Nuero4j Flows – Developer Guide.

NEURO4J.ORG

PDF version available online at http://static.neuro4j.org/download/doc/workflow/UserGuide.pdf

ABOUT THIS GUIDE

This guide describes base concepts of Neuro4j Flows.

Online html version available at http://neuro4j.org/docs/wf/concepts

CONTENTS

CONCEPTS

Neuro4j Flow makes the bridge between software architects and developers. It allows you to model your business process by describing the steps that need to be executed using a flow chart.

Neuro4j Flows distinguishes four different programming layers:

- Flow Layer;
- User Blocks Layer;
- Business Object Layer;
- Presentation Layer (can be used in web-based applications);

Flow Layer

Flow displays particular business process, such as the processing of login information.

User Blocks Layer

User Block is small, re-usable unit or building block that perform the specific business logic in Flow, such as saving object into database.

Business Object Layer

Using Business Objects allows to separate business data and logic. Here is persistence layer e.g. Hibernate, Spring DAO, etc

Presentation Layer (optional)

Presentation Layer can be used for output data rendering. E.g.

HTML pages in web applications (rendered by JSTL, Velocity, etc)

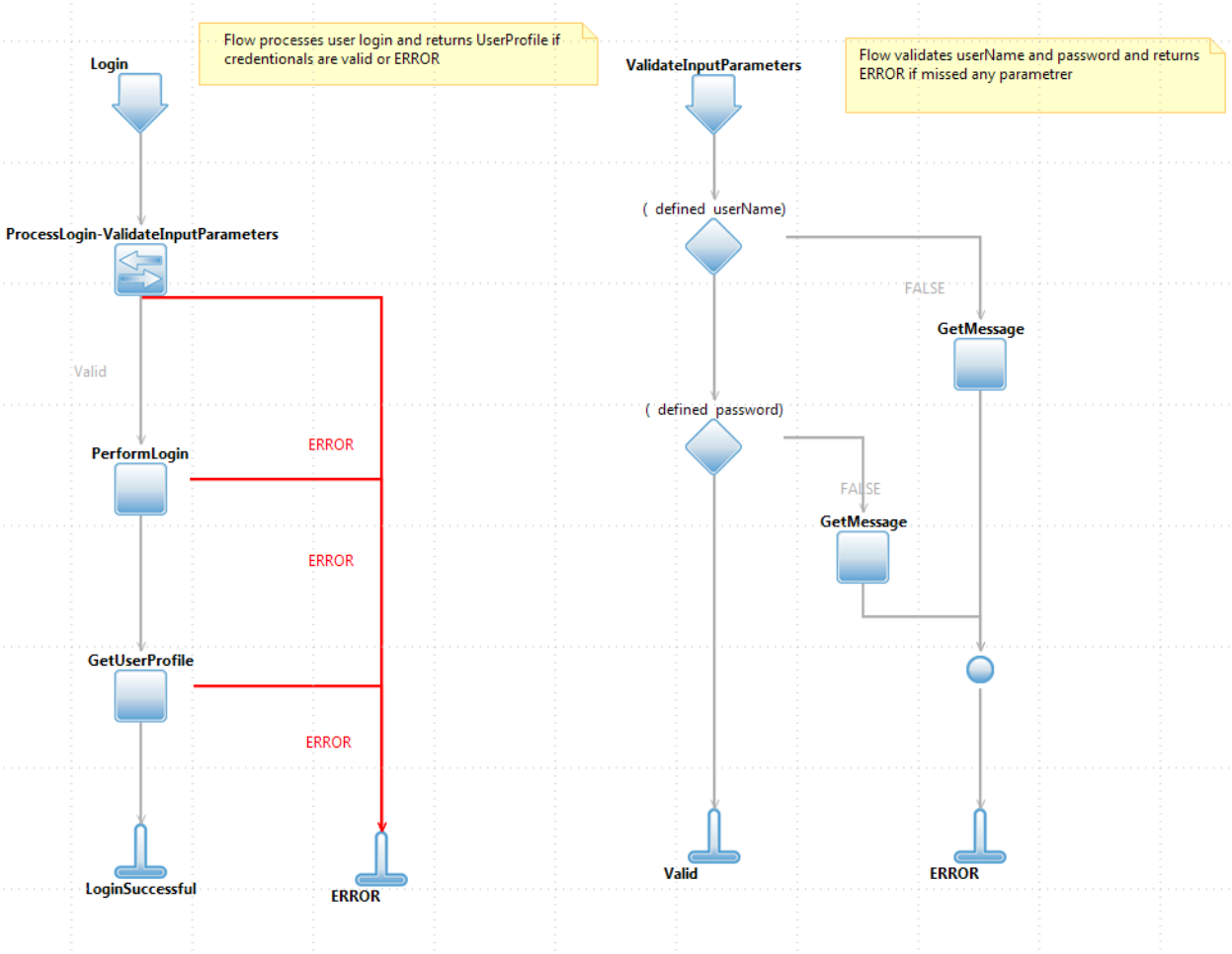
Rendering layer in report generation. E.g. Flows generate complex analitic report and use Jasper Reports for output rendering to PDF, HTML, etc..

What is execution context?

Execution context allows to exchange data between flow elements and between several flows. The Execution Context is a dynamic list of key-value pairs of data.

WORKFLOW FILE

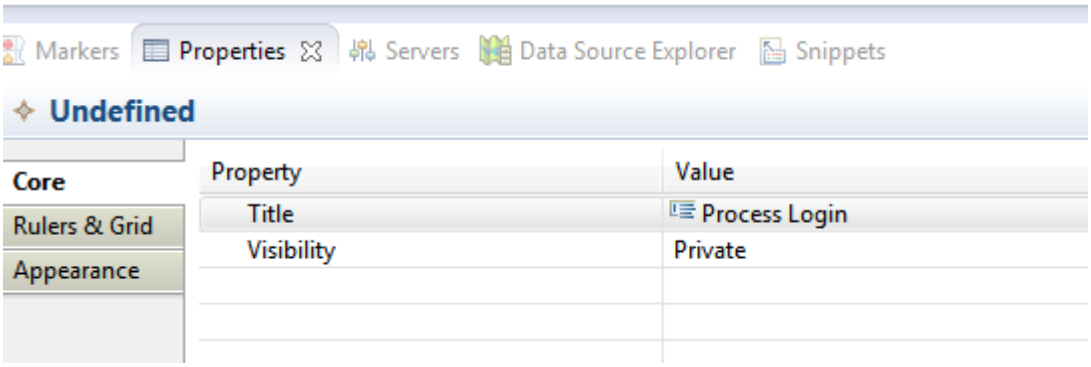
Workflow file holds one or several flows.



Workflow has following properties:

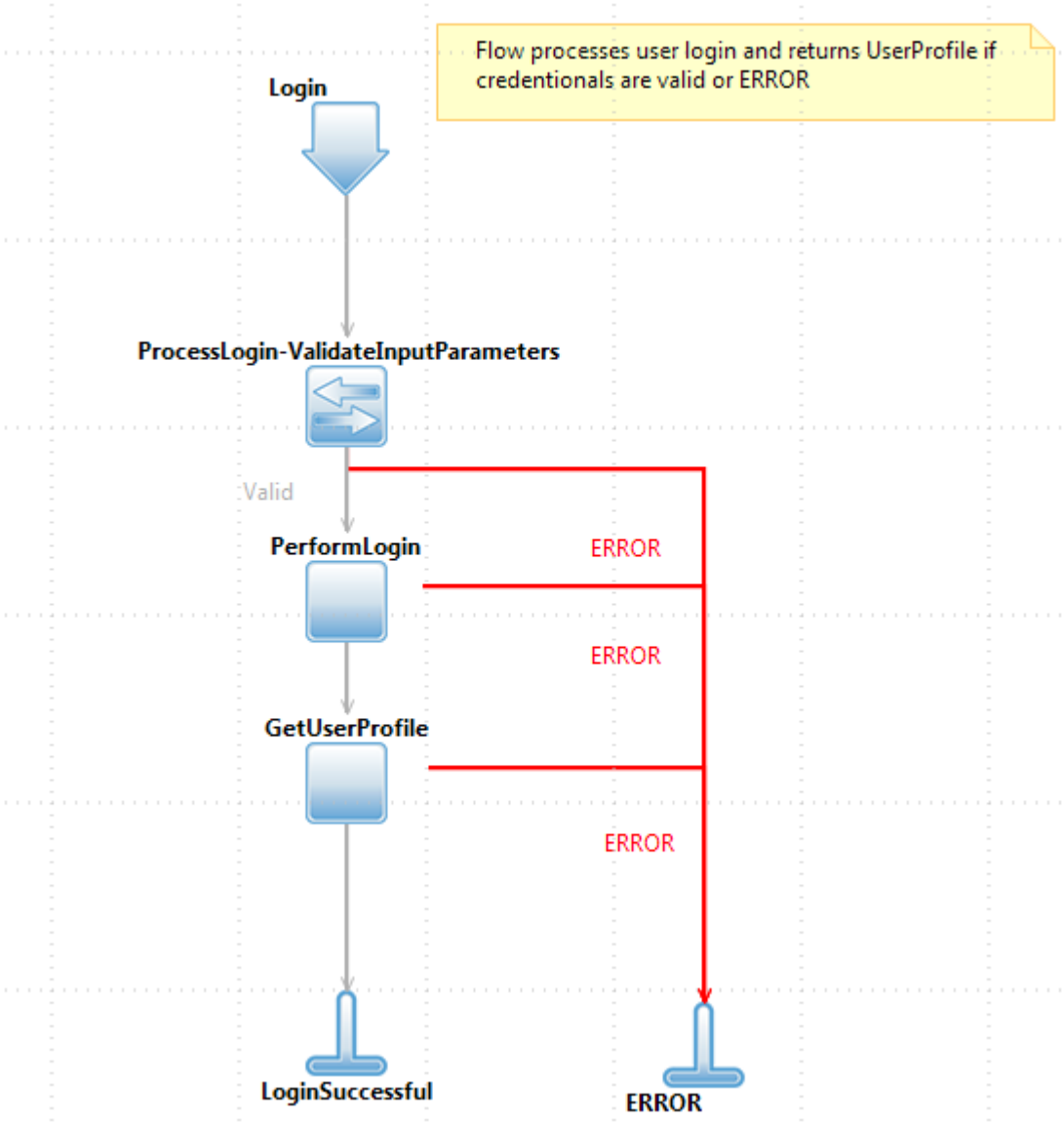
Property name	Description
Title	The name of workflow
Visibility	Workflow visibility. Can be *Public* or *Private*

If workflow is **Public** all flows from this file can be executed directly by Processor. If workflow is **Private** it can be executed just via CallNode.



FLOWS

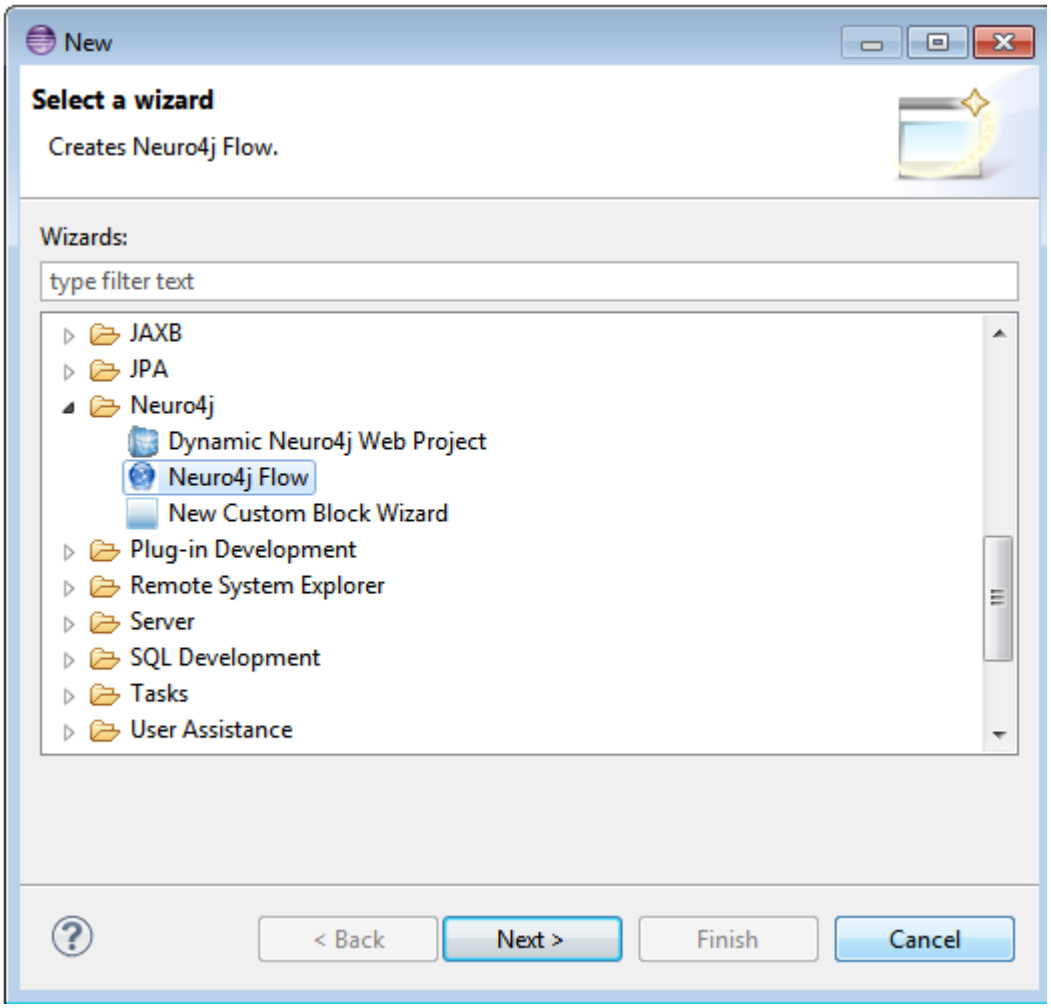
Flow is a set of connected nodes which represents some business logic.



Example of Flow which processes user login.

CREATING NEW FLOW

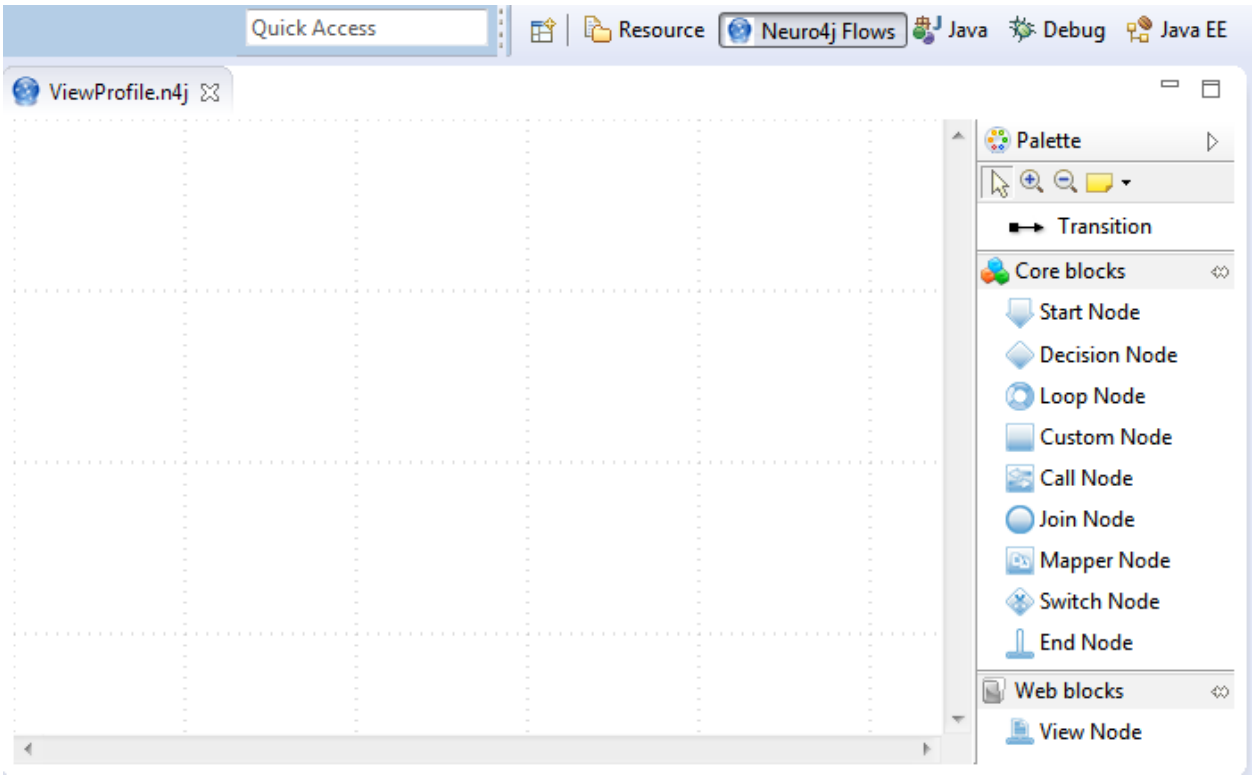
1) Create and select java package;
2) Right-Click on package - > New -> Other-> Neuro4j-> Neuro4j Flow;



