

---

**FrameSolutions**

**Knowledge Editor User  
Manual**

---

**Author:** Wei Li

**Version:** 5.8.0

# TABLE OF CONTENTS

<b>1</b>	<b>Knowledge Editor Preface .....</b>	<b>4</b>
1.1	Intended audience .....	4
1.2	Java versus .Net version.....	4
1.3	Content of this user manual .....	4
<b>2</b>	<b>Installing Knowledge Editor .....</b>	<b>5</b>
<b>3</b>	<b>Working with Projects .....</b>	<b>6</b>
3.1	Why use projects?.....	6
3.2	Start a new project .....	6
<b>4</b>	<b>Working with Process Bases .....</b>	<b>8</b>
4.1	Process Base View .....	8
4.1.1	Attach and detach.....	8
4.1.2	Defining a new task .....	9
4.1.3	Defining a new step .....	9
4.1.4	Defining conditions .....	10
4.1.5	Defining actions.....	12
<b>5</b>	<b>Working with Task Libraries .....</b>	<b>14</b>
5.1	Process libraries view .....	14
5.2	Editing process library entries .....	15
5.3	Creating process library entries directly from tasks .....	16
<b>6</b>	<b>Working with the Rule Editor .....</b>	<b>18</b>
6.1	Rule Base View .....	18
6.1.1	Defining a new rule.....	18

<b>7</b>	<b>Version Control of definition files .....</b>	<b>20</b>
<b>8</b>	<b>Using the Knowledge Editor for Rapid Prototyping ..</b>	<b>21</b>
8.1	Introduction .....	21
8.2	Preparations and planning the prototype .....	21
8.3	Making the prototype .....	23
8.4	Brief review of prototyping functionality .....	24
8.4.1	Prototype ribbon menu (1) .....	25
8.4.2	Prototype image list (2) .....	26
8.4.3	Prototype image area (3) .....	26
8.4.4	Prototype linking area (4) .....	26
8.4.5	Saving the prototype .....	26
<b>9</b>	<b>Connecting Custom Code .....</b>	<b>27</b>
9.1	Workflow .....	27
9.2	Process .....	27
<b>10</b>	<b>Generating process base documentation .....</b>	<b>28</b>
10.1	Documentation of process bases .....	28
10.2	Overview of the generated documentation .....	28
10.2.1	The ribbon menu to generate, view and export documentation: .....	32
<b>11</b>	<b>Appendix - Object Expressions in the Knowledge Editor .....</b>	<b>34</b>
11.1	Overview .....	34
11.2	BNF .....	35



# 1 KNOWLEDGE EDITOR

## PREFACE

The FrameSolutions Knowledge Editor User Manual describes how to use the knowledge editor in a typical FrameSolutions project.

This document is compatible with FrameSolutions 5.8.0

### 1.1 Intended audience

The intended reader is a business analyst or a programmer that models work processes in a FrameSolutions application. Anyone editing task definitions or rules bases will need to use the Knowledge Editor. The documents "Introduction to FrameSolutions" and "Business Analyst's Guide" are prerequisites for reading this manual.

### 1.2 Java versus .Net version

FrameSolutions consists of two different distinct versions: one implemented in Java and one implemented in .Net C#. The core components for process execution, workflow and declarative rule execution are functionally equivalent in the two versions. The process and rule definition files created in the Knowledge Editor are interchangeable. The difference in the technology versions are mainly in the number of additional supporting modules which is higher in the Java version than in the .Net version.

### 1.3 Content of this user manual

FrameSolutions is a framework and there is an unlimited number of possibilities for how you can assemble applications on top of the framework. To provide a usable and practical user manual we have built the manual based on some assumptions. We assume the reader wants to start a new FrameSolutions application, creating Task definitions, adding rules and finally connecting the activities in tasks to domain objects. This manual is not a reference manual with description of each and single element in the editor.

## 2      INSTALLING KNOWLEDGE EDITOR

The Knowledge Editor is a stand-alone Windows application. All prerequisites are included in one .msi file. The installation is a simple double click on the [KnowledgeEditorInstaller.msi](#).

## 3 WORKING WITH PROJECTS

The Knowledge Editor is a stand-alone Windows application to edit the project, process bases, process library and rule bases in FrameSolutions. There is one or more XML files for each of these concepts. The editor is built around the concept of FrameSolutions with a project view showing all the entries belonging to the project.

### 3.1 Why use projects?

A FrameSolutions project in the Knowledge Editor is nothing more than a collection of process definitions, process libraries and rule bases. A good convention is to create one and only one FrameSolutions project for each FrameSolutions application. It is technically possible to have more than one FrameSolutions project for one application, but in this scenario we will only work with one FrameSolutions project. Using one FrameSolutions project gives you the following benefits:

- Quick overview of all process bases, libraries and rule bases used in your application.
- Open all XML files in one operation, when you open the project you load all project related files.
- Transparent handling of standard elements that facilitates reuse.
- Search in all project related files.
- See dependencies between process bases.
- Set up a global execution context to bind domain specific code into your declarations.

### 3.2 Start a new project

To start a new project you need to go to the "Knowledge Editor" ribbon and click the "New FS Project" in the upper left corner. Then go on to choose the "FS Project" tab to configure the project. In the project tab you should register the following:

- Name: this is the name of your project, the name is not used in runtime, but it is very wise to use the name of your application.
- Filename: filename of the project xml file.
- ProcessBases Path: path where all the process bases with your Task definitions in the project is located.
- Process Library: the Task definition library file. You need to create a new Task Library and save the file before you can register it on a project.
- Rule Base Path: path where all the rule base files in the project is located.

- Codes Path: path where all the codes xml files in the project is located
- Code Relations Path: path where all the code relations xml files in the project is located
- Export Path: Reserved for future use.
- XML format: Choose the "Standard" format for all new projects.
- Encoding: Use "UTF8" for all new projects.
- Technology: Choose FSJava if the target application is in Java or FSNet for a .Net application.
- Line Ending: End of Line type means how the editor saves the definition files. You can choose between Windows, Unix or Mac type. This will effect the use of CR/LF in the end of each line in the file. Note that if you change the line ending in the project the process, library and rule base files will be saved with the correct line ending next time you save them.
- Help Base URI: If all help texts in the project are located at the same URI, you can set the base URI project wide.
- Global Execution Context: Here we list the variables that are "hooks" into the domain specific model. Read more about connecting the declarations with the domain model in the chapter "Connecting Custom Code". At this point it is enough to note that all variables used in object expressions should be declared here. There is a number of predefined parameters in FrameSolutions that you do not need to declare.



## 4 WORKING WITH PROCESS BASES

The real work in a FrameSolutions application is done by execution tasks and the step (also called activities) within. The template tasks are defined as Process definitions and stored in Process Bases. One project may have one or more process bases. In an application with many processes, we strongly recommend to use one process base for each main functional area. A FrameSolutions application loads all the process bases so the split into different process bases has no significance in runtime. It is only a convenience when working in the Knowledge Editor and for avoiding concurrent update of the XML files when many analysts work on the same file.

