesn't have ContainerLinks to transport the TributaryLink.

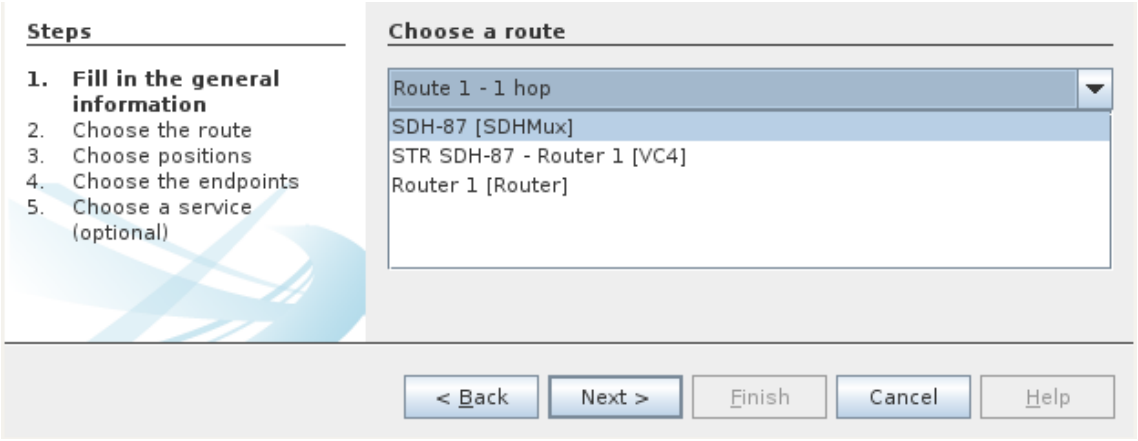


Figure 82: New TributaryLink, step 2

The third step consists of choosing the timeslot in the ContainerLink (VC4)

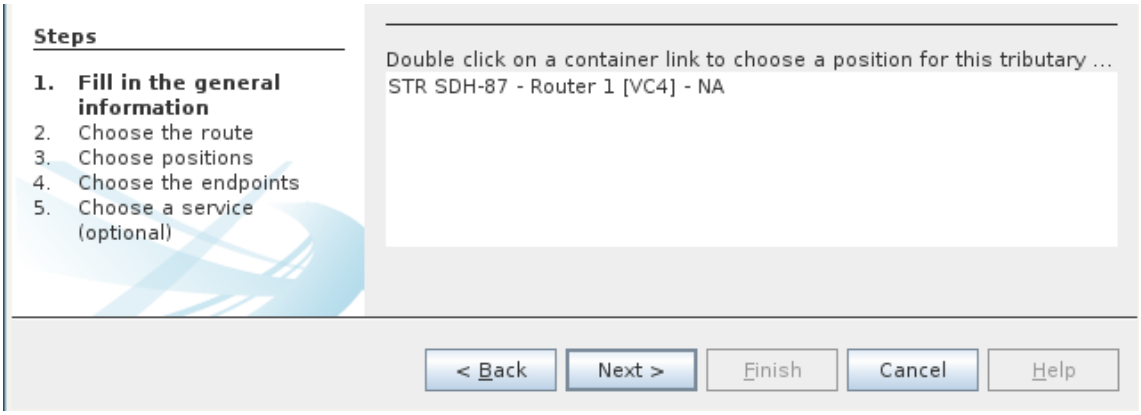


Figure 83: New TributaryLink, step 3

Again, by double-clicking the ContainerLink, you can pick a timeslot

Figure 84: Picking a timeslot from all the available ones in the ContainerLink(VC4)

The next step is choosing the endpoints. For this example, we're going to end the SDH connection in a tributary port of a PDHBoard. It, of course, could end in any port of any board, however this is the most common scenario.

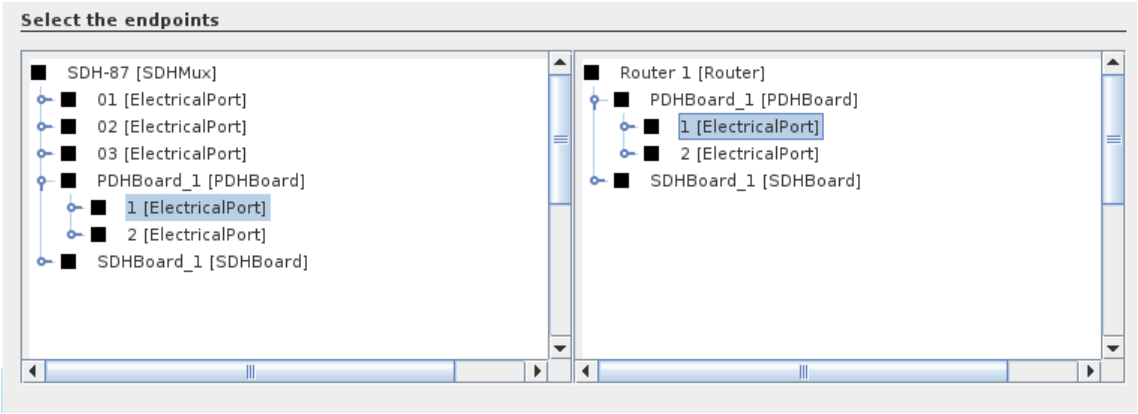


Figure 85: Picking a timeslot from all the available ones in the ContainerLink(VC4)

Finally, as an option, you can associate the newly created TributaryLink to a service instance. Note that this list will be populated **only** with objects of subclasses of **GenericSDHService** previously created in the Service Manager, as seen in the figure 87.