3) Click “Next”;

4) Update Flow name and click “Finish”

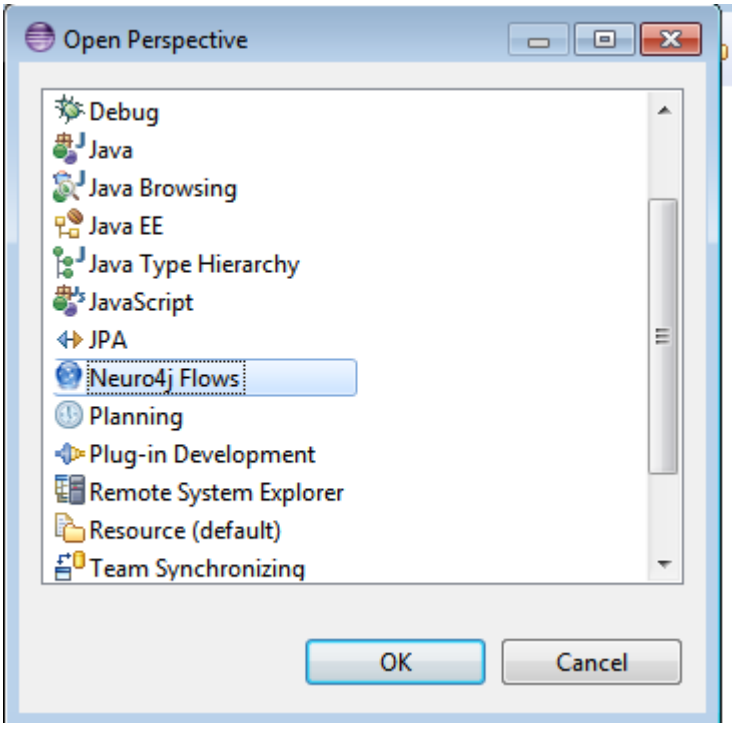
5) New file will be created;



NEURO4J FLOWS PERSPECTIVE

To open "Neuro4j Flows" perspective:

1) Select Window->Open Perspective->Other;
2) Select "Neuro4j Flows" and click Ok;

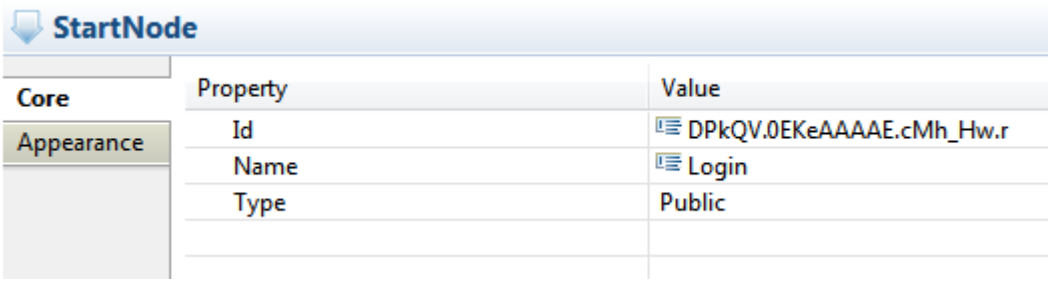


Flow can use following nodes:

START NODE

Start Node represents entry point into flow.

Property name	Description
id	Unique node id
name	Node display name; To run flow developer should specify flow name and start node name ex. org.neuro4j.example.ViewFlow-StartNode
type	Node type can be: Public or Private; Public – Node can be called by CallNode from other package  Private – Node can be called by CallNode just from Workflow where node defined; 

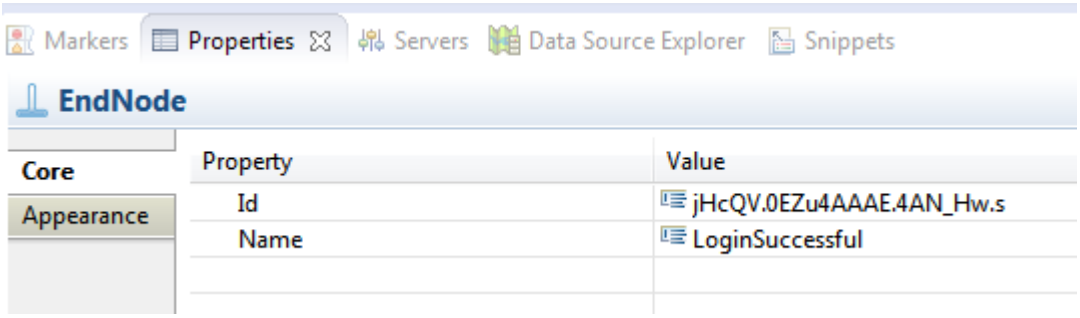


END NODE

End Node determines end of flow execution and pass execution to called flow if this flow was called by CallNode.

Property name	Description
id	Unique node id

name	Node display name;



LOOP NODE

Loop Node will be used when developer needs to execute a block of code several number of times.

Property name	Description
id	Unique node id
name	Node display name;
Iterator Key	Holds path to object which implement interface java.util.*Iterator, java.util.Collection or Array* in execution context. Please refer to BeanUtils regarding syntax.
Element Key	Holds element's name in context after each iteration.

EXAMPLES OF USAGE

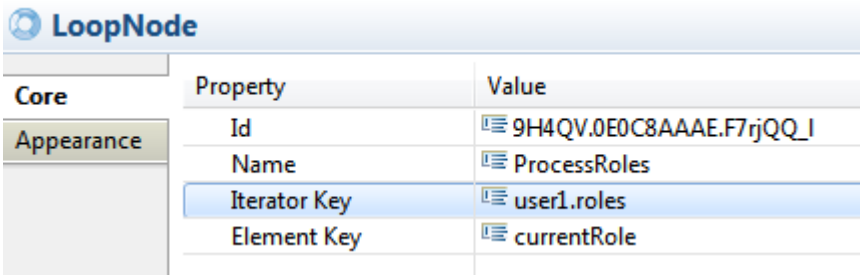
1)

Assume we have User object in context under name "user1" which holds list of Role objects

