

User guide for the FBQL Editor

Table of contents

1 Purpose and Installation of FBQL Editor	3
2. Starting the program, general toolbar	4
3. Creating a basic function block using JavaScript	8
4. Creating a SPARQL block and the corresponding SPARQL query diagram in the graphical editor	10
5. Configuring and connecting to the SPARQL server	13
6. Creating a composite function block	15
7. Using GraphViz to visualise the execution of the function block diagram	18
8. Test case of creating a project from basic blocks	19
9. Test case of a simple SPARQL block	21
10. Test case of creating a complex project with different types of blocks	23

1. Purpose and Installation of FBQL Editor

Purpose of the program

The program is designed to create, edit and execute function block diagrams (FBD) over RDF data store. For example, FBQL Editor can be used for assessing the properties of systems represented as RDF-based ontology. The FBD can include three types of (function) blocks: basic, composite and SPARQL. It is possible to use both library and custom blocks. In the current version, SPARQL blocks have only one output representing the number of the resulting tuples.

Location

To install FBQL Editor, go to the repository on GitHub
<https://github.com/FBQLEditor/FBQLEditor>

OS Requirements

The program is guaranteed to work on Windows 10 and Ubuntu 22.10 operating systems, on other operating systems, work is allowed, but not guaranteed.

For the program work, at least 500 MB of free RAM is required.

For an OS running on a Linux kernel, you need to install the graphviz library.

How to install

Windows 10: The Executable folder contains the program installers, installation and operation are guaranteed on a clean system (without additional installation of third-party programs).

Ubuntu 22.10: Installation is done from source, you will need to install the Qt Creator program at least version 5.0.2, as well as install additional libraries using the command:

```
sudo apt-get install qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools sudo  
apt-get install qt5-qmake qtdeclarative5-dev qtscript5-dev libqt5svg5-dev  
sudo apt-get install graphviz-dev  
sudo apt-get install libqt5webkit5-dev qtwebengine5-dev
```

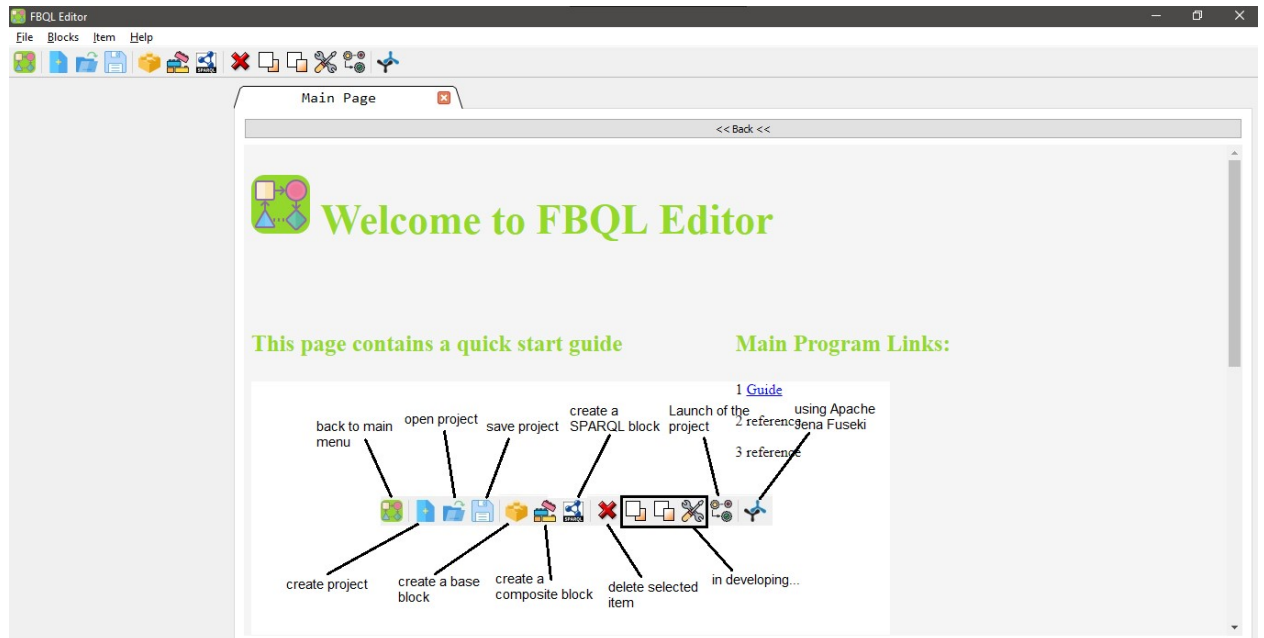
How to run

Windows 10: The startup is performed using the executable file ".exe", which appeared in the folder when installed the program.

Ubuntu 22.10: Run the executable file that was obtained when building the project.

2. Starting the program, general toolbar

Once you have opened the program, you will be greeted by a welcome screen:

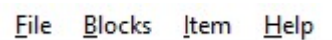



To work with the software, you can use the navigation bar



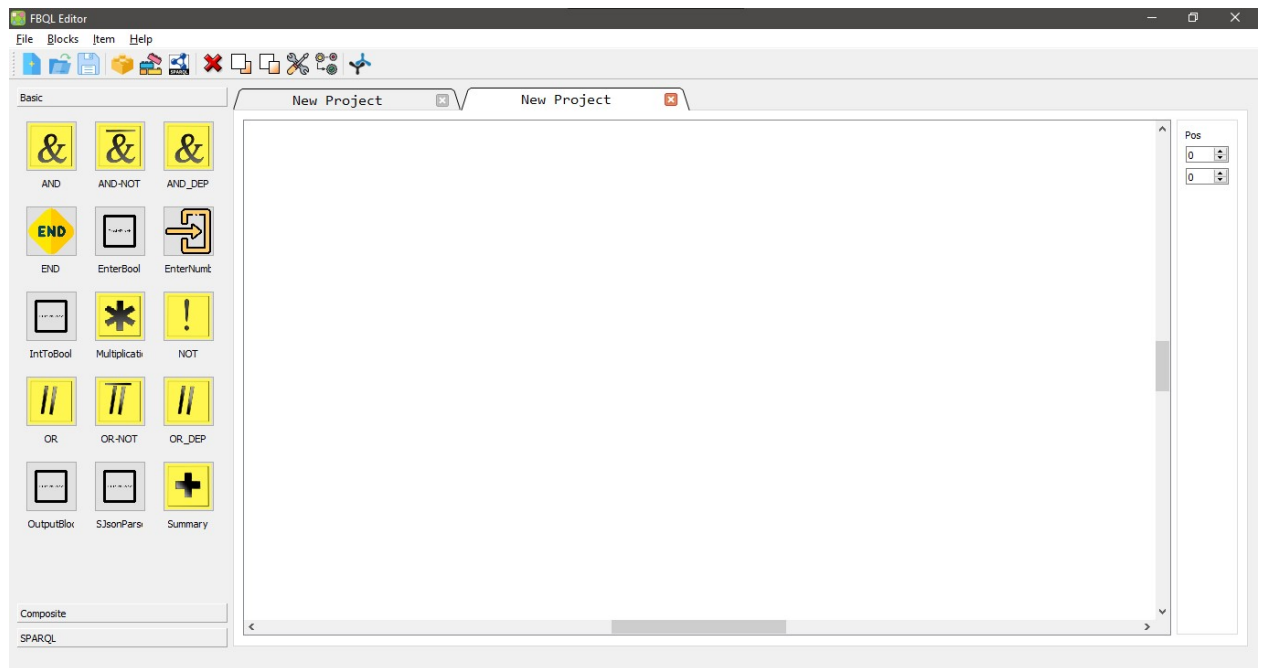
To return to the home page, click on this button:

You can also use the tabs in the upper left-hand corner of the program; by hovering over them, you can select the desired item.

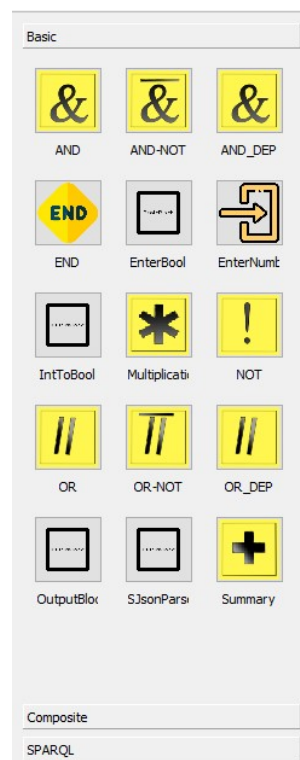


To create a new project, click on this button .