Each process base can contain process definitions and standard elements. You can define standard elements for process definitions, activities, conditions and actions. Standard elements can be used by other elements in the process base they are defined and in any other process bases loaded in the editor. The purpose of standard elements is to facilitate reuse and standardization within a project. Reuse of standard elements work in the same way as object oriented inheritance. This means that you for instance can base a process definition on a standard process definition and only override the configuration specific for the current process definition.

The Knowledge Editor will validate the consistency of process and rule bases. This might lead to validation errors. A validation error will be shown as a red exclamation mark. To see the actual error message move the cursor over the exclamation mark.

### 4.1 Process Base View

The process view is opened from the project view by double clicking a process base.

To create a new process base click the "New Process Base" in the "FS Project" ribbon. You need to enter a unique name within the project and the filename for the xml definition file. In the process base view there is a "Dependencies" field showing process bases containing standard elements that this process base depends on. Initially this field is empty. It will be updated automatically when you start using standard elements from other process bases.

#### 4.1.1 Attach and detach

If you want to include a process base that is not created within your project you can use the "Attach" operation either from the "FS Project" ribbon or right click on the "Processbases" list in the "FSProject" view. Here you can include the process base in your project. The other way around you can detach a process base and thereby take the process base out of the project.

### 4.1.2 Defining a new task

A task definition (earlier called a process definition) is a sequence of steps (earlier called activities). The task definitions are instantiated as task instances at runtime. It is the task instances that can be passed around in a workflow. To create a new task definition click the "Process" button in the "ProcessBase" ribbon. The following attributes are displayed for a task definition:

- Inherited from: Use this drop down to base the new Task Definition on a standard definition.
- IsIndependentProcess: This field is deprecated.
- HelpTextValue: You can write help texts directly in the editor, but we recommend to write help text in separate .html files.
- Href: URI to help text in HTML format.
- DisplayName: Can be anything, but it is recommended to use the same as the Name attribute.
- Name: It is recommended to have unique names within a project.
- Id: Created automatically, must be unique within the project, read only.
- Context Variables: Here we define the context variables used in only this task definition. These are used in addition to the Global Execution context. Read more in the chapter "Connecting Custom Code".
- Annotation: All the annotation attributes are used for documentation and is not used in runtime.

When a task definition is based on standard task definition you will see the values that are inherited in the right hand side of the screen. This is just for convenience. You cannot change these without editing the standard task definition. Typically you will only change the name of the inherited definition and later on configure the steps in the standard task definition.

Attributes/Context		Conditions/Actions	Annotation	Prototype/Documentation
Inherit from:				
None				
AttributeName	Value			
HelpTextValue				
Href				
Mandatory	<input checked="" type="checkbox"/>			
MustFollow				
Repeatable	<input checked="" type="checkbox"/>			
DisplayName	Register candidate profile			
Name	RegisterCandidateProfile			
PrototypePath	\\.\prototype\RecruitmentProcessbase\RegisterCandidate\RegisterCandidateProfile\RegisterCandidateProfile.xml			
Id	CXaec80dbb-adae-46d8-944b-3ee22ee5cb14			
CreationDate	01.01.0001 00:00:00			

### 4.1.3 Defining a new step

A task definition contains a list of steps and sub tasks (sub tasks are also called subprocesses). Steps are called Activities in the editor. To create a new step click the "Activity" button in the "ProcessDefinition" ribbon. A step can inherit from a standard step. Select the standard step you

want the new step to inherit from in the drop down list labeled "Inherited from:". The following attributes are displayed for a step definition in the "Attributes/Context" tab:

- Inherited from: Use this drop down list to base the new step on a standard step.
- HelpTextView: You can write help texts directly in the editor, but we recommend to write help text in separate .html files.
- Href: URI to help text in HTML format.
- Mandatory: Check the check box if the step is mandatory. It is possible to specify that a step is conditionally mandatory. This is done in the "Conditions/Actions" tab.
- MustFollow: If it only should be possible to execute the step after one other step has completed, use the drop down list to select the step that has to be completed.
- Repeatable: Check the check box if the step is repeatable or in other words can be executed more than once. It is possible specify that a step is conditionally repeatable. This is done in the "Conditions/Actions" tab. Experience shows that setting steps to repeatable is a practical way to handle exceptions and error situations even if you initially think about the step to be something you do once.
- DisplayName: Text to be shown in the GUI.
- Name: Does not need to be unique
- Id: Created automatically, must be unique within the project, read only.
- CreationDate: Set automatically, read only.

If you want to make your new step into a standard step just select the step and click "Standard Activity" in the "ProcessBase" ribbon. The step will then appear under the "Standard Activities" tree view to the left of the screen and in the "Inherited from:" drop down list.

Attributes/Context		Conditions/Actions	Annotation	Prototype/Documentation
Inherit from:				
None				
AttributeName	Value			
HelpText\Value				
Href				
Mandatory	<input checked="" type="checkbox"/>			
MustFollow				
Repeatable	<input checked="" type="checkbox"/>			
DisplayName	Register candidate profile			
Name	RegisterCandidateProfile			
PrototypePath	\\.\prototype\RecruitmentProcessbase\RegisterCandidate\RegisterCandidateProfile\RegisterCandidateProfile.xml			
Id	CXaec80dbb-adae-46d8-944b-3ee22ee5cb14			
CreationDate	01.01.0001 00:00:00			

#### 4.1.4 Defining conditions

You can configure different conditions for a step. These are:

- When a step is mandatory

- When a step is repeatable
- When the step is included in a task instance. This is the "Include Condition". It means that if the condition is false the step will not be displayed for the task instance. The default value is true.
- When a step can be executed in a task instance. This is the "Pre condition". This is typically used when some other steps or events must have occurred. The default value is true.
- When the true actions or the false actions of a step should be executed. This is the "ActionChoice Condition". The default value is true.
- When the state is as expected after executing the other parts of the step. This is the "Post Condition". The default value is true. When the "Post Condition" is true the step is regarded as completed.

The example below shows the include condition for the step "Register candidate profile". This include condition specifies that the context variable Position must be null for the step to be included. In other words, if it is already registered a position, the "Register candidate profile" step is not included in the task instance.

The screenshot shows the Knowledge Editor interface with the 'Conditions/Actions' ribbon selected. The 'Register candidate profile' step is expanded, showing its conditions. The 'Include Condition' is selected, and its details are shown in the 'Condition Details' pane. The condition expression is '?Position == null'. The 'Condition Details' pane also shows a table with attributes and values.