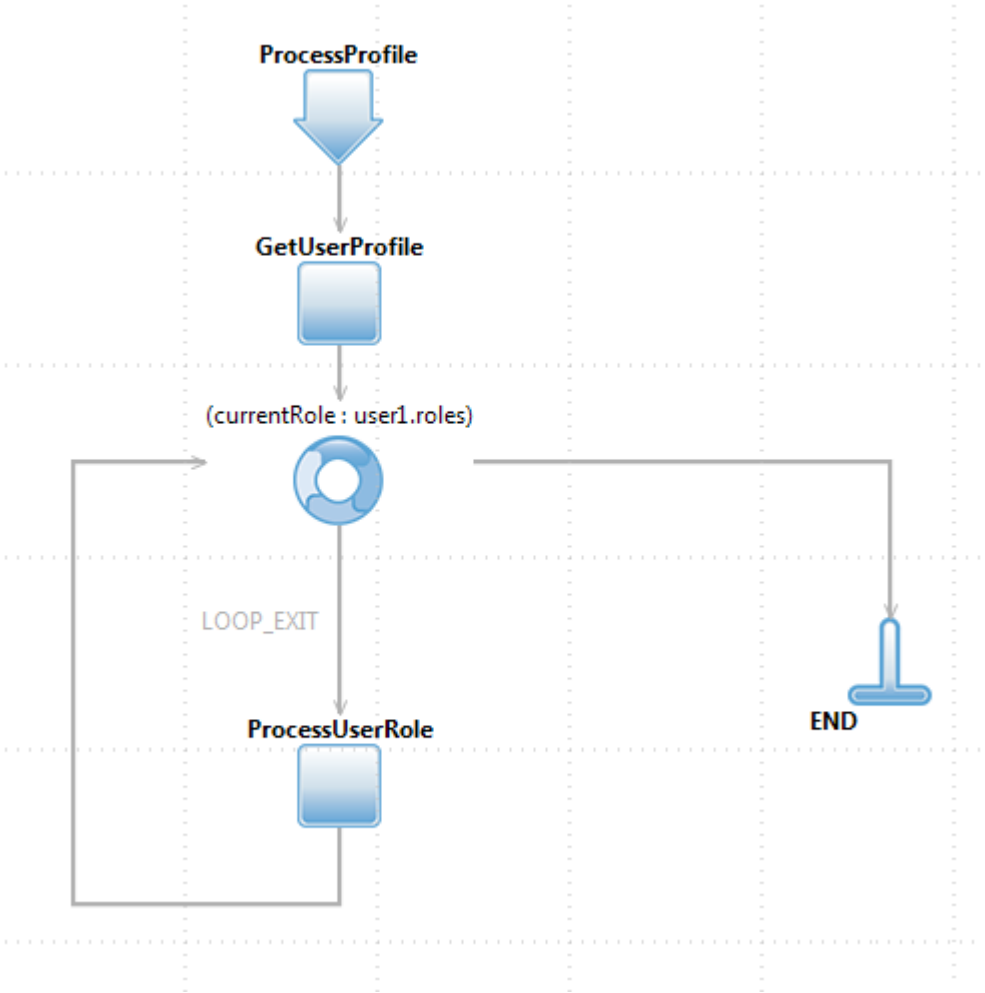
```
public class User {

	public User(){}
```

```
List<Role> roles;

public List<Role> getRoles() {
	return roles;

}
```

Path to “roles” will be - *user1.roles*



After each iteration current role will be available in execution context under name *currentRole*



DECISION NODE

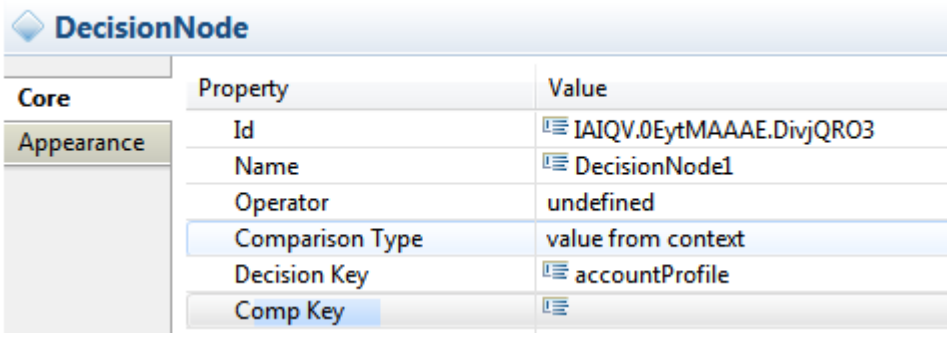
A decision node compares two values in the execution context or checks for a value in the context. Decision node has 2 exits – true and false.

Property name	Description
id	Unique node id
name	Node display name.
Operator	Operator can be one of following values: *Defined* Checks existence of an item in context. *Undefined* Checks if variable is not exist in context or equals null. *= (string)* String Equals. Checks if String in Decision Key equals to String in Comp Key. *!= (string)* String Not equal. Checks if String in Decision Key NOT equals to String in Comp Key. *empty string* Checks if String in Decision Key is equal to "". == Numeric Equals. Compare numbers for equality. *!=* Numeric not-equal < Numeric "Less than". > Numeric "Greater than". *has elements* Checks whether an iterable object has elements.
Comparison Type	Can be: - Constant value; - Value from context;
Decision Key	Name of object in context;

Comp Key	Value in this field will be compared with value from Decision Key

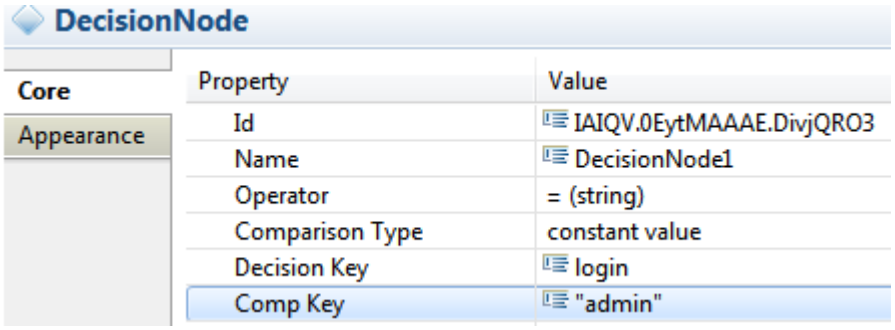
EXAMPLES OF USAGE

1)



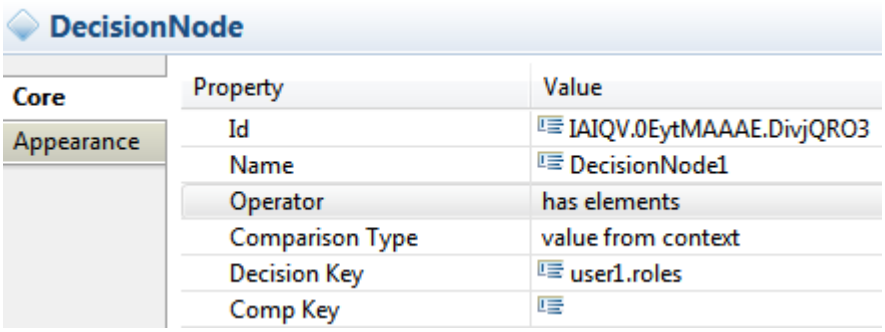
Checks if variable accountProfile does not exist in context or equals NULL.