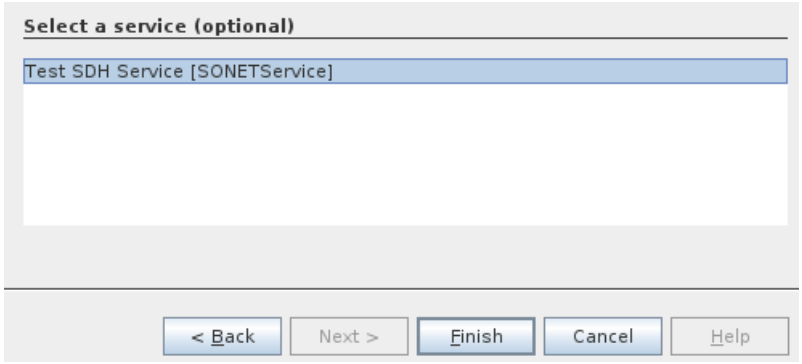


Figure 86: Optionally, relating the circuit to a service

To see the resources used by the circuit, right-click on it either in the search results or the service manager and select the option Reports → TributaryLink Resources.

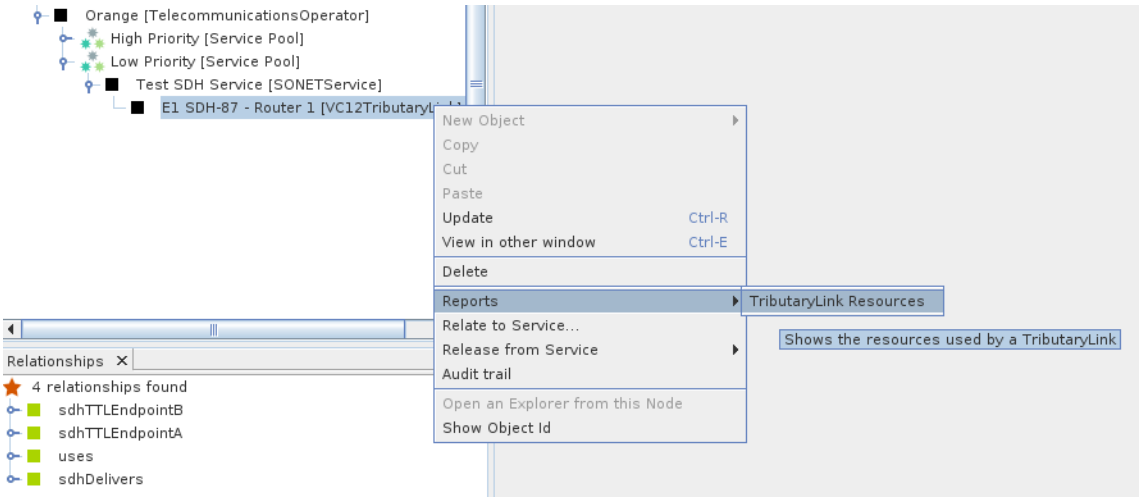
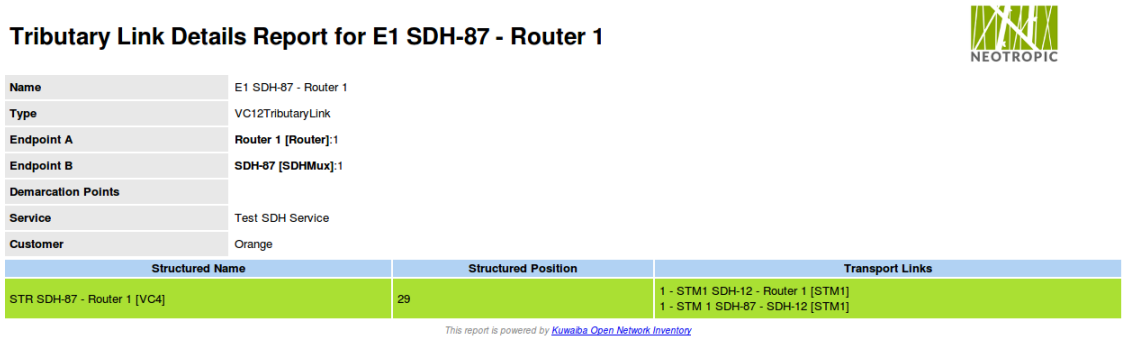


Figure 87: SDH service in the Service Manager and report execution

The result shows the timeslots, TransportLinks involved, end ports, services associated and optionally, demarcation points (next physically connected ports, usually in an ODF/DDF)



Tributary Link Details Report for E1 SDH-87 - Router 1

NEOTROPIC

Name	E1 SDH-87 - Router 1
Type	VC12TributaryLink
Endpoint A	**Router 1 [Router]**:1
Endpoint B	**SDH-87 [SDHMux]**:1
Demarcation Points	
Service	Test SDH Service
Customer	Orange

Structured Name	Structured Position	Transport Links
STR SDH-87 - Router 1 [VC4]	29	1 - STM1 SDH-12 - Router 1 [STM1] 1 - STM 1 SDH-87 - SDH-12 [STM1]

This report is powered by Kuwaiba Open Network Inventory

Figure 88: Report results (zoom in to read better)

Important
As an exercise for the reader, create Container and TributaryLinks using the alternate STM4-based route.