AttributeName	Value
FalseFeedback	
TrueFeedback	
DisplayName	New Condition
Name	
Id	CX8aa4025f-62f0-49aa-a1c0-5f3be27c4f26

To configure a new condition click the appropriate condition name in the "Conditions" ribbon. A condition can be based on a standard condition in which case you choose from the "Inherit from:" drop down list. Conditions are logical expressions and can contain logical operations, object expressions or simply true or false. When editing conditions you will see them as a hierarchy. Click a node and edit the element in the pane that appears to the right. To enter a new And branch, a new Or branch or a new Not branch click the "Introduce node" drop down list and choose the desired option. Note that the editing is done in a hierarchy. The example below shows how to enter

the logical expression: ?Candidate.isPrequalified() OR (?Candidate.age > 70 AND ?Candidate.isProfessor())

The screenshot shows the 'Condition Details' panel in the Knowledge Editor. It includes a search bar, a dropdown for 'Inherit from' (set to 'None'), and a 'Condition Expression' field with a red error icon. Below this is a tree view showing a logical expression: 'or' containing '?Candidate.isPrequalified()' and 'and' containing '?Candidate.age > 70' and '?Candidate.isProfessor()'. To the right, the 'Expression:' field contains '?Candidate.isPrequalified()'. At the bottom, a table lists attributes for the condition.

AttributeName	Value
FalseFeedback	
TrueFeedback	Candidate is prequalified
DisplayName	Prequalified Condition
Name	PrequalifiedCondition
Id	CXec0b3492-d348-470b-a601-651fc20a8bb3

The following attributes are displayed for a condition:

- **FalseFeedback:** If the condition evaluates to false, the text in the value column is shown. The value given can be an object expression. The attribute is optional which defaults to nothing.
- **TrueFeedback:** If the condition evaluates to true, the text in the value column is shown. The value given can be an object expression. The attribute is optional which defaults to nothing.
- **DisplayName:** text to be shown in the Knowledge Editor GUI.
- **Name:** Does not need to be unique
- **Id:** Created automatically, must be unique within the project, read only.


#### 4.1.5 Defining actions


You can configure different actions on a step.

- **Pre Actions:** These are executed as soon as the step starts, if the pre condition evaluates to true
- **True Actions:** These are executed if the ActionChoice condition evaluates to true
- **False Actions:** These are executed if the ActionChoice condition evaluates to false
- **Post Actions:** These are executed after all the other actions, just before the step completes, regardless the outcome of the evaluation of the Post condition.

It is in these actions that the domain logic of the application is executed. All of the action types above can be a list of actions. Actions are object expressions so you can manipulate context variables like setting the value of a variable or execute methods in your domain code. To configure a new action click the appropriate action name in the "Actions" ribbon. To add an action to a list of

actions right click the node in the tree view and select "Action". An action can be based on a standard action in which case you choose from the "Inherit from:" drop down list.

 Note that when implementing clients running against the Rest API you can have only one action that opens up a web interface. That means if the user for instance needs to use two web registration forms they need to be modeled as two steps.

Action Details	
Inherit from:	
None 	
Expression:	
<code>?UserAction.clientUserFormUrl("fsdemo/collect-partial-evaluation.html", ?Context)</code>	
AttributeName	Value
Remote	<input type="checkbox"/>
Suspending	<input checked="" type="checkbox"/>
ResultKey	
LocalResult	<input type="checkbox"/>
Feedback	
DisplayName	Evaluate Result
Name	EvaluateResult
Id	CXc7594e42-971e-4a1c-a9f1-caa74a849360
CreationDate	30.09.2013 13:46:28

The following attributes are displayed for an action:

- Remote: For actions to be executed somewhere else than the server, for instance on a client in a rich client scenario.
- Suspending: The step is suspending after the action and waits to be resumed. Set suspending to true for actions that opens up a web interface with clients running against the Rest API.
- ResultKey: If the action returns a result it will be assigned to the result key. Typically you write the name of a context variable here.
- LocalResult: The default value of LocalResult is false. Only set this attribute to true if the result value of an action should be locally scoped, in other words only used inside the current step.
- Feedback: This is feedback given to the user after the action is executed. Can be an object expression.
- DisplayName: text to be shown in the Knowledge Editor GUI.
- Name: Does not need to be unique
- Id: Created automatically, must be unique within the project, read only.
- Annotation: All the annotation attributes are used only for documentation and is not used in runtime.

## 5 WORKING WITH TASK LIBRARIES

The FrameSolutions Task Library (earlier called the Process library) is an important facility to expose the adaptive process support to the application users. Rather than simply creating a list of defined tasks, the Task Library contains a structure where the task definitions are represented by a small structure in order to embed logic describing how each task definition should be exposed in each situation the process library is consulted. This means that the very same task definition may be exposed differently for the same user in different situations, for example in different cases.

In addition to being used as a filtering mechanism the task library can handle different versions of a task definition. This is useful in scenarios where a business process changes at some predefined point in time. One example could be if the process for handling complaints and compensation changes 1st of January. We cannot simply change the Task definitions on new year day since this is a long running process and there will be cases in process. With the process library we can have different versions of the task definition with time stamps so all new cases will use the new definition while already started cases will use the old definition.

In a running FrameSolutions application the task library is the only place where the user can manually start new tasks. This means that tasks that can only be started automatically does not need and should not be in the task library unless you need to handle different versions of a task definition. It is not mandatory to use a process library in FrameSolutions, it is however strongly recommended if you have many processes and process bases.

### 5.1 Process libraries view

To create a process library for your project go to the "FSProject" ribbon and click the "New Process Library". Then open the process library view by double clicking the "Process Library" in the tree view.

The process libraries view shows the process libraries as a tree view on the left side. The following attributes are displayed for a process library:

- **Include Condition:** Include condition is not set in FrameSolutions applications with only one task library.
- **DisplayName:** Is the name you want to use for display.
- **Name:** Should be unique within a project. A unique Id is automatically generated but can be changed.

## 5.2 Editing process library entries

A process library entry is a representation of a task definition. We use the process library to organize task definitions, filter out only relevant task for users and handle multiple version. Therefore, a task definition can appear more than once in the process library. We can for instance present the same task definition with different names for different user groups to make the terminology fit with their daily work situation. Since we handle different versions in time we must connect the task definitions to a version of a process library entry.

To create a process library entry you click the "Process Library Entry" in the "ProcessLibrary" ribbon. The following attributes are available for a process library entry:

- Include Condition: Here you can make a condition for when to include the process library entry.
- Version: Task Definitions are connected to versions. When you create an entry you will have one version. To create more versions click the "New version" in the "ProcessLibraryEntry" ribbon. A version has an automatically generated unique id which is read only.