2)



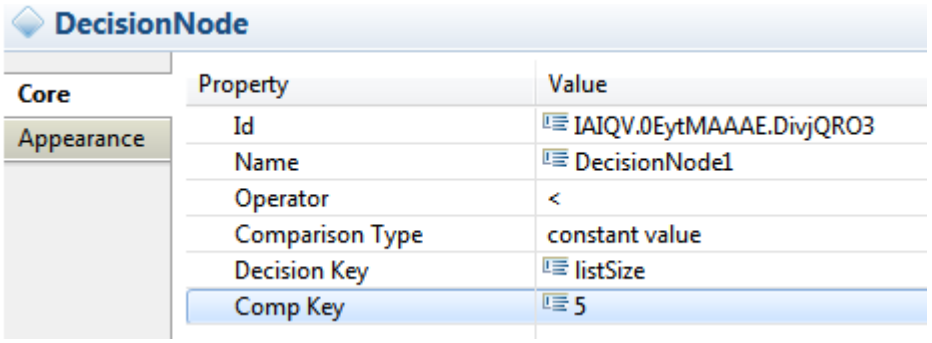
Checks if variable login from context equals “admin”

3)



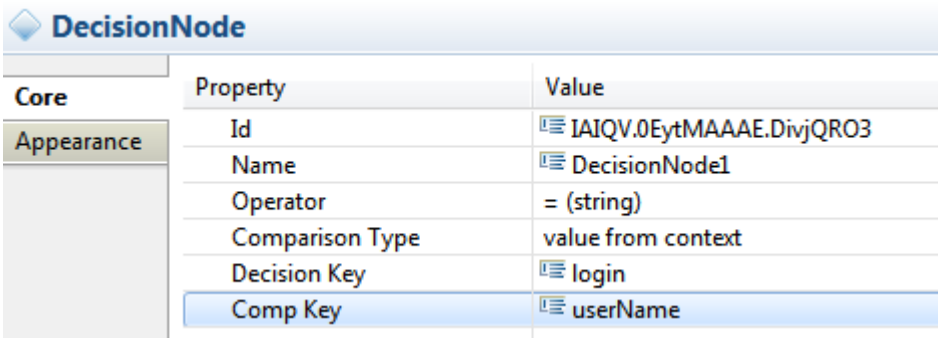
Checks if user has assigned roles.

4)



Checks if listSize less than 5

5)



Checks if variable login equals username.

JOIN NODE

Join Node used to join several transitions into one.

Property name	Description
id	Unique node id

name	Node display name;

CALL NODE

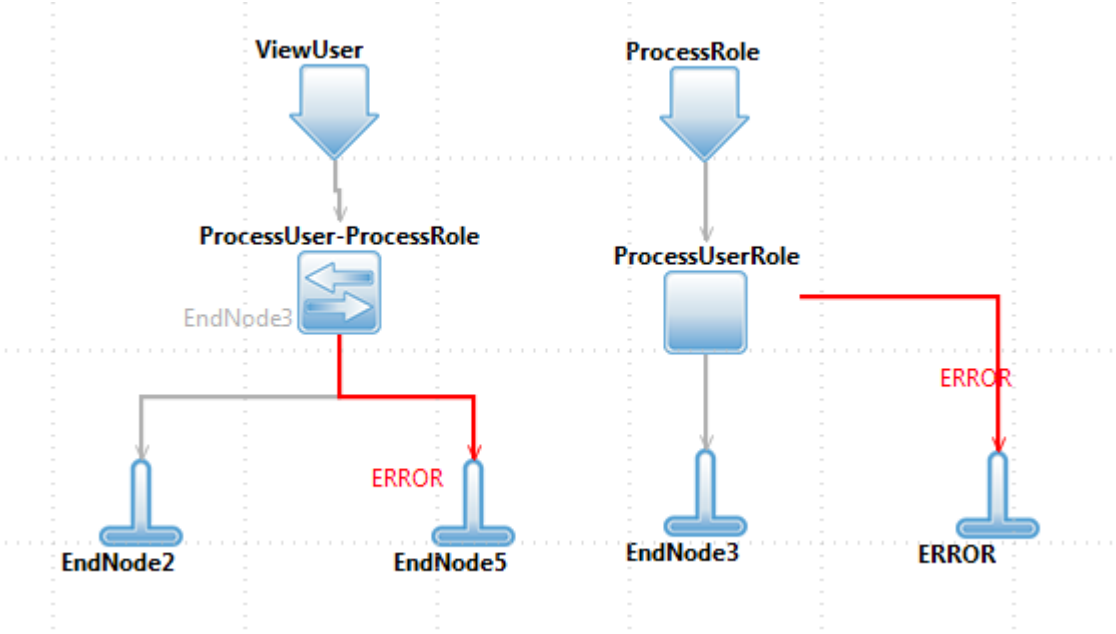
User can call another flow via Call Node . Processor will put context of original flow into called flow. Control returns to the next node in the original flow after the called flow finishes.

Property name	Description
id	Unique node id
name	Node display name;
Flow name	Static name of flow which will be executed; ex. org.neuro4j.example.ViewAccounts-List where “org.neuro4j.example.ViewAccounts” is flow name and “List” is Start Node name;
Dynamic Flow Name	Name of variable in execution context which holds name of flow which should be executed;

*CallNode can execute **public** and **private** flows from current file and just public flows from workflow which defined in other packages.*

CallNode will use name of EndNode from called flow as name of relation which will conect CallNode with another Node;

EXAMPLES OF USAGE



Example above shows how CallNode execute another flow “ProcessRole” in the same flow ProcessUser.

If user block finished with error (ERROR EndNode) CallNode will use transition to EndNode5.

Properties of CallNode will be:

CallNode

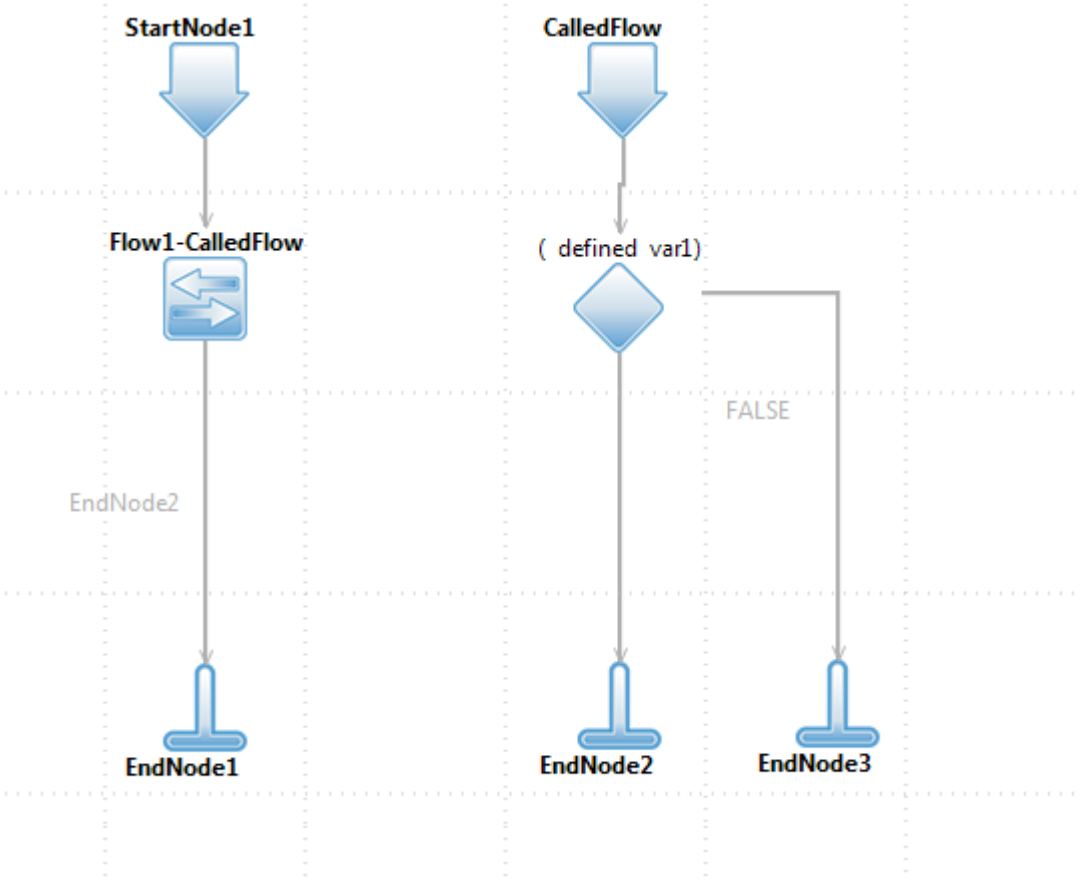
Core | Appearance

Property	Value
Id	QBIQV.0ECeAAAAE.Xd8XaRSH
Name	CallNode1
Flow Name	org.neuro4j.tutorial.flows.ProcessUser-ProcessRole
Dynamic Flow Name	

If CallNode has just one transition but called flow can return > 1 - processor will use this single transition.