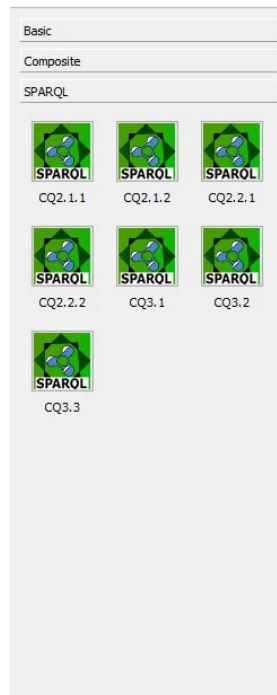
A project creation screen will then appear:



The left-hand side shows all the blocks you can use:



All blocks are divided into 3 tabs: basic, composite and SPARQL. To move between tabs, click on the block type name.

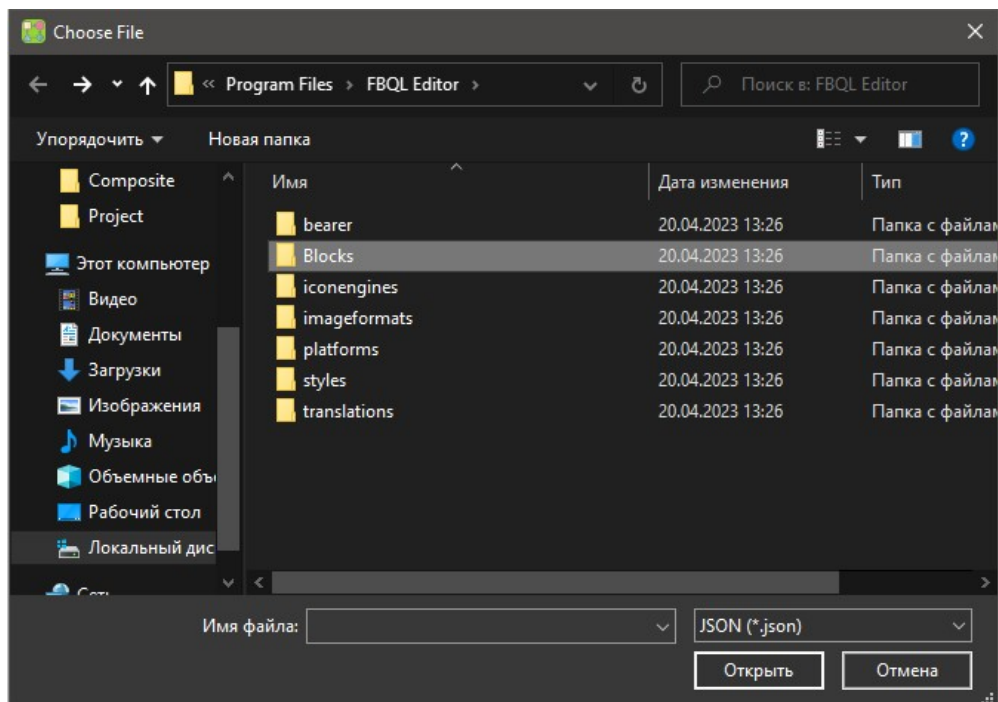


After clicking on the "SPARQL" tab, the corresponding block window opens.

To open existing projects, click on this button:

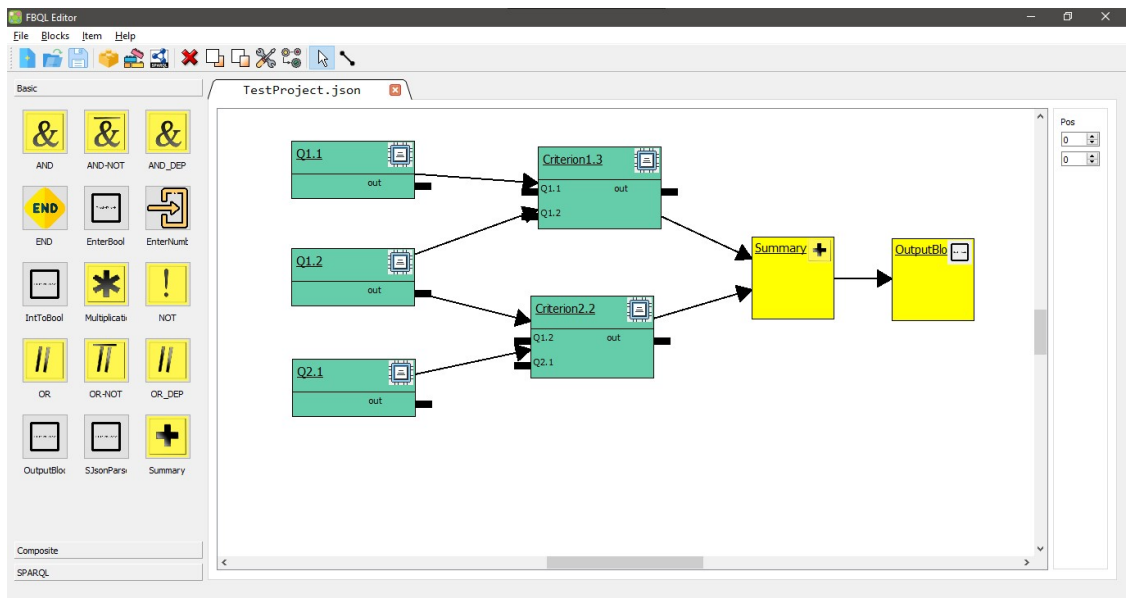



A file selection window will then open:



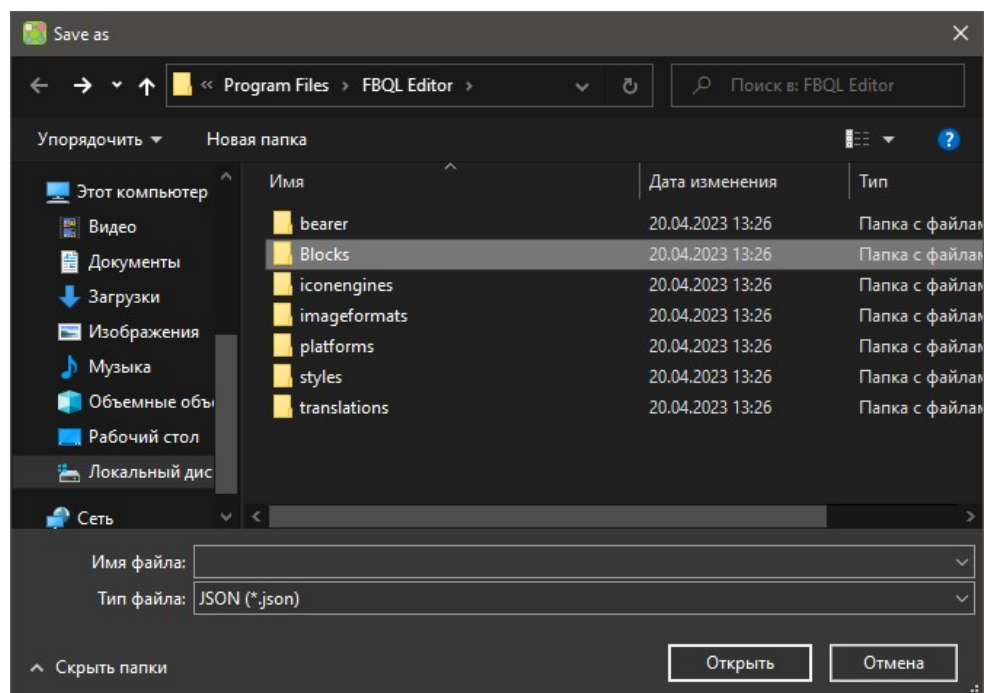
Initially, the projects are saved in the *Documents/FBQLEditor* folder

After opening the project file in *.fbd* format, the project will load to the screen:



In order to save the created project or overwrite the current project, press this button: 

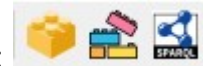
Once pressed, a window will open with a folder selection window:



The format for saving the project is *.fbd*.

3. Creating a basic function block using JavaScript

To create blocks, you need to click on these buttons:



1 - Basis block, 2 - Composite block, 3 - SPARQL block.

The basic block creation screen is below:

The screenshot shows a web interface for creating a new basic block. The title bar indicates 'Main Page' and 'New Basic'. The form includes a 'Name Block:' field with the value 'BasicBlock'. Below it are checkboxes for 'Label' (checked) and 'LineEdit' (checked), each followed by a corresponding text input field. To the right of these are buttons for 'Set Image' and 'Delete Image'. At the bottom of the form are buttons for 'Info', 'Test Script', 'Open', and 'Save'. A large text area for the 'Script:' is located at the bottom of the form.

To create a basic block, you must specify a block name:

A close-up of the 'Name Block:' field, which contains the text 'Name'.

It is also possible to insert additional information:

A close-up of the 'Label' and 'LineEdit' checkboxes and their corresponding text input fields. The 'Label' checkbox is checked, and the 'LineEdit' checkbox is also checked.

The main part is the "Script" field, where the block logic will be described. The code must be written in JavaScript.

A close-up of the 'Script:' text area, which is empty and ready for input.

These buttons are needed to open and save the block:

Buttons for 'Open' and 'Save'.