Each Process Library Entry version must be connected to a Task definition and has the following attributes:

- Description: Description for this particular version.
- ValidFrom: The time when the version is to be included in other words be the valid task definition for a Process Library entry. Leave this blank to indicate from the "beginning of time". Time is in the format YYYY-MM-DD
- ValidTo: The time when the version expires.
- ProcessDefinitionId: This is the name of the Task Definition, use the drop down to choose the name of a task Definition from one of the loaded Process Bases.
- DefinitionId: The unique id of the Task Definition linked to the version. Read only.

Process Library Entry

Find and filter candidates

Include Condition

Version

CXb261fa62-4fd1-4010-a4fe-ca07118f4804

CXa127f042-dee9-4f0b-b6bb-86e9c56ff55e (ValidTo: 2015-12-31 - ValidFrom: 2013-01-01)

Version Details

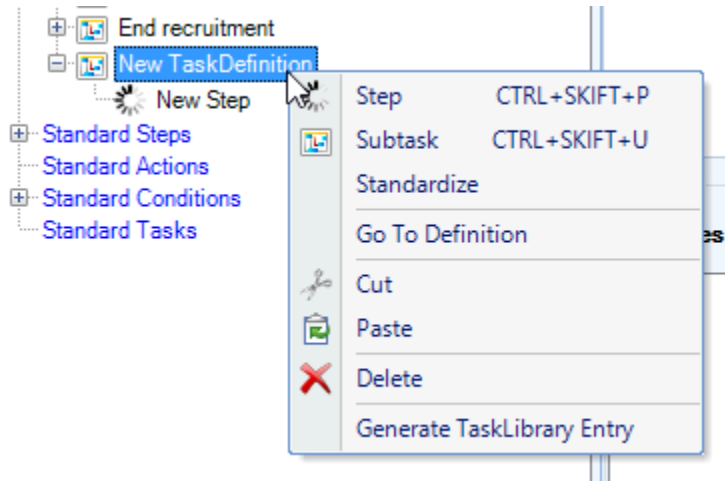
AttributeName	Value
Description	CXa127f042-dee9-4f0b-b6bb-86e9c56ff55e (ValidTo: 2015-12-31 - ValidFrom: 2013-01-01)
ValidFrom	2013-01-01
ValidTo	2015-12-31
ProcessDefinitionId	Find and filter candidates
DefinitionId	CXb90dd85b-b56c-4f03-ad2b-332554cade90



## 5.3 Creating process library entries directly from tasks

When you are working with the tasks in a processbase, you can easily navigate from a task definition to its corresponding task library entry.

Right click the task definition and choose "Generate TaskLibrary Entry".

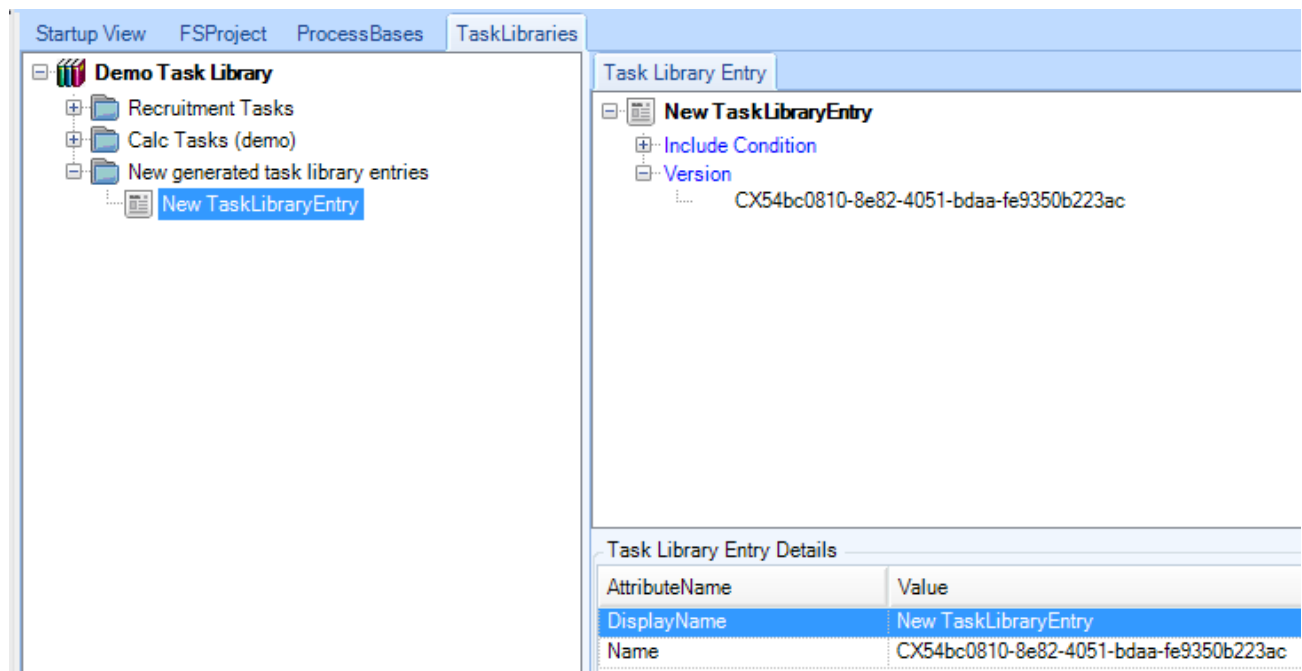


The KnowledgeEditor will switch the view to the TaskLibraries pane.

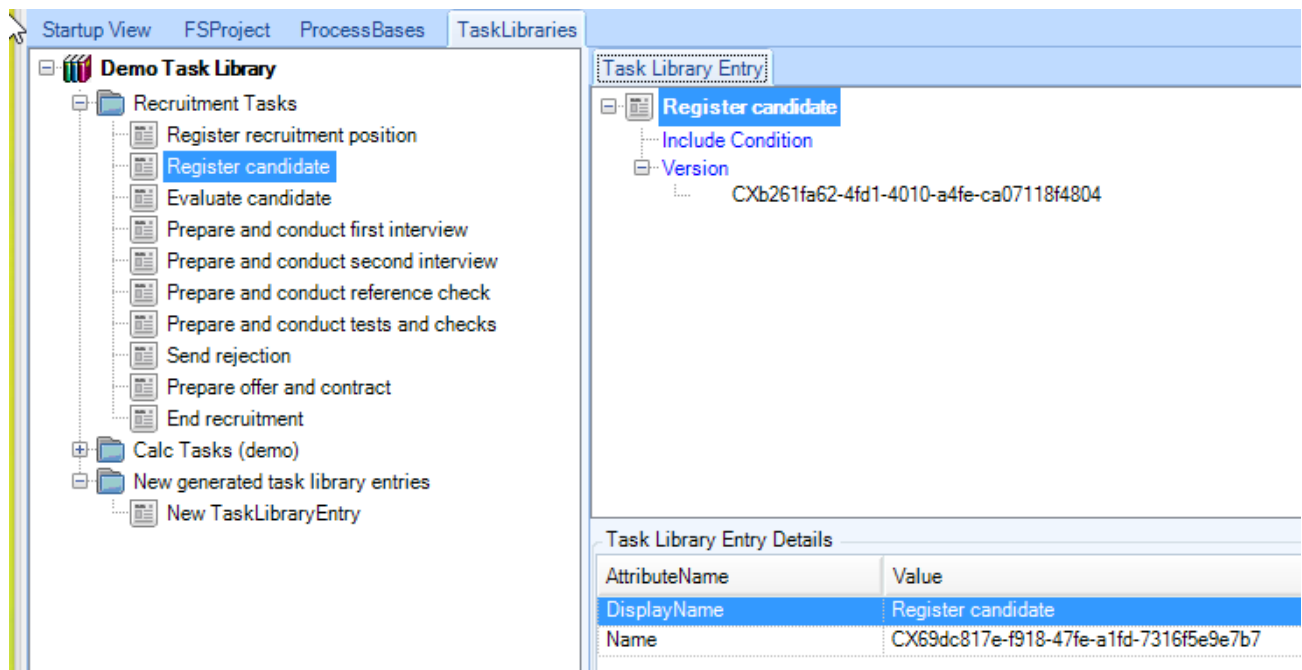
If the task did not already have a task library entry, a new will be created and placed in a new task library called "New generated task library entries".