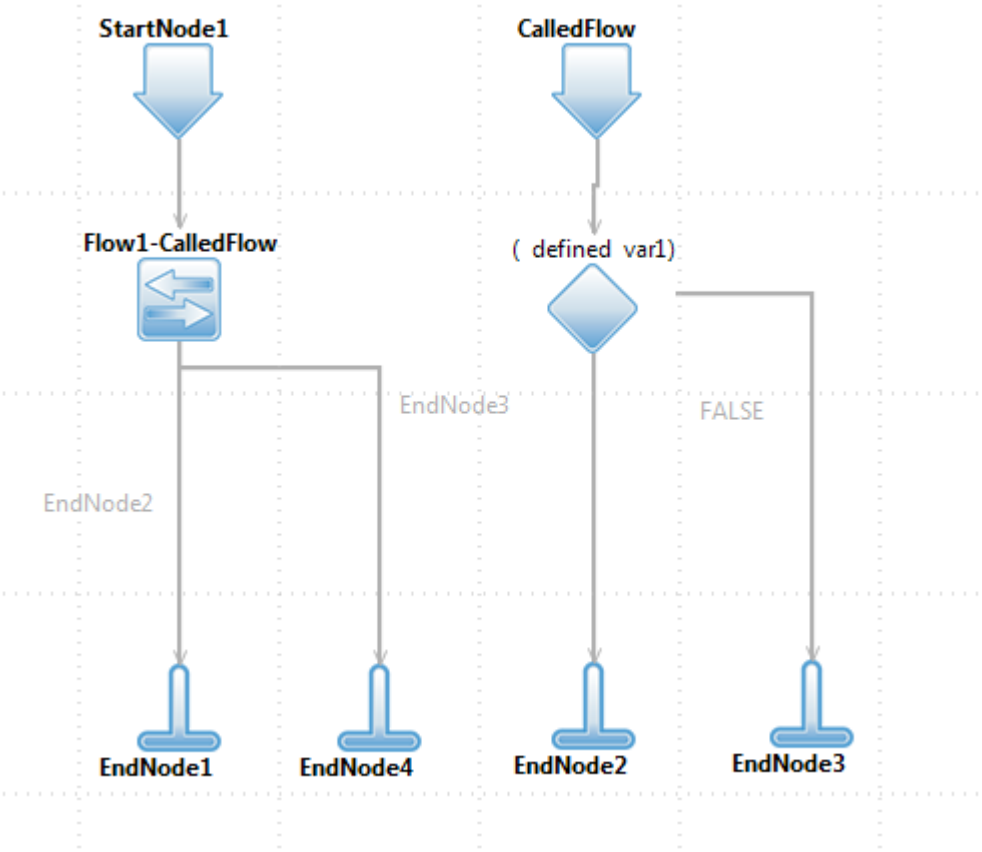
Example: CalledFlow has 2 EndNode's: EndNode2 and EndNode3

But developer has been defined just on transition for "EndNode2" during call "Flow1-CalledFlow".



In this case Processor will use this single transition even CalledFlow returns EndNode3.

On following example developer has been defined own transition for each EndNode.



SWITCH NODE

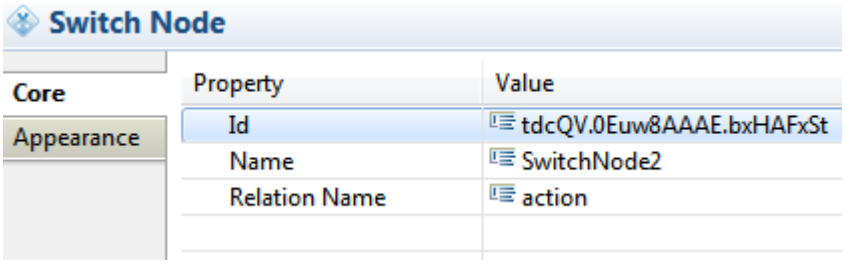
Switch node determines which relation will be used next by Processor;

Property name	Description
id	Unique node id
name	Node display name;
Relation Name	Name of relation which will be used next; The name can be static if it has been provided with " " or dynamic. If name is dynamic Processor will use value of variable in context. If there is not such relation it will use relation with name "DEFAULT"

EXAMPLE OF USAGE

1)

Assume we have SwitchNode with following properties:

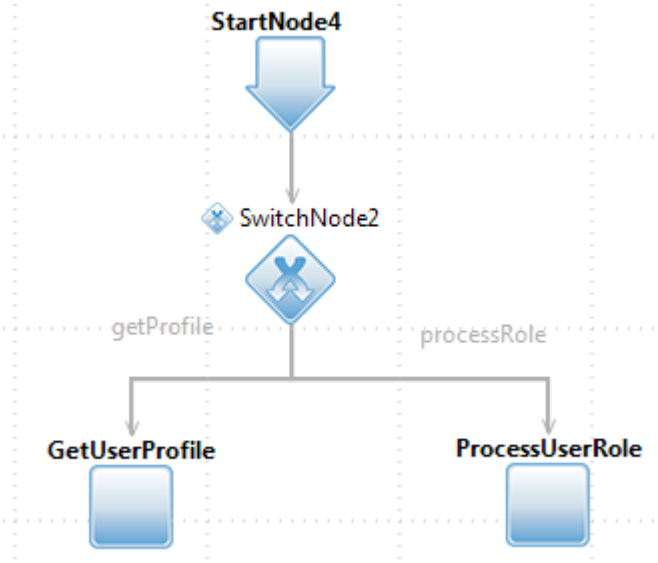


Processor will use variable action from context to determine next step:

If action == “getProfile” it will execute GetUserProfile block;

If action==”processUser” it will execute ProcessUserRole block;

If next step is unknown, Processor will try to execute transition with name “DEFAULT”.



MAPPER NODE

Mapper Node puts variable into context below provided name.

Property name	Description
id	Unique node id
name	Node display name;
Source	Path to variable in context. If it uses “ ” it will create string with such value; If developer uses “(class_name)” it will create object with given class. Ex.

	"(java.util.HashSet)" – will create new HashSet object and put it to context.
Target	The name below which variable will be available in context

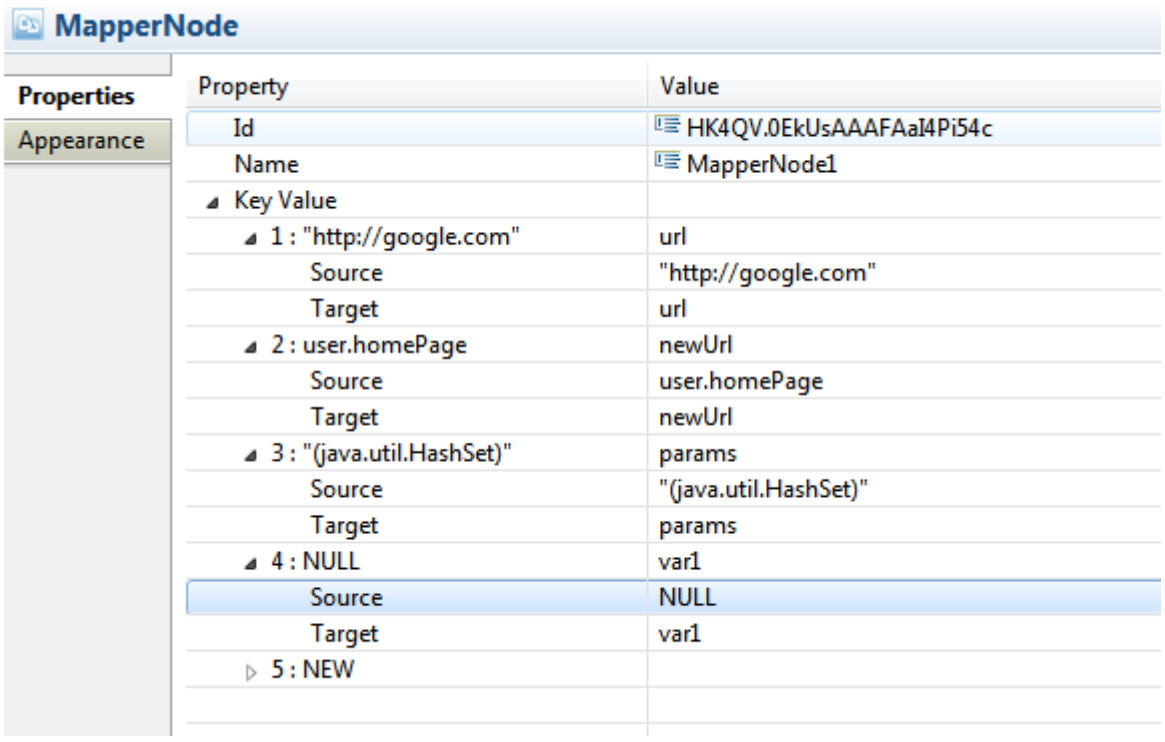
1)