As an example, consider the implementation of a "logical OR" block.

The following JavaScript code is used to implement this block.

Script:

```
output = "false";
for(var i = 0; i < inputs.length; i++){
    if( inputs[i] == "true"){
        output = "true";
        break;
    }
}
```

The inputs of this block are represented as an array of Boolean values - "inputs". The result will be written to the variable "output". The logic is as follows: run through all the values of the input array, and if at least one value is true, then the result of the block will be true, otherwise false.

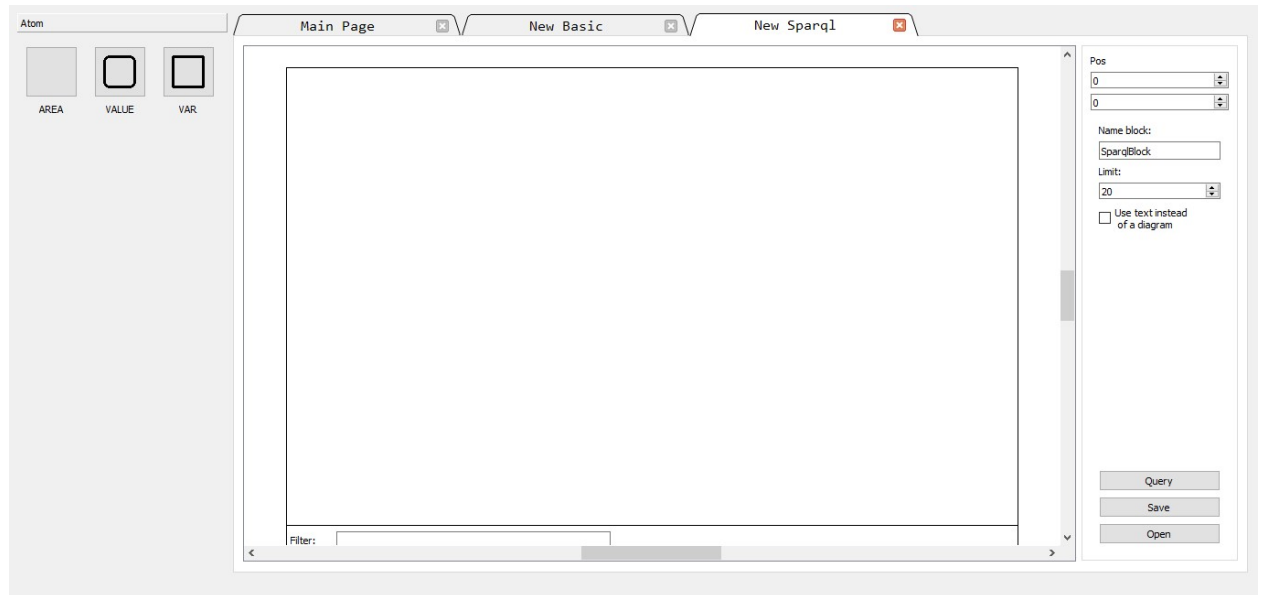
The following code section is used to set up block dependencies with other blocks and for the decision explanation method to work:

```
// depends
for(var i = 0; i < inputs.length; i++){
    if(inputs[i] == output){
        causes.push(i);
    }
}
```

In order for a block's input to be written to the dependency array - 'causes', the input value must be equal to the output value, as these are the inputs that affect the block's operation.

4. Creating a SPARQL block and the corresponding SPARQL query diagram in the graphical editor

SPARQL block creation window is below.



The AREA rectangle is a place where you can create a graph pattern consisting of triple patterns.



AREA rectangles can be connected together, using the appropriate SPARQL graph patterns combining operations (for example, MINUS or UNION). There is a Filter field inside this rectangle to create a FILTER construct in a SPARQL query. Anything inside this field will go into the FILTER statement in the resulting textual query.

The "VALUE" node (the square with rounded edges) is needed to create a literal (constant value) in a triple.

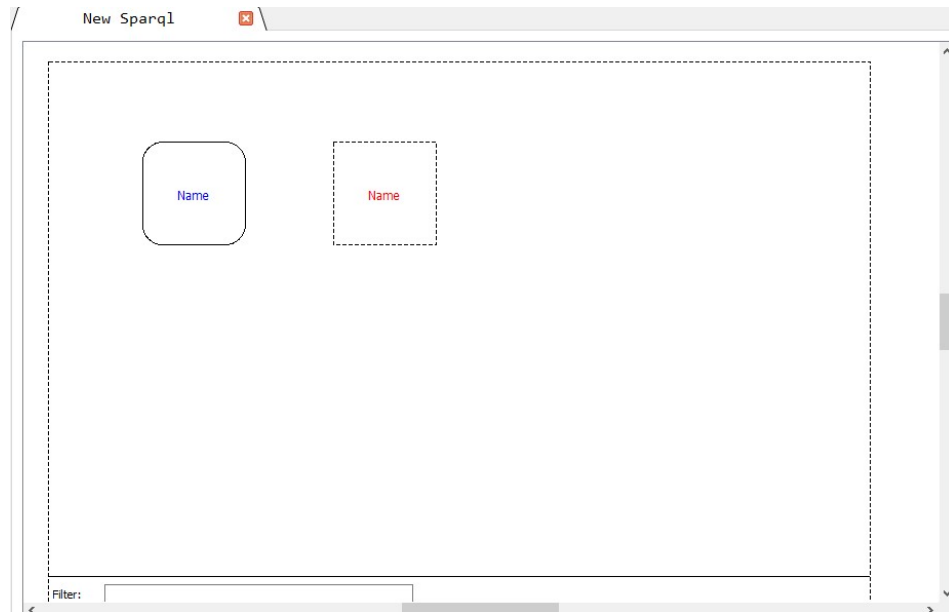


The "VAR" node (the square with straight edges) is needed to create a variable in a triple.



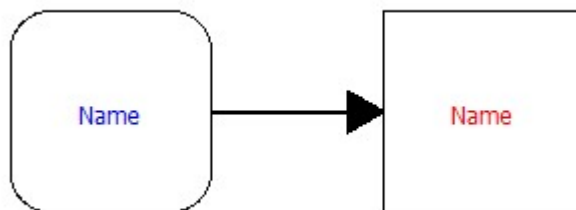
To move the desired node onto the area, you need to click on it and then click anywhere on the area for the node to appear.

Once a node has been set up, you must assign its attribute. To do this, click on the middle of the node and enter the attribute.



To get a triple, you need to connect two nodes together. You can do this by right-clicking on the start node and moving the arrow to the end node. To give an attribute to an arc, left-click on the arc and enter its attribute in the text box.

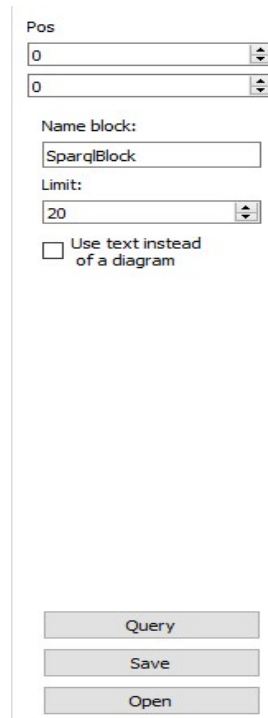
One of the triple patterns is below:



In this construction, the left node is the subject, the arc is the predicate and the right node is the object (here it is a literal).

To connect the two "AREAs", you must draw an arc from the first area to the second one, and click on the arc and name it with a specific keyword (MINUS, UNION).

The right-hand pane contains 3 buttons and a text box for naming blocks when saving:



Pos
0
0

Name block:
SparqlBlock

Limit:
20

☐ Use text instead of a diagram

Query
Save
Open

The "Query" button allows you to write the query yourself in text form, after pressing it, a text box will appear for entering the query. For this type of query to work, you must tick the "Use text instead of a diagram" box.

The "Save" button is used to save the created SPARQL block for later use. The "Open" button allows you to open an already saved SPARQL block.

SPARQL blocks have only one output and return only one value - the number of resultant tuples.

The appearance of the SPARQL block after it has been created is as follows.