From here, it can be moved into one of the existing task libraries or be renamed if it belongs to a brand new collection / task library.

When empty, the "New generated task library entries" can safely be deleted.



If the task already had an associated task library entry, it will be opened for editing:



## 6 WORKING WITH THE RULE EDITOR

Rules can be used directly in conditions in the task definitions. It is possible to create one or more separate rule bases. The rules can then be either executed from the conditions by specifying a named predicate or executed directly from the application.

### 6.1 Rule Base View

The rule view is opened from the project view by double clicking a rule base.

To start a new rule base click the “New Rule Base” in the “FS Project” ribbon. Give the rule base a unique name within the project. The filename displayed in the attributes is specified after you have used “Save As” to save the file.

#### 6.1.1 Defining a new rule

To create a rule, use the “New Rule” in the “RuleBase” ribbon.

In the properties view you can edit the following:

- Name, the rule name is significant if you want to use named predicates in conditions.
- Author, just for documentation
- Version, is just an annotation and not used in runtime.
- Priority is a number to assign the priority for which rules to be evaluated first. It can be any integer. The higher the integer, the higher the priority.

Rules in FrameSolutions can have action just as you can in task steps. You can have the following actions:

- Pre-actions: Always run before the rule is executed. Rules are executed if the consequent matches a named conditions or is a premise for another rule.
- True-actions: Executed if the rule evaluates to true.
- False-actions: Executed if the rule evaluates to false.
- Post-actions: Always run if the rule has been executed.



Note that if a higher priority rule has evaluated to true the other rules with the same consequent will not be executed.

To edit the rules, we need to enter a consequent and the premise. In the premise we use the same syntax as in conditions. The consequent needs a predicate name which is the name used in named conditions to execute the rule. The consequent takes any number of parameters and can also return values.

Assume we want to express a rule like the below and then return a value which is two times the person's income: IF a person is of legal age AND a person's credit rating is okay, THEN a person gets the loan.

In the knowledge editor the rule will be describes as follows:

The screenshot displays the Knowledge Editor's Rule Editor interface. At the top, there are two tabs: 'Rule Editor' and 'Properties'. The 'Rule Editor' tab is active, showing a tree view of a rule named 'testrule'. The tree includes 'Pre-actions', 'Premise' (with a question mark), 'Consequent' (highlighted in blue), 'True-actions', 'False-actions', and 'Post-actions'. The 'Consequent' node is expanded, showing the expression 'LoanWorthy(Person)(?Person.getIncome()\*2)'. Below the tree view is the 'Predicate Editor' section. It contains three main areas: 'Consequent predicate' with a 'Predicate name' field (containing 'LoanWorthy') and a 'Consequent expression' field (containing 'LoanWorthy(Person)(?Person.getIncome()\*2)'); 'Parameters' with an 'Arguments' list (containing 'Person') and 'Add'/'Remove' buttons; and 'Return expressions' with a field (containing '?Person.getIncome()\*2') and 'Add'/'Remove' buttons. At the bottom, there is another 'Predicate Editor' section with a toolbar containing 'Atomic', 'And', 'Not', 'Or', 'Exists', 'ForAll', 'Named', 'Introduce node', and 'Delete'. The 'And' button is selected, and a tree view shows an 'and' node with two children: '?Person.getAge >= 18' and '?Person.hasCreditRating()'.

## 7 VERSION CONTROL OF DEFINITION FILES

All the artifacts that the Knowledge Editor works on are stored in plain XML files. In these files, there may be references to other files like help files in HTML or image files for prototype use. FrameSolutions is a framework which together with your custom code constitute the working application. It is therefore of paramount importance that your code and the definitions are compatible. We recommend to use the same version control system for the definition files as you use for the rest of your code. FrameSolutions does not contain any versioning control system nor mandate a particular versioning control system.

If your application supports many processes and hence has many Task definitions, we recommend to split the different process areas into different process base files to avoid change conflicts. The same applies to rule bases.

In practice you should avoid having more than one process designer work with the same process base file at the same time. You should then save the xml files in the project as a logical unit of work for instance after defining one complete Task definition and commit the file to your source control system. If by any chance anyone else has changed the file in the source control system in the meantime you need to merge the file as XML text.

# 8 USING THE KNOWLEDGE EDITOR FOR RAPID PROTOTYPING

## 8.1 Introduction

Using the Knowledge Editor, one can annotate task steps with screenshot images (photos, .png etc). In this way, you can quickly create a prototype to show to users and discuss functionality.

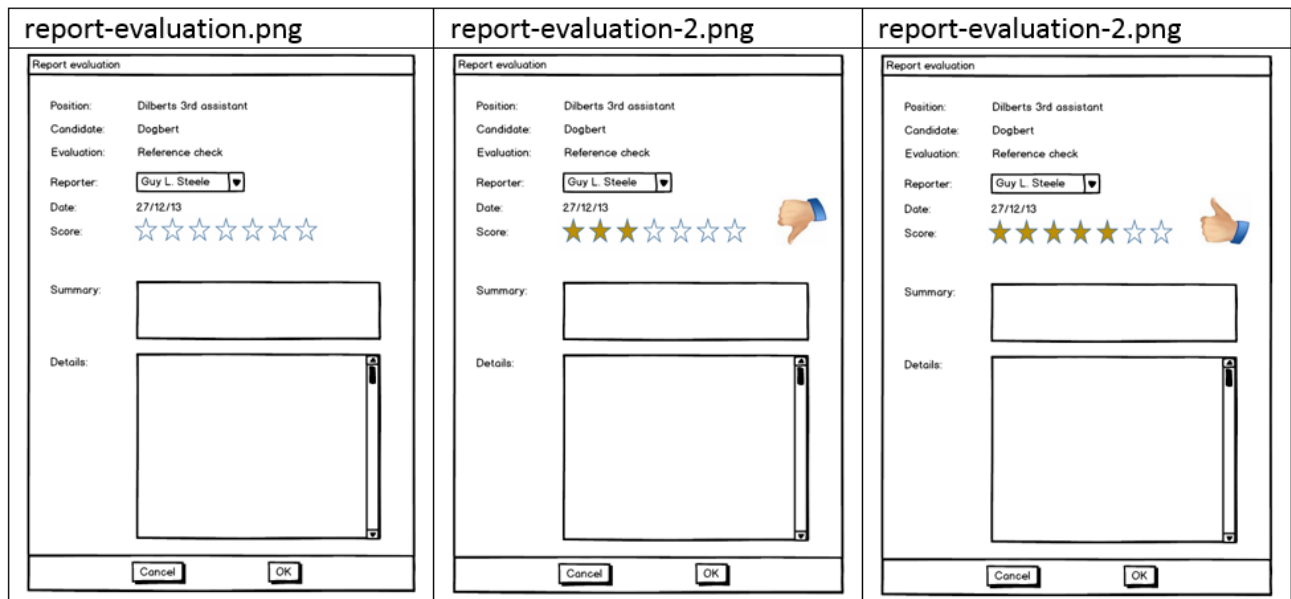
A task step can be connected to a single screenshot or to a collection of linked screenshots.

The basic idea here is that some task steps will have a GUI, and these can be demonstrated to the customer prior to their implementation.

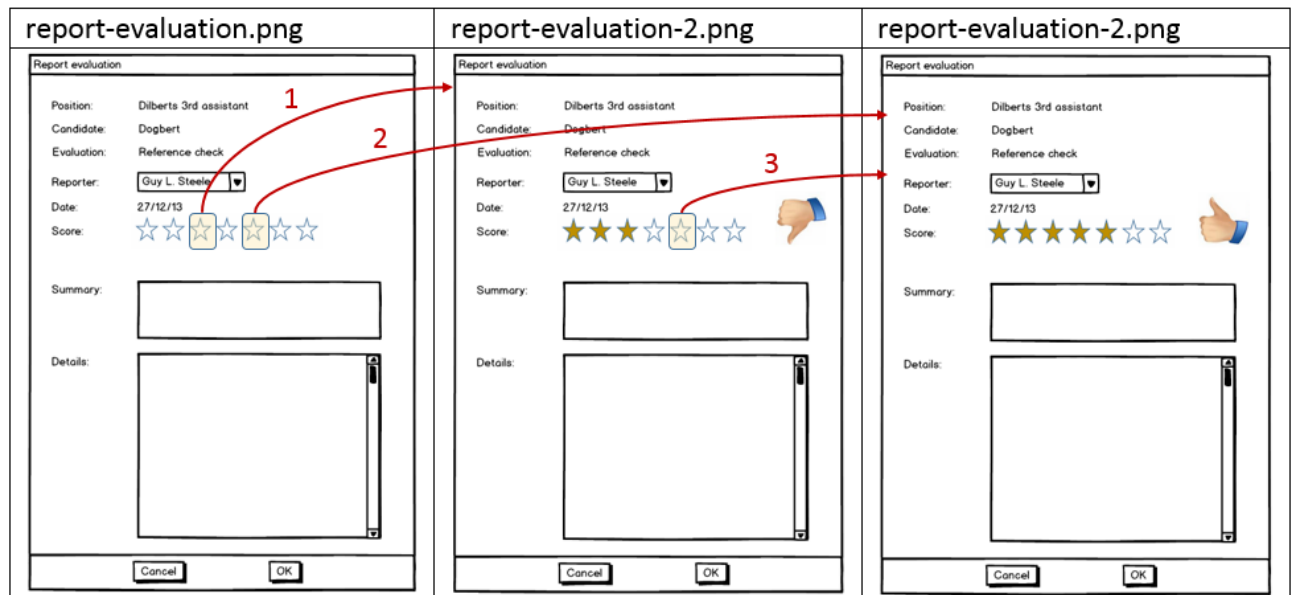
Note: You also can annotate the steps of a delivered solution with actual screenshot images; this can be useful additional information that augments the generated documentation of FrameSolutions processbases.

## 8.2 Preparations and planning the prototype

We had 3 screenshots as a preparation. (We used the balsamiq tool and captured screenshots using the windows snipping tool, but any way of creating these picture files would suffice; even drawing on paper or a whiteboard and taking photos using a phone).



In particular, we want the user to be able to indicate a score by picking a star. A score of three or fewer stars results in a "thumbs down" overall recommendation, while a score of four or more stars gives a "thumbs up". If we were doing a very simple prototype, the first screenshot (report-evaluation.png) would suffice. In these example, we want to demonstrate a little more of GUI functionality.



The figure above hints at how we want to demo the wanted dynamic behaviour.

The animations in the list below correspond to the animation number close to each transition in the figure.

1. Clicking at the third star in the first screenshot results in the second screenshot with a thumbs-down.

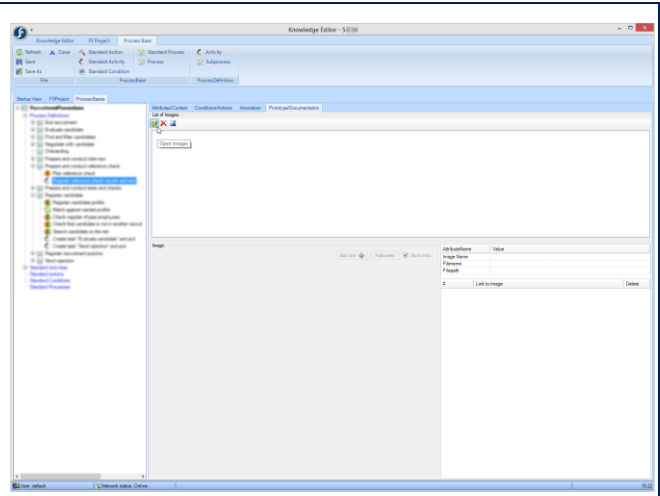
- Clicking at the fifth star in the first screenshot results in the third screenshot with a thumbs-up
- Clicking at the fifth star in the second screenshot (with a thumbs-down) results in the third screenshot with a thumbs-up.

### 8.3 Making the prototype

Start by choosing the wanted step of the task definition where you want to add a prototype.

Select the "Prototype/Documentation" tab.