In first case new variable will be created in context with name url and value "http://google.com"

In second case will be created new variable with name "newUrl" and value will be assigned from object "user" field "homePage"

In third case will be created HashSet with name "params".

Last example will reset value for var1 to *null.*



CUSTOM NODE

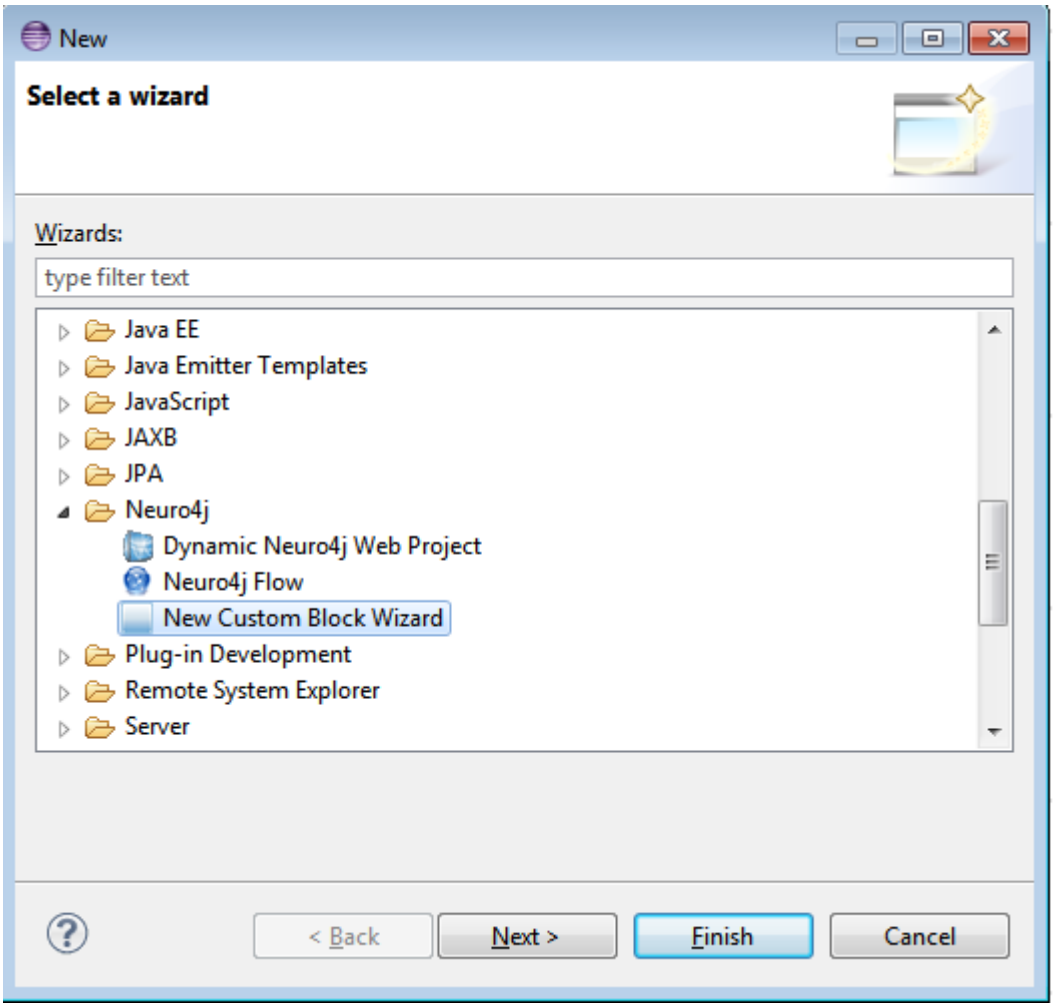

Custom Node call user defined java class. Custom Block is small, re-usable unit or building block that perform the business logic in Flow. This Block extends ***org.neuro4j.workflow.node.CustomBlock*** and provides implementation of ***execute*** *method.*

Property name	Description
id	Unique node id
name	Node display name;
Class Name	Name of java class which extends ***org.neuro4j.workflow.node.CustomBlock.***
Input parameters	Holds list of input parameters; All input parameter are defined like annotations in class.
Output parameters	Holds list of output parameters. All output parameters are definded like annotations in class.

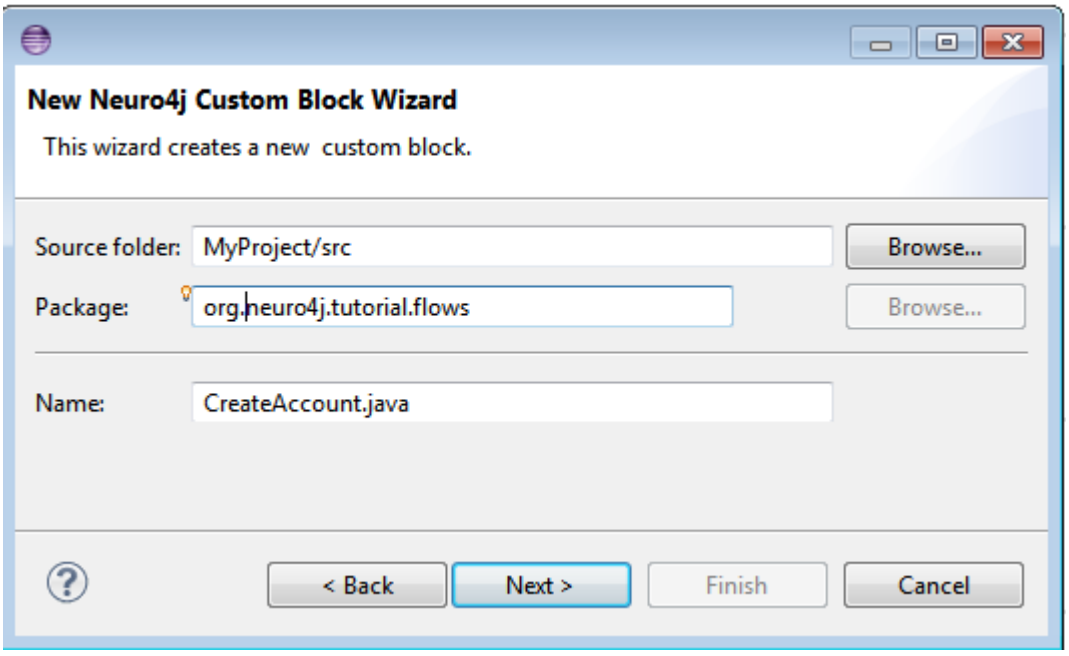
CREATING NEW CUSTOM BLOCK

Custom Block Wizard allows you to add new custom blocks to your project. Wizard collects the required information like name, input and output parameters and generates code.