5. Configuring and connecting to the SPARQL server

To work with the SPARQL server "Apache Jena Fuseki" there is a tab "Fuseki" :



IP-adress Fuseki Server:

Port Fuseki Server:

Dataset Name:

Prefixes:

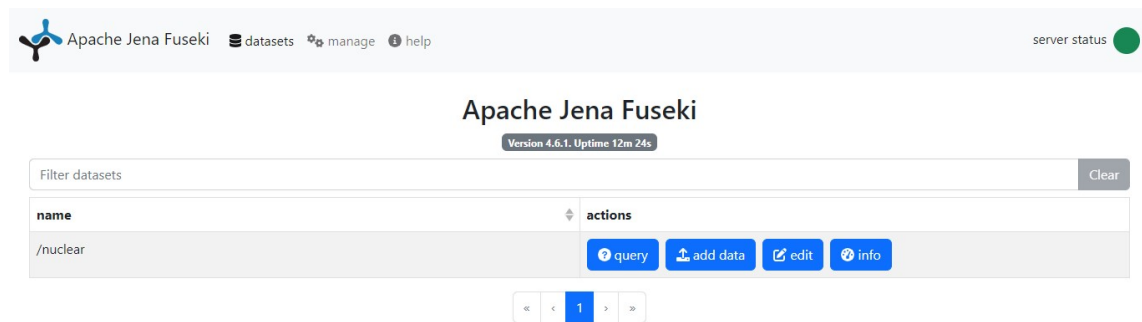
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX : <http://www.semanticweb.org/SEARCH/ontologies/2021/4/OICA/Classes#>
PREFIX USEPR: <http://www.semanticweb.org/SEARCH/ontologies/2021/4/OICA/USEPR#>

The "IP-adress Fuseki Server" text box is needed to enter the IP address of the server.

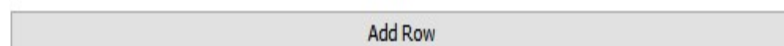
The "Port Fuseki Server" text box is needed to set the server operation port.

The text field "Dataset Name" contains the name of the data set associated with RDF ontology used.

Apache Jena Fuseki Web interface is shown below. Any Web browser can be used to configure the SPARQL server and upload the used RDF ontology to it.



The Prefixes field in FBQL Editor is used to set shortenings for the used IRI. To add a prefix, click on the "Add row" button:



Next, on the line that appears, you must write the prefix in angle brackets:



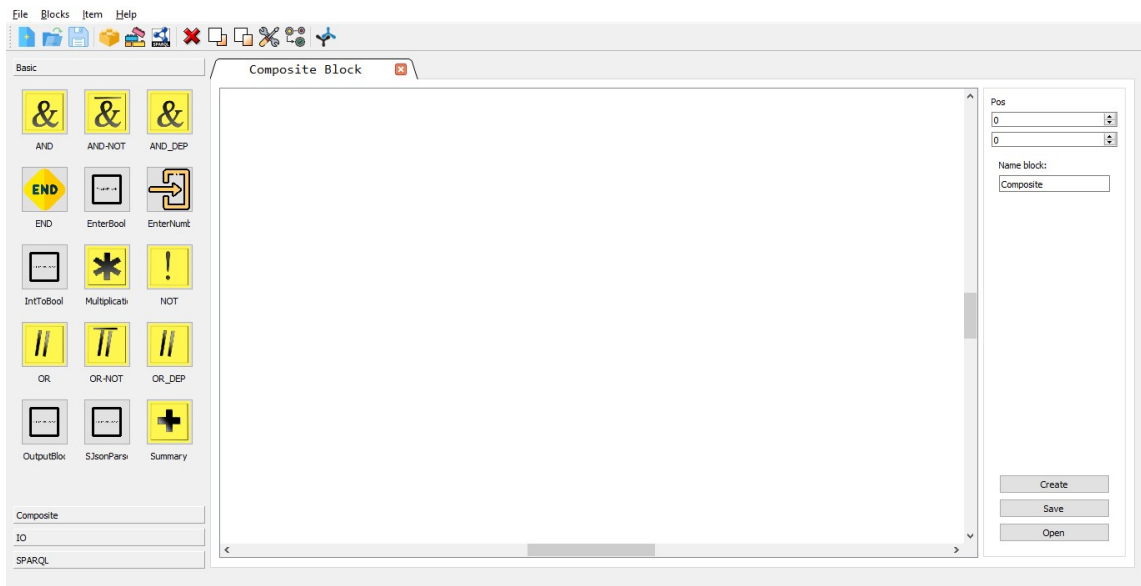
To delete a row with a prefix, use the "Remove row" button:



To run the SPARQL server on Windows, click on the *.bat* file in the server folder installed with the project.

6. Creating a composite function block

Window for working with composite blocks:



On the left-hand side of the window are all the available blocks that can be used to create a new composite block.

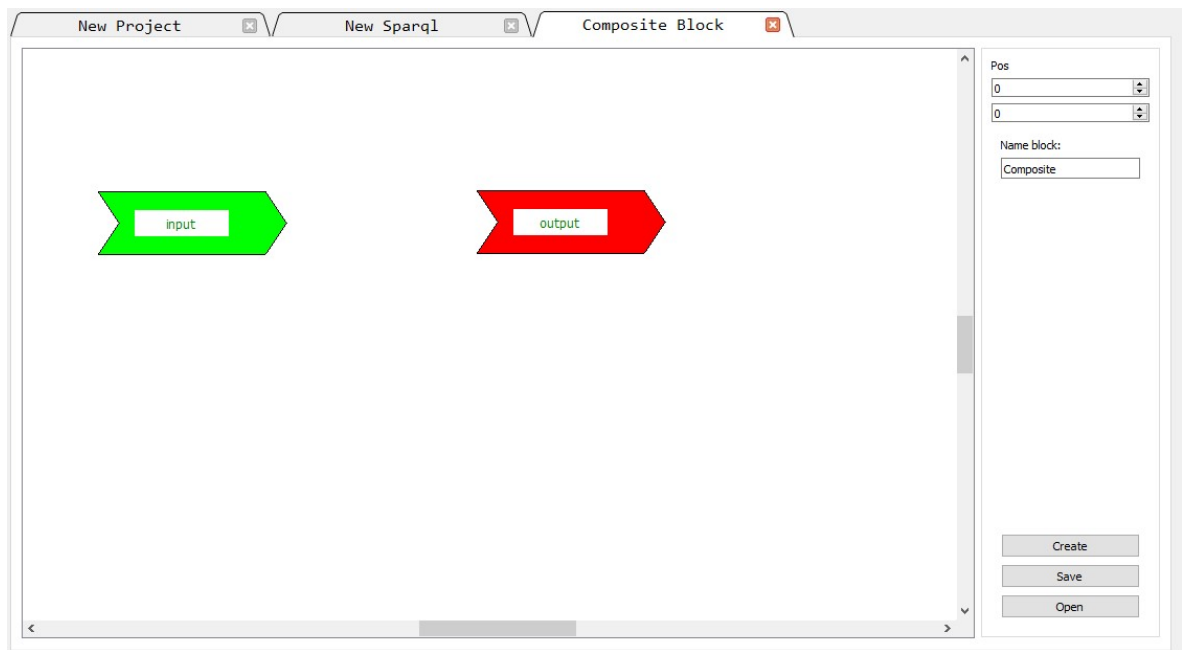
Similar to the SPARQL block, to place a block on the stage, left-click on the selected block, then left-click anywhere on the stage to display the block.

The composite block has no restrictions on inputs or outputs. In order to make an input or output for the composite block, you must click on the "IO" tab:



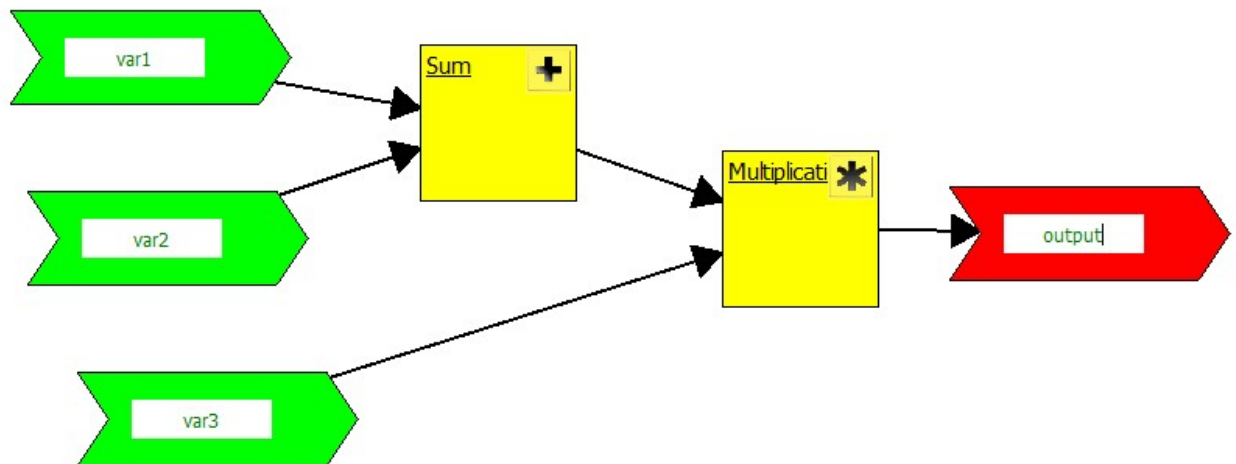
These elements are the inputs and outputs of a net of (component) blocks representing the content of the corresponding composite block.

This is what these elements look like on stage.



The (component) blocks are interconnected inside the net.

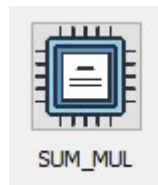
An example of a composite block that adds up two values and then multiplies them by a third value is below:



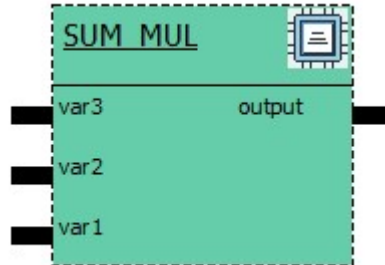
There are three buttons in the right-hand panel.

Button "Save" - To save the created block.

This is what the composite block will look like after it has been saved in the left-hand block panel:



On stage, the composite block will have the following appearance:



Button "Open" - To open an already saved composite block.

7. Using GraphViz to visualise the execution of the function block diagram

There is a built-in library, Graphviz. It can be used to visually trace the operation of the blocks and their connections. Once the project has been started, the software will draw a diagram of how the blocks are executed and interconnected.

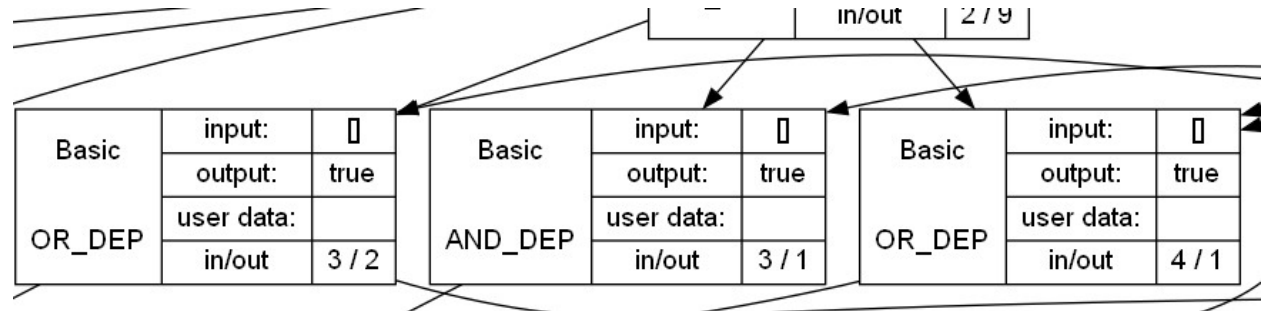
An example of using the Graphviz library.

An example of a block in the library's interpretation is below:

SPARQL CQ3.1	input:	□
	output:	
	user data:	
	in/out	0 / 1

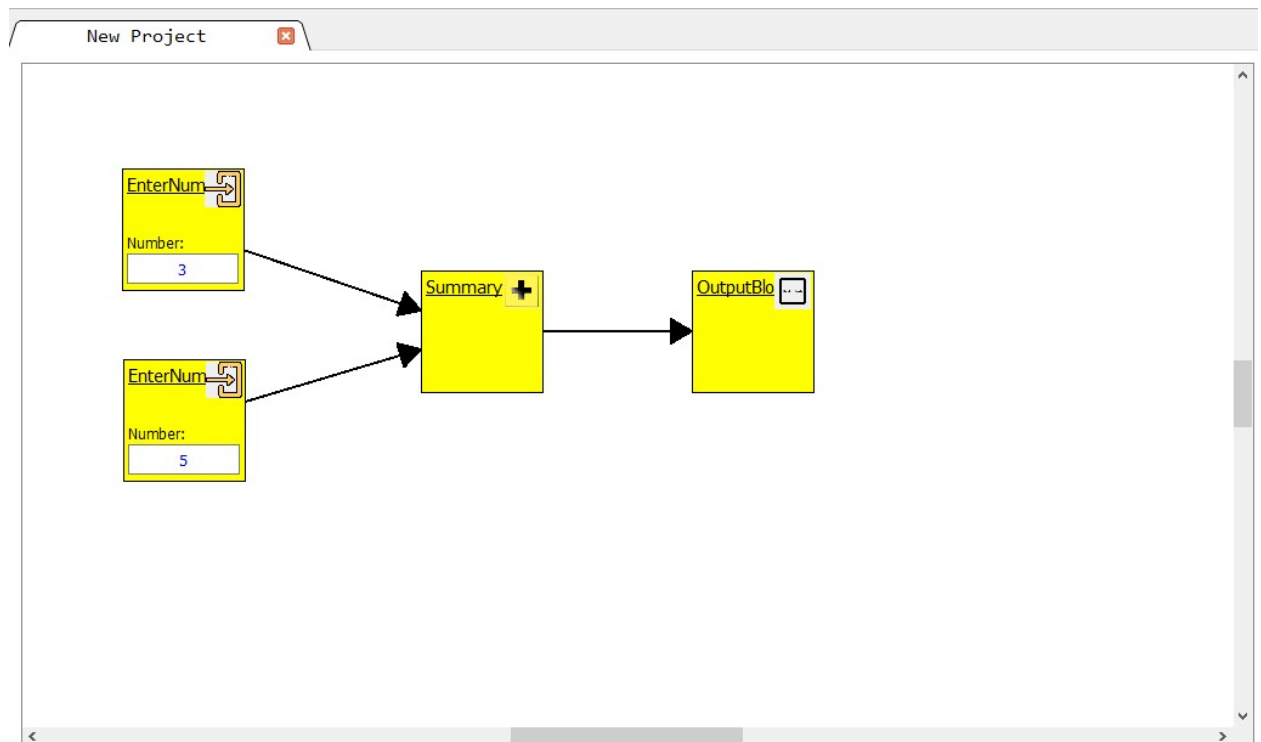
The block includes: its type, name, number of inputs/outputs, input/output values and user data.

An example of block connections is below. That shows the execution order.



8. Test case of creating a project from basic blocks

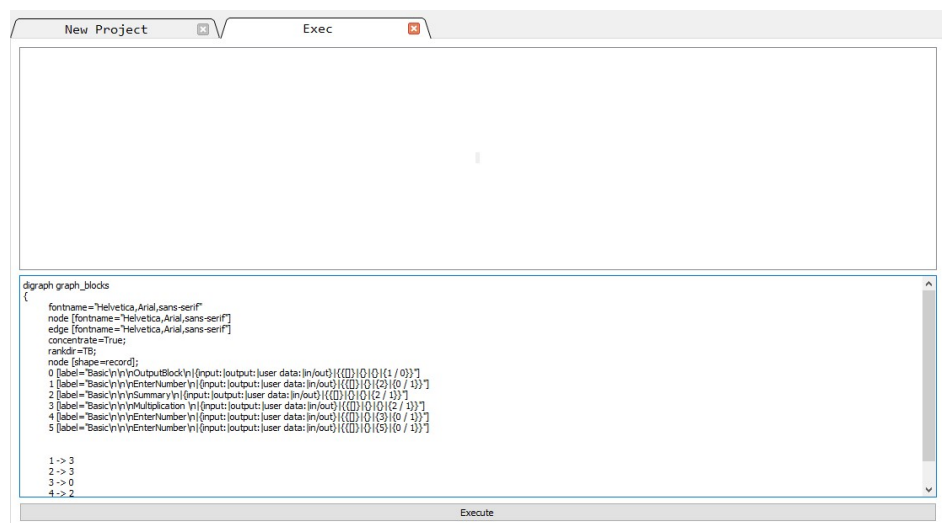
Example of a created project:



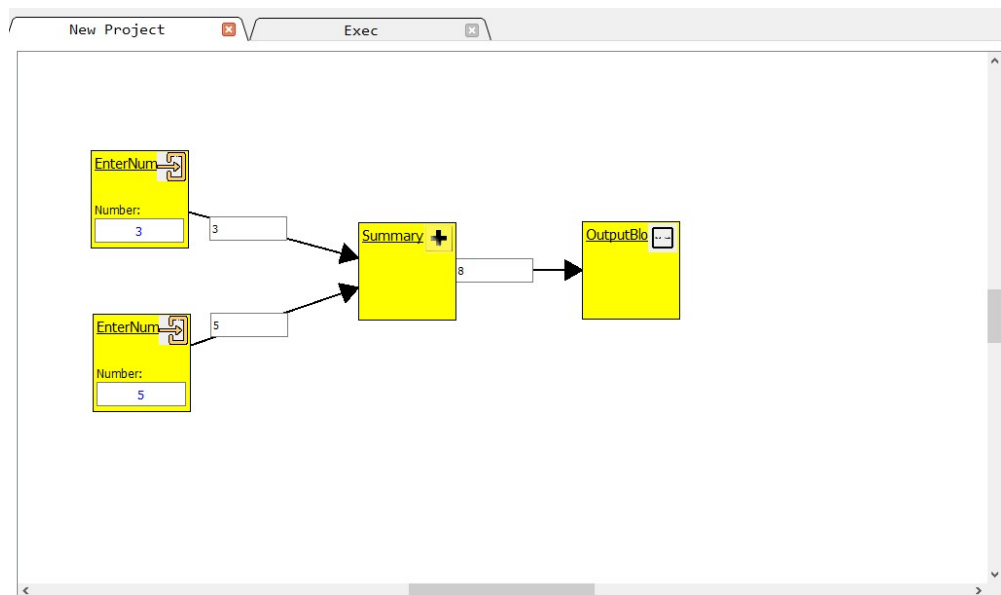
The input has two values, they are added, then multiplied by a third value, the result is output using the "OutputBlock". To run the project, press the "Run"

button: 

The block execution window will open. To run, press the "Execute" button.



Once completed, the inscriptions on arcs will show their output values:



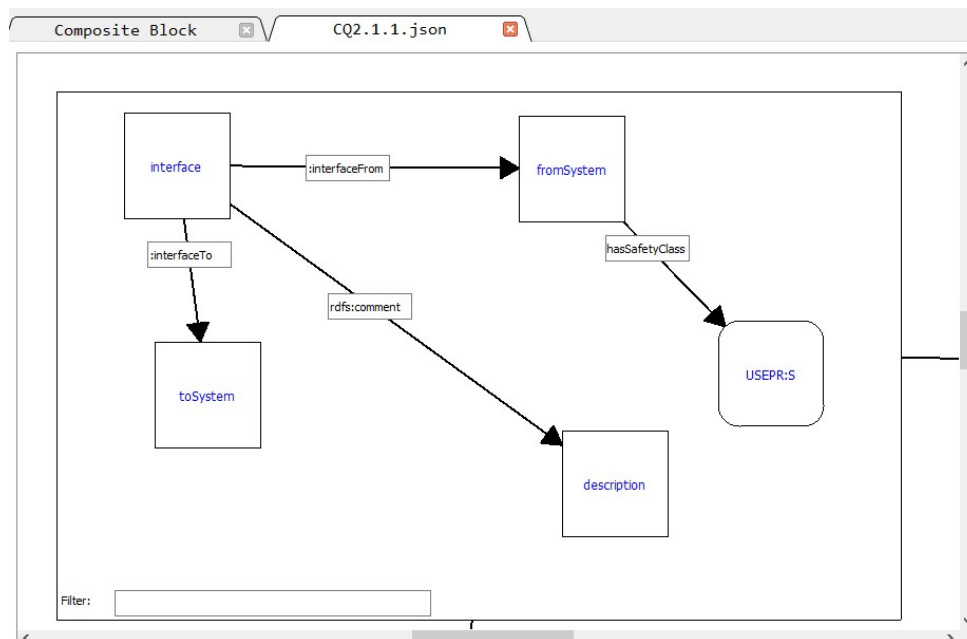
The result will be displayed in the progress window:

8

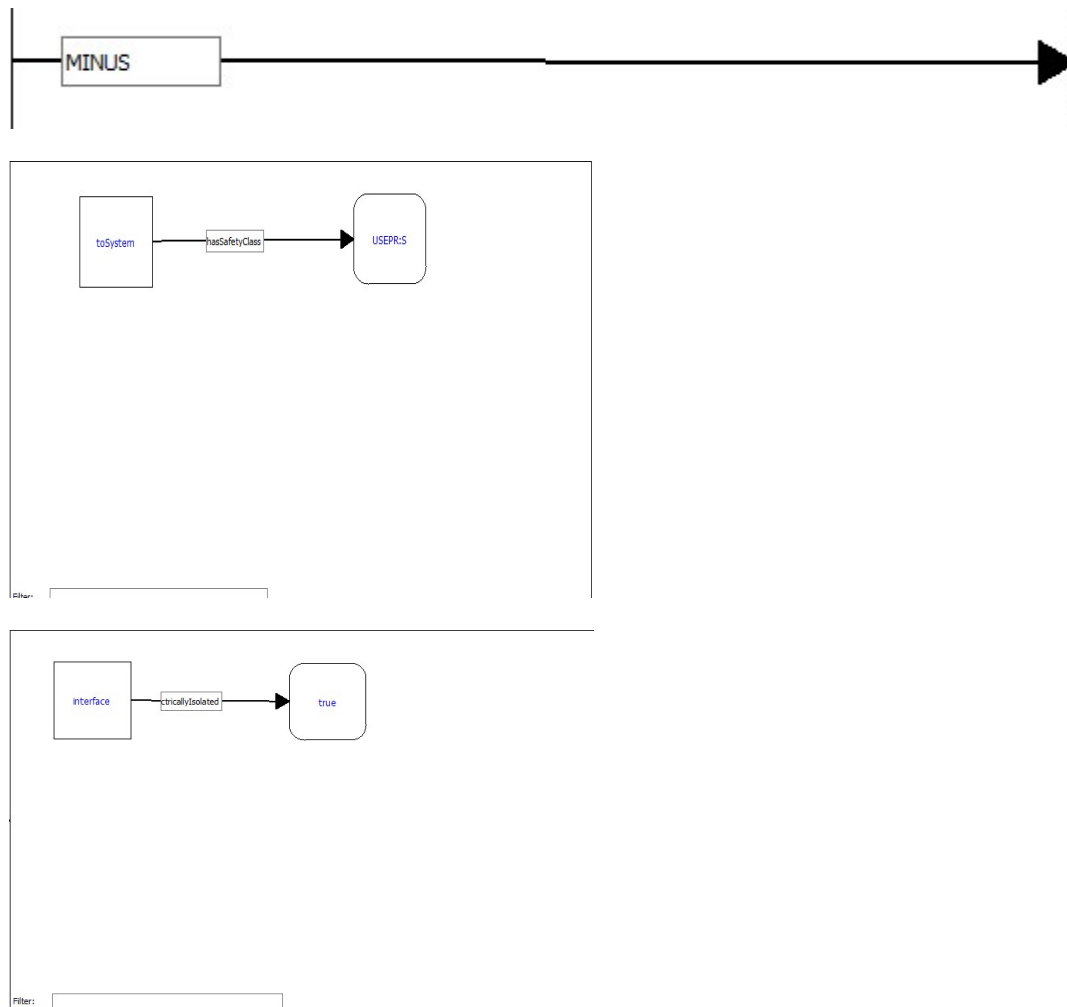
9. Test case of a simple SPARQL block


Example of a simple SPARQL block.

This is the main query area and contains the bulk of the query (in graphical form).

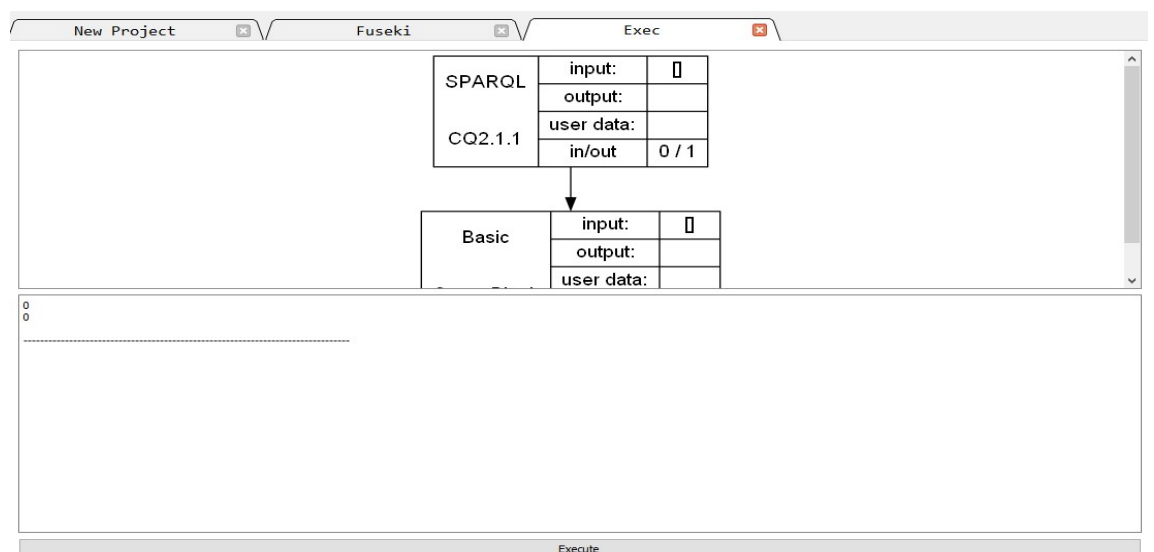


Further, this is connected to the two other areas, by means of a MINUS operation.



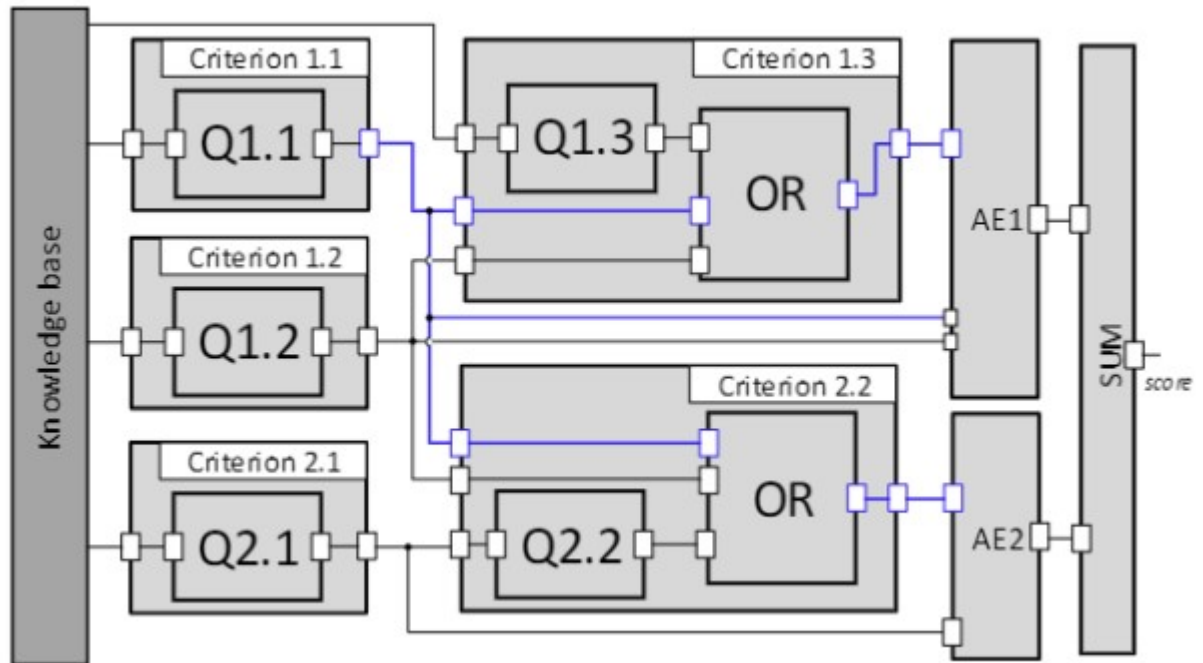
To run the resulting SPARQL query, start the server "Apache Jena Fuseki" see point 5, then click on the "Run" button: 

After clicking Execute, the output window will show the result, and the Graphviz library operation will be visible in the top pane.

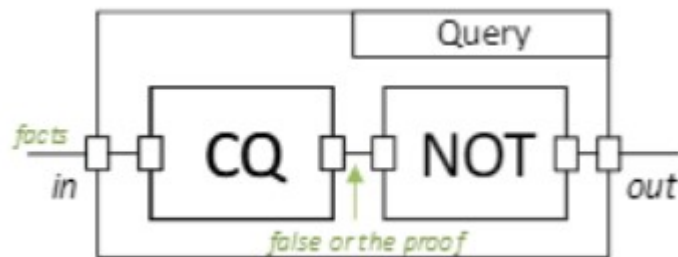


10. Test case of creating a complex project with different types of blocks

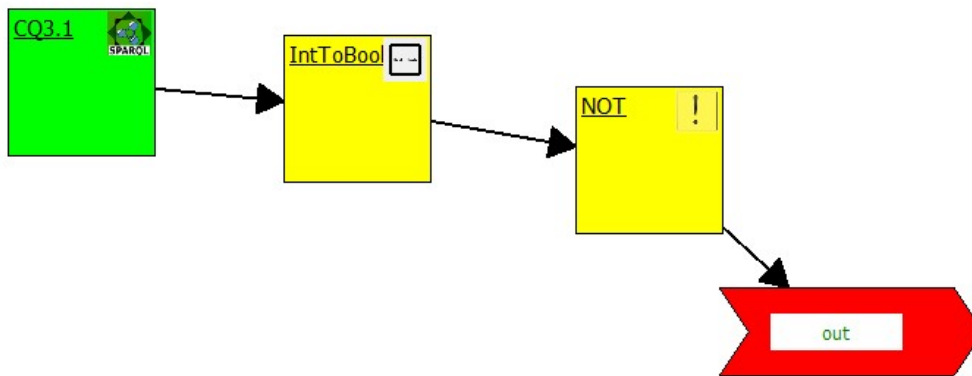
We will use this scheme as the basis for the test case:



Each criterion inside will be a SPARQL query, a conversion block from an integer value to a boolean value, and an inversion block.

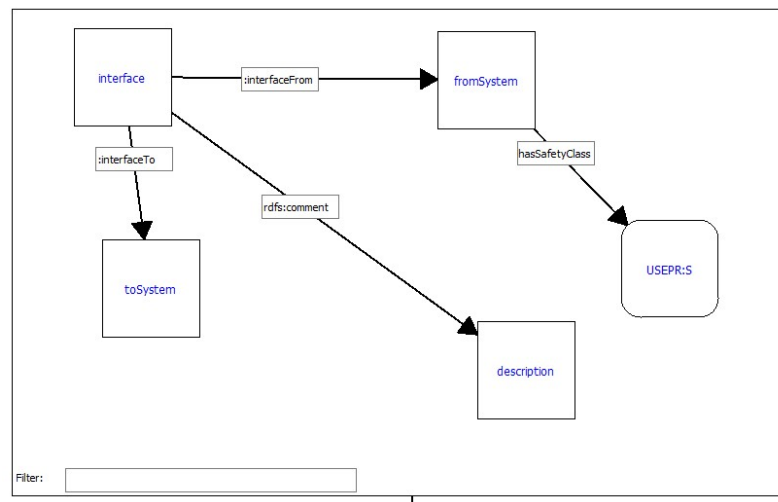


Example Criterion “Criterion 1.1”:

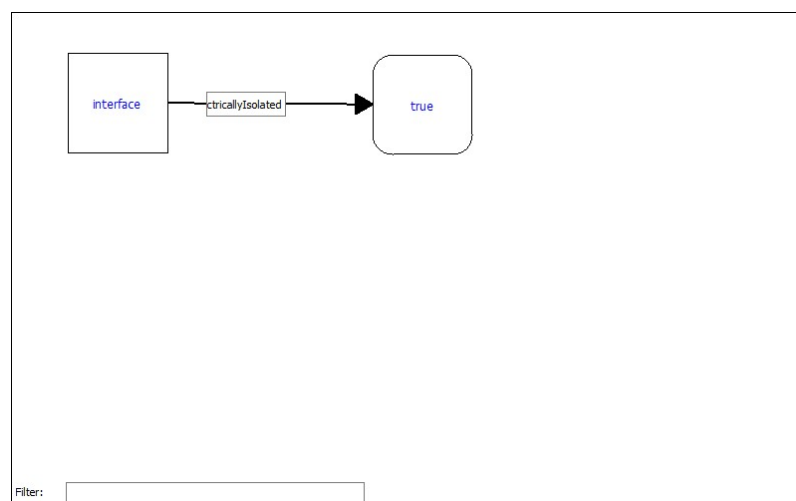


Example query Q1.1:

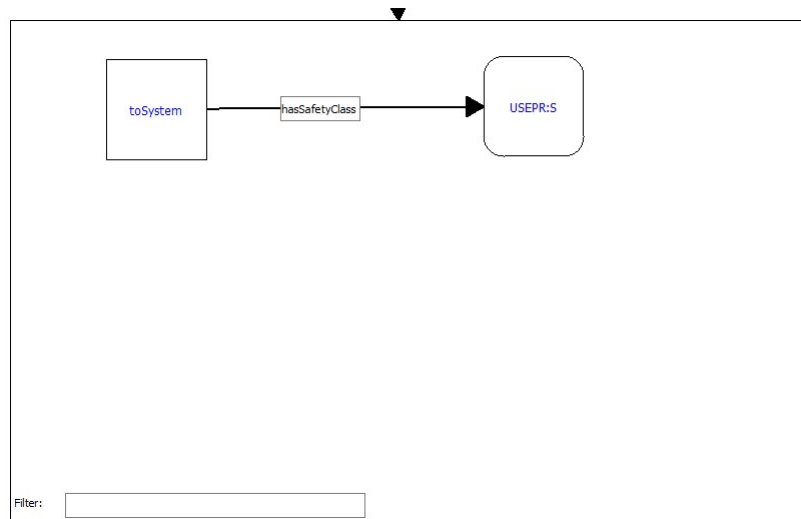
The main body of the request.



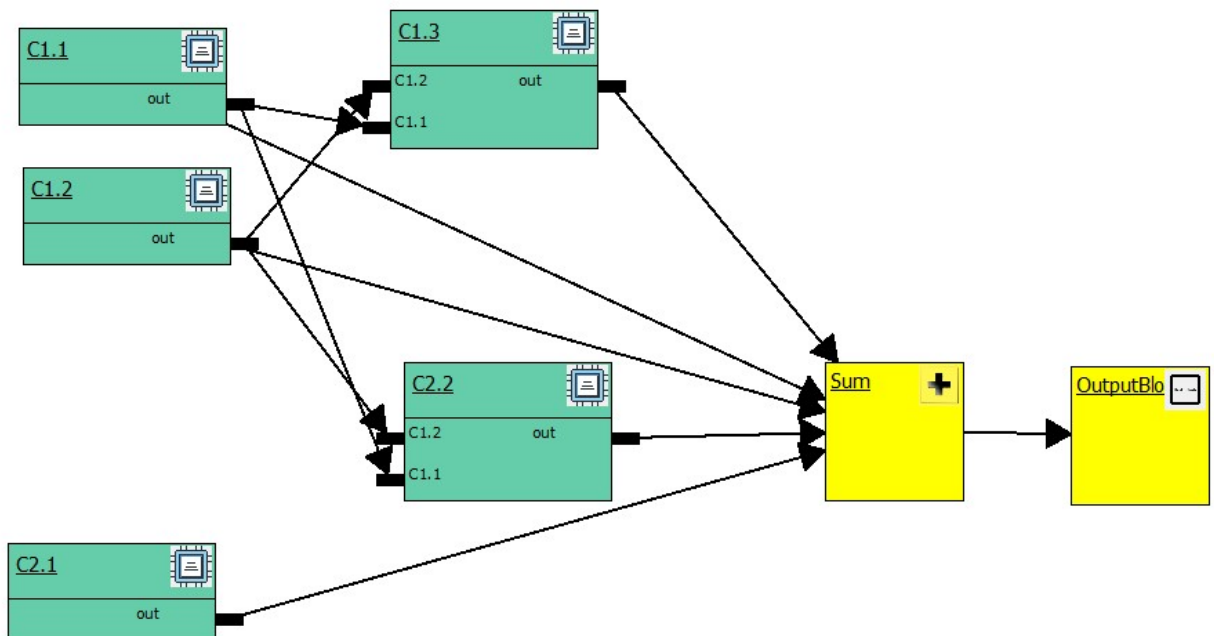
Implementation of the first design with MINUS.



Implementation of the second design with MINUS.



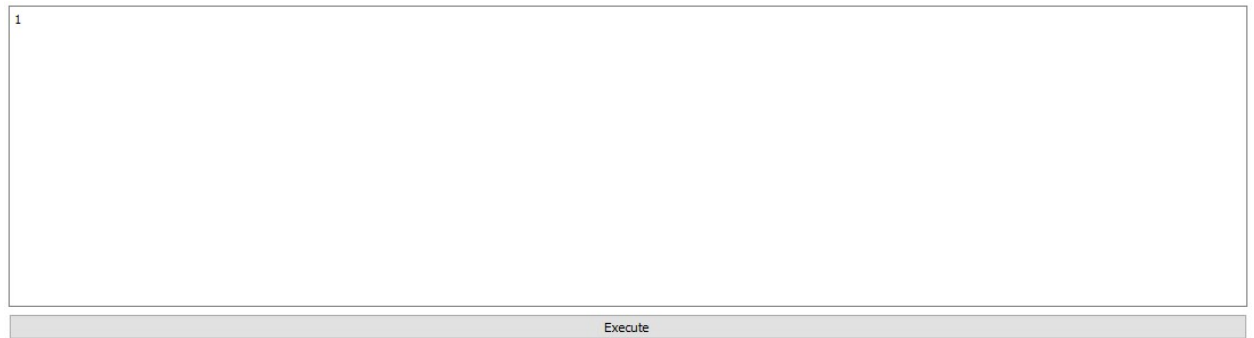
The resulting project is based on composite, SPARQL and basic blocks.



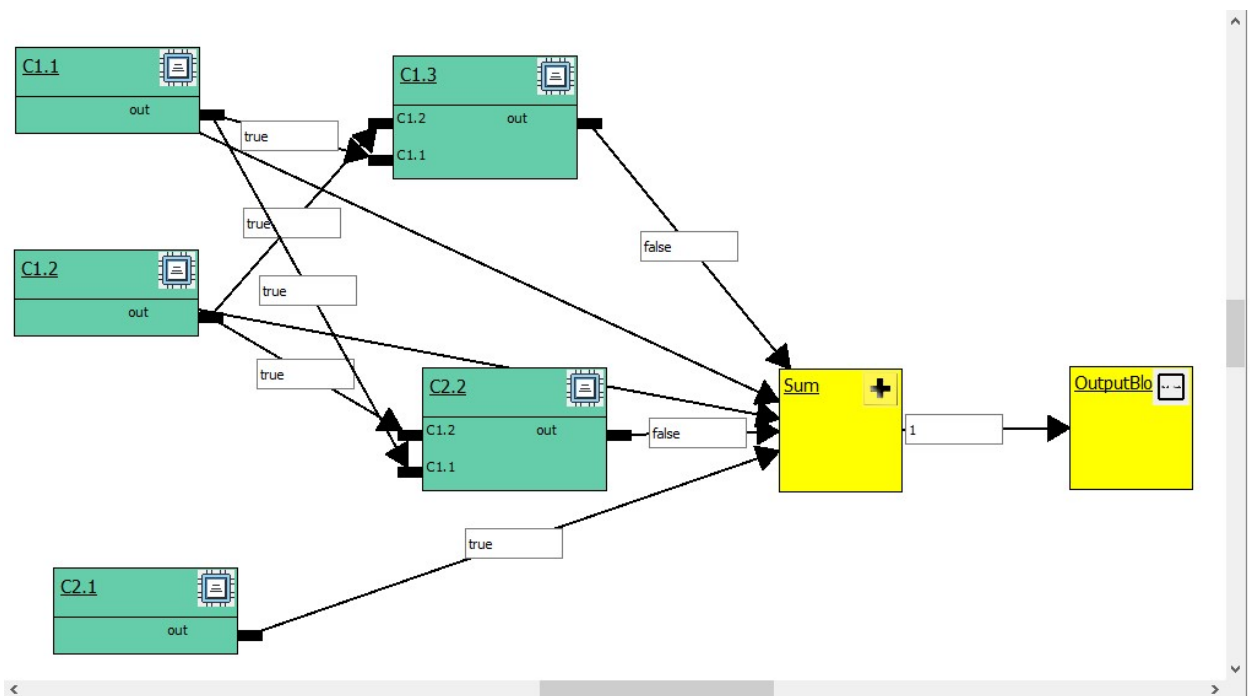
To run the resulting function block-based query diagram, start the server "Apache Jena Fuseki" (see point 5), then click on the "Run" button:



Once the program is running, the output window will show the result:



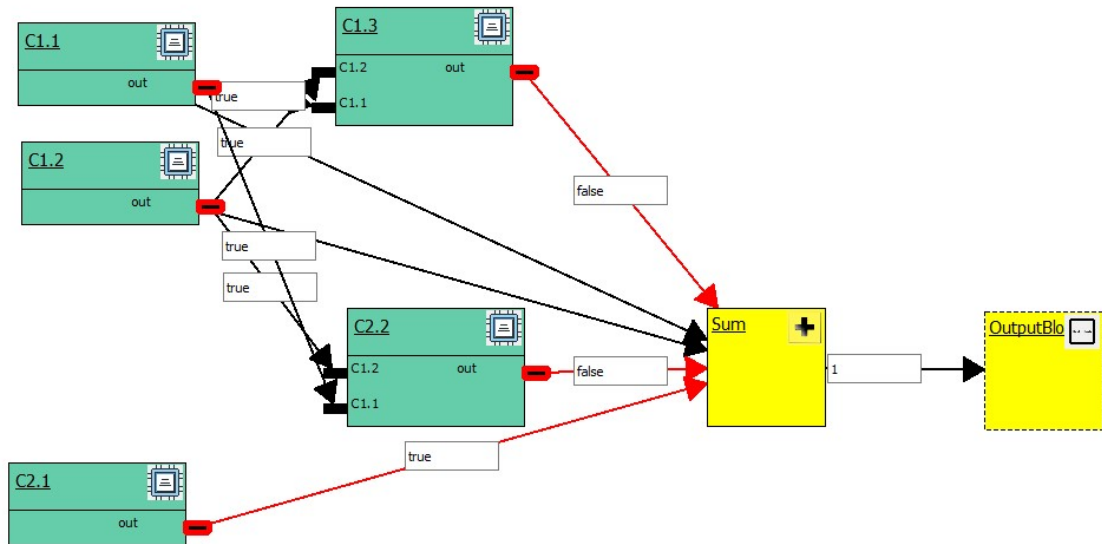
To understand why there is a one in the answer, you can go back to the project window and look at how the decision explanation method works.



As you can see, three values have arrived at the sum block, two of which are false, i.e. 0 and one value is 1. It follows that the result of this diagram is true.

It is also possible to look interactively at the block outputs that led to the current value.

This example after pointing at the amount block.



From the picture you can see that the sum block will work with the output values of the blocks: C1.3, C2.2, C2.1.

In the process of developing this test case, the following web page is used:

<https://zenodo.org/record/5010644#.Yg96EuhBwuU>