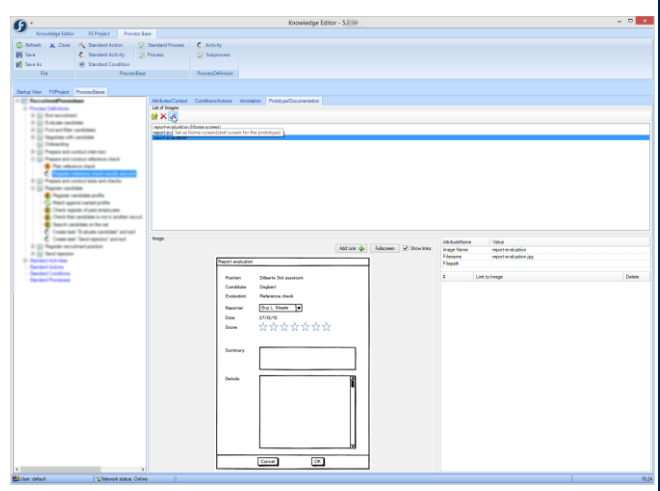
Upload the set of screenshots.



Make sure the right screenshot is set as the home screen (picture bottom left)

For building the first animation above, click the "Add Link" button.

Select a rectangular area on the screenshot (colored yellow when setting).

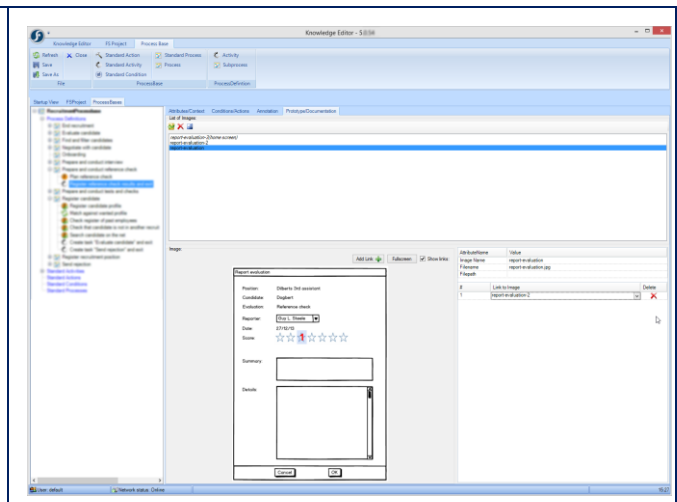




Indicate the wanted transition.

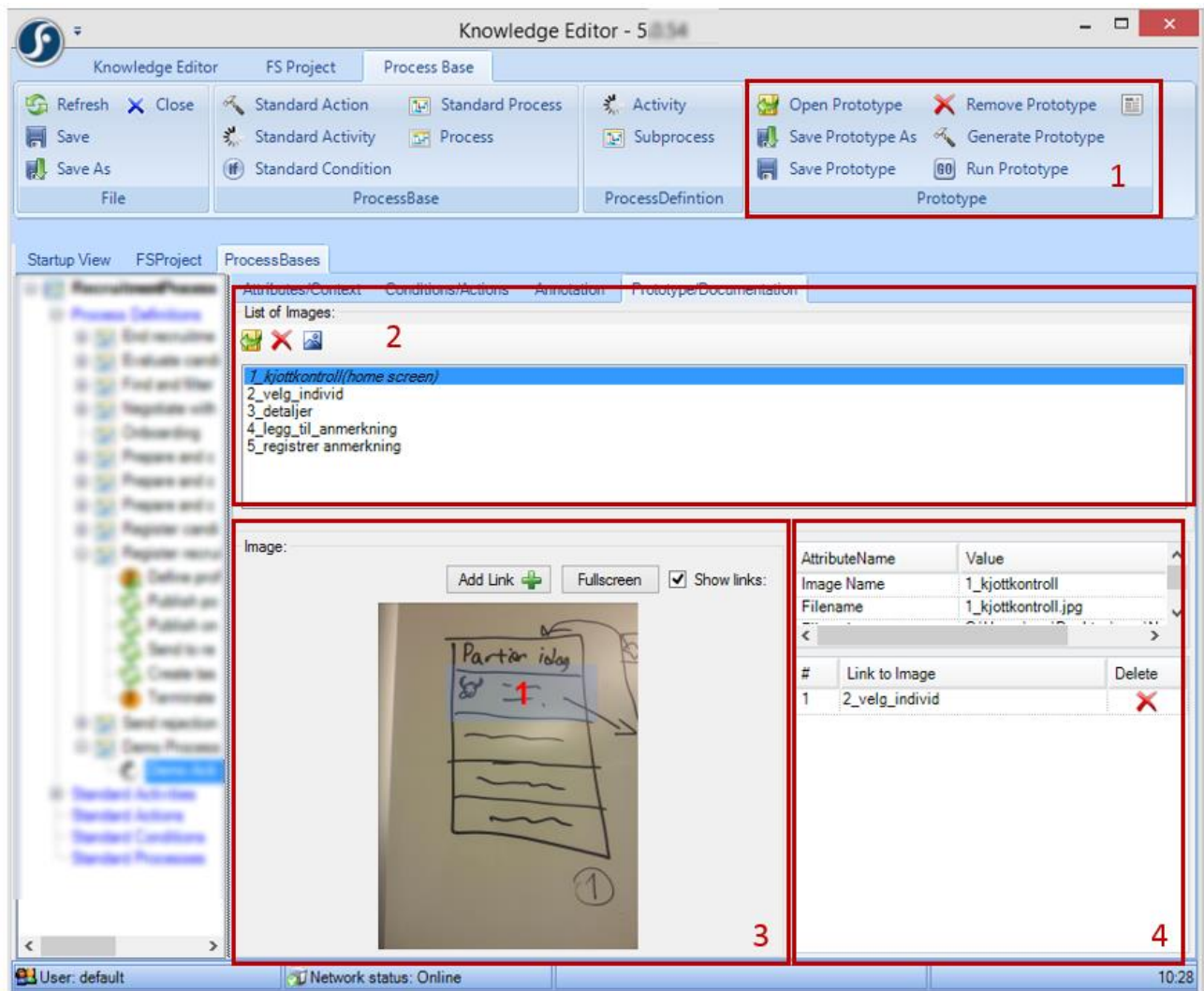
Notice the link area is indicated and numbered on the screenshot (left).

Choose the resulting screenshot to navigate to in the menu (right).



## 8.4 Brief review of prototyping functionality

A brief walktrough of the prototyping functionality is provided with reference to the screenshot above and the numbered areas of the screen.



#### 8.4.1 Prototype ribbon menu (1)

- Open Prototype - Open an existing prototype
- Save Prototype As - Save the prototype, if it already exists a new copy of the prototype is created
- Save Prototype - Save changes in an existing prototype
- Remove Prototype - Deletes the prototype
- Generate Prototype - Generates an html-version of the prototype to run in any browser
- Run Prototype - Starts the prototype in your default browser
- Export Prototype - Exports all the generated files for a prototype to a chosen folder

### 8.4.2 Prototype image list (2)

- 3 buttons at the top of Area 2:
  - Button to upload an image / screenshot
  - Button to delete selected image
  - Button to set the start image (home screen of prototype)
- The list of uploaded image files
  - In which one image is set as home screen
  - When an image in the list is selected, the image preview area below is updated accordingly (left), including the list of links from the chosen image (right).

### 8.4.3 Prototype image area (3)

This is where you work with a particular chosen image.

- The "Add link" button is used to associate a link with a chosen rectangular area of the image.
- There is a button to show the image fullscreen; you can also work with linking on the full screen.
- You can toggle links on / off. The defined links for a particular image are shown in a different color and with a number associated. In the picture above, there is only one linking area.
- Notice that you can click directly on the defined links to verify the associated behavior.

### 8.4.4 Prototype linking area (4)

This is where you associate the linking area of one image with a destination image (which it links to).

- List of links in the chosen image
- Possibility of picking which (other) image to link to in pulldown list.

### 8.4.5 Saving the prototype

The prototyping functionality is a self-contained unit with its own functionality to open / save / delete prototype parts in the prototype ribbon menu. The link from a task step (activity) to its prototype can be stored in the process base file. You therefore need to save the process base to keep the link to the prototype the next time you open the process base in the knowledge editor. The attribute "PrototypePath" of a task step stores the link to a particular prototype.

## 9 CONNECTING CUSTOM CODE

The conditions and actions created by using the Knowledge Editor will need to connect to the rest of your application code. This is done by writing object expressions which is basically expressions in either Java or C# depending on the programming language used in your application. The object expressions must match exactly with your application code and are case sensitive. To be able to connect FrameSolutions with custom code you need to agree with the application programmers on what classes, objects and methods to expose. We will strongly recommend to expose application logic as a set of classes called service providers. The service providers will then be a clearly defined layer in the application.

The whole syntax of object expressions are defined in the Appendix section.

The following variables are predefined in the specified framework.

### 9.1 Workflow

- `?CurrentTask` - The currently executing task.
- `?MainTask` - If you assign a subtask to a user other than the one performing the top-level task, the `MainTask` is the task associated with the top-level task. The framework does not use this attribute, but it is available to applications.

### 9.2 Process

- `?Process` - The currently active process (in new terminology, the task definition of `?CurrentTask`)
- `?Activity` - The currently executing activity (in new terminology, the currently executing step)
- `?Context` - The context of the current task —a convenient way to access any context variables.
- `?Strategy` - The strategy to use to evaluate rules.
- `?CurrentUserName` - The user who is currently logged in. Define this variable in your application if you wish to monitor or record who executed a process or activity.

## 10 GENERATING PROCESS BASE DOCUMENTATION

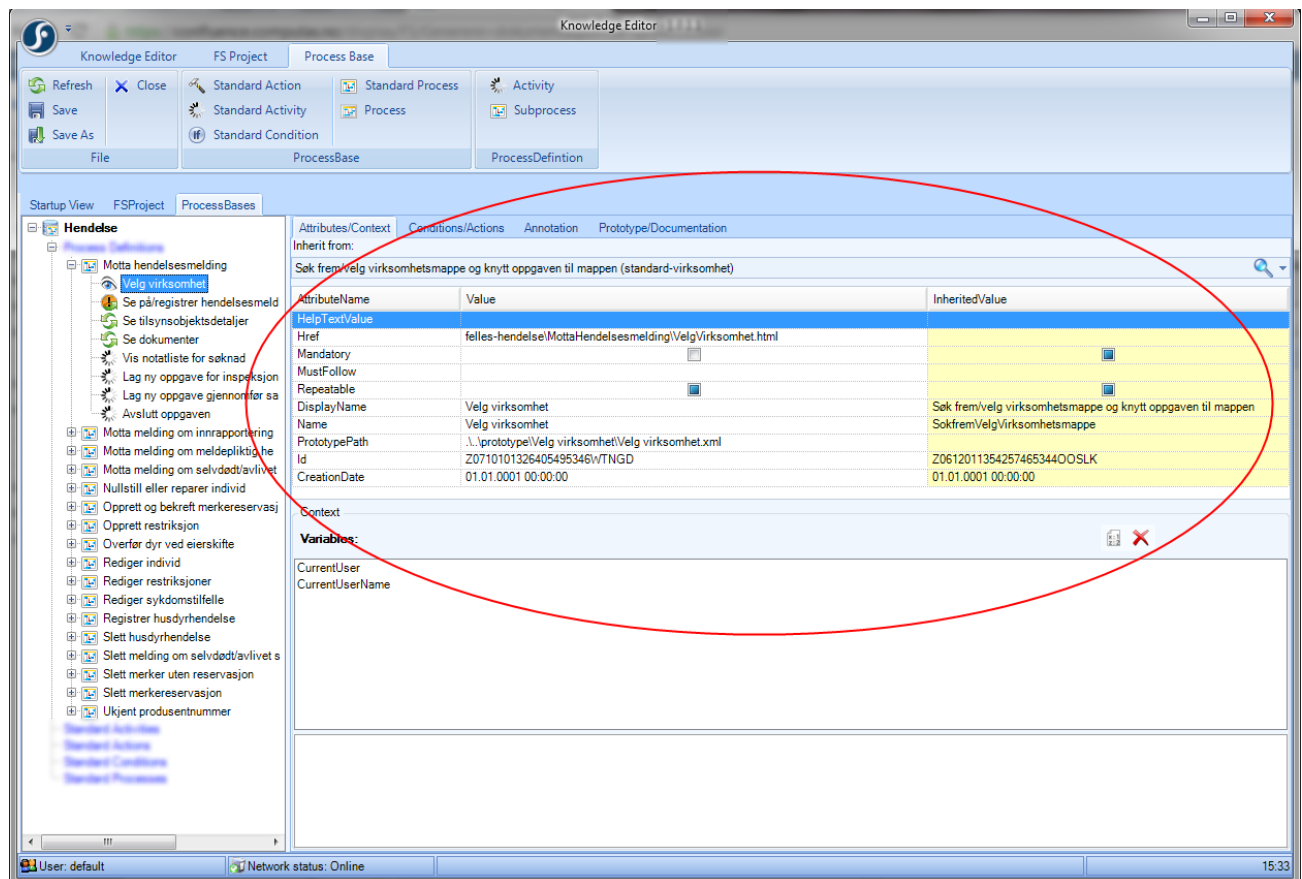
### 10.1 Documentation of process bases

Using the Knowledge Editor or command line tools, full documentation of a process base with all task and step definitions can be generated. The generated documentation is for all kinds of stakeholders - developers, functional architects, administrators and users of the solution based on FrameSolutions.