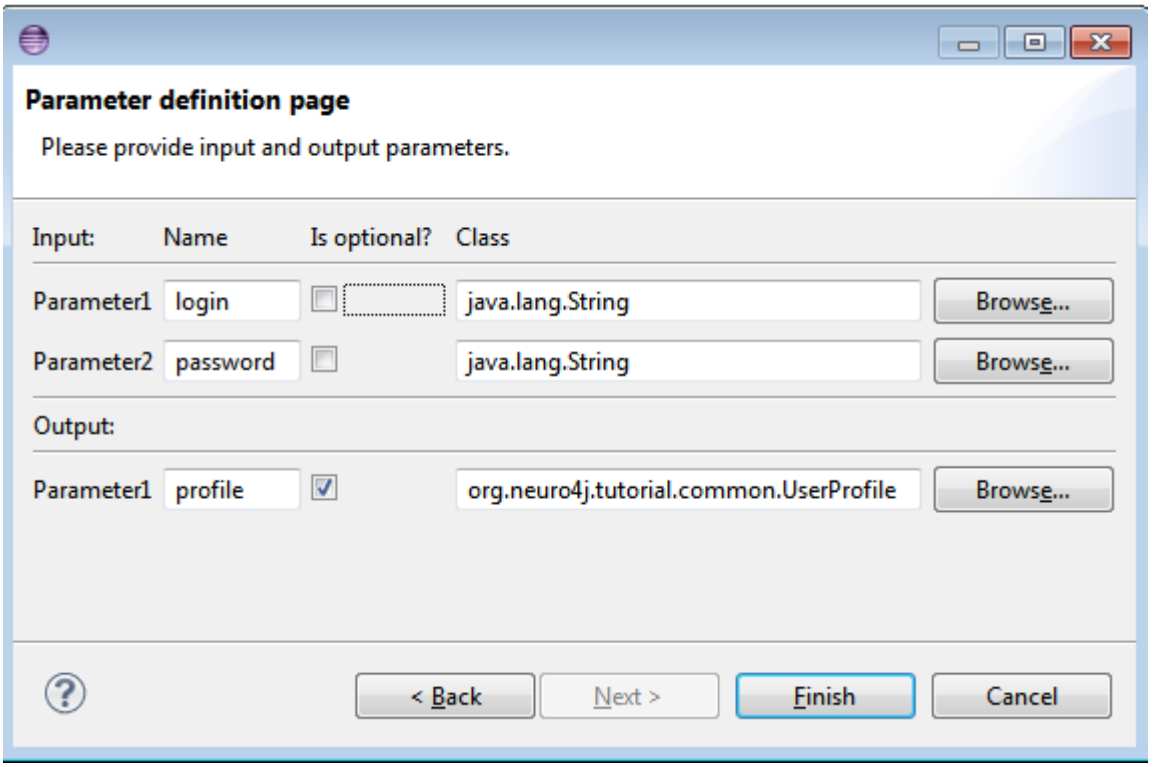
1) In Package Explorer, select java package -> right-click - > New
2) New ->Other->Neuro4j->New Custom Block Wizard->Next



3) Provide class name and click “Next”



4) Provide input and output parameters and click “Finish”

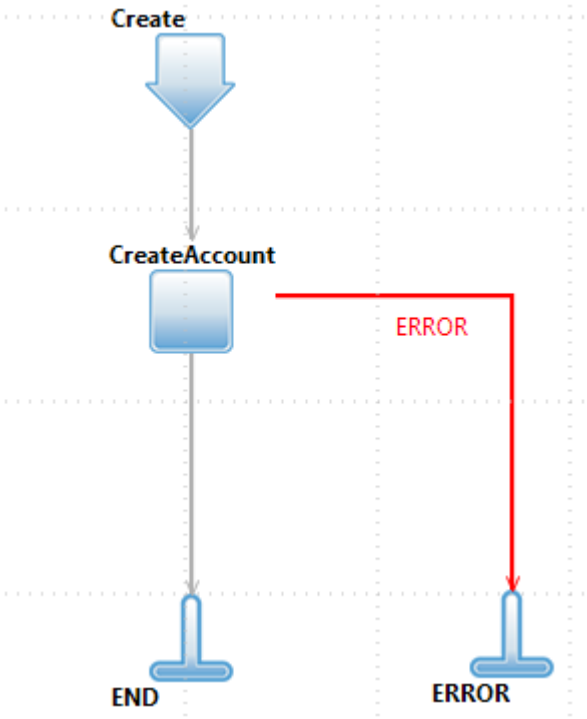


Wizard will create new java class. Developer must provide implementation for ***execute*** method.

```
public int execute(FlowContext ctx) throws FlowExecutionException {

		String login = (String) ctx.get(IN_LOGIN);
		String password = (String)  ctx.get(IN_PASSWORD);


		Account account = null;
		try {
			account = accountMng.createAccount(login, password);
		} catch (CreateException e) {
			e.printStackTrace();

			ctx.put("errorMessage", e.getMessage());
			return ERROR;
		}

		// created object puts to context
		ctx.put(OUT_ACCOUNT, account);


		return NEXT;

	}
```

Method ***execute*** returns "NEXT" if execution finished in normal way otherwice return "ERROR".

Depending on which result returned block, Processor will use “NEXT” transition or “ERROR”



All managers/services should be initialized in method **init()**

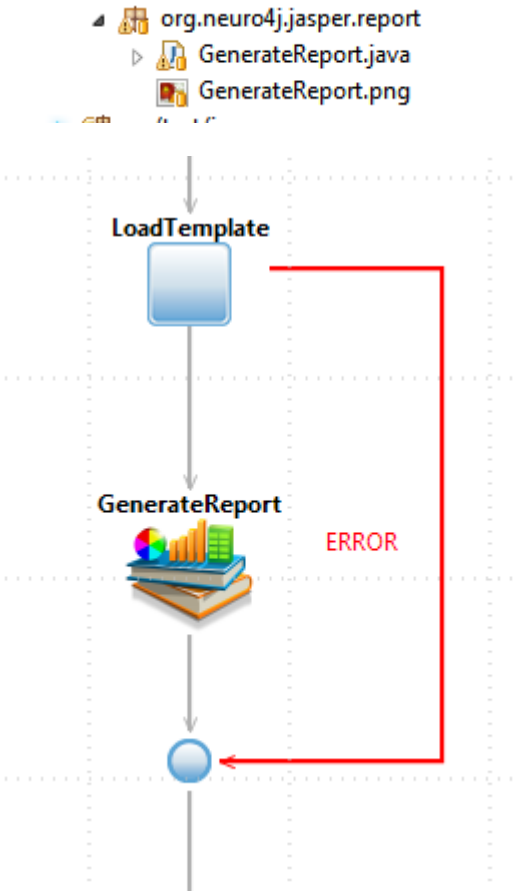
```
    private AccountMng accountMng = null;

.     .     .     .     .     .     .     .     .     .

       @Override
       public void init() throws FlowInitializationException{
              super.init();
              accountMng = AccountMngImpl.getInstance();
       }
```

Developer should not use fields of custom block to save states of objects.

```
public class CreateAccount extends CustomBlock {

       Account account = null; // wrong usage!

.     .     .     .     .     .     .     .
```

CUSTOM ICON

Developer can define own icon for Custom Block. Icon should be located in the same package with class and has the same name in png format. For example GenerateReport block:



VIEW NODE

View Node can be used in web application to define name of jsp file which will be used to display web content

Property name	Description
id	Unique node id
name	Node display name;
renderType	Which render will process view node. Supported types: *jsp* | *velocity* | *jasper.* Default render: jsp.
Resource Name	Path to file
Dynamic View Name	Name of variable in context which holds name of jsp file;

VIEWNODE RENDERS

Currently web-processor supports following types: jsp, velocity, jasper, json.

name	Description	Project	File extension
jsp	default render	*neuro4j-logic-web*	*.jsp*
velocity	Allows render Apache Velocity templates	*velocity-web-plugin*	*.vm*
jasper	Allows generate reports using JasperReports library.	*jasper-report-web-plugin*	*.jasper*
json	Allows to generate response in json format.	*json-web-plugin*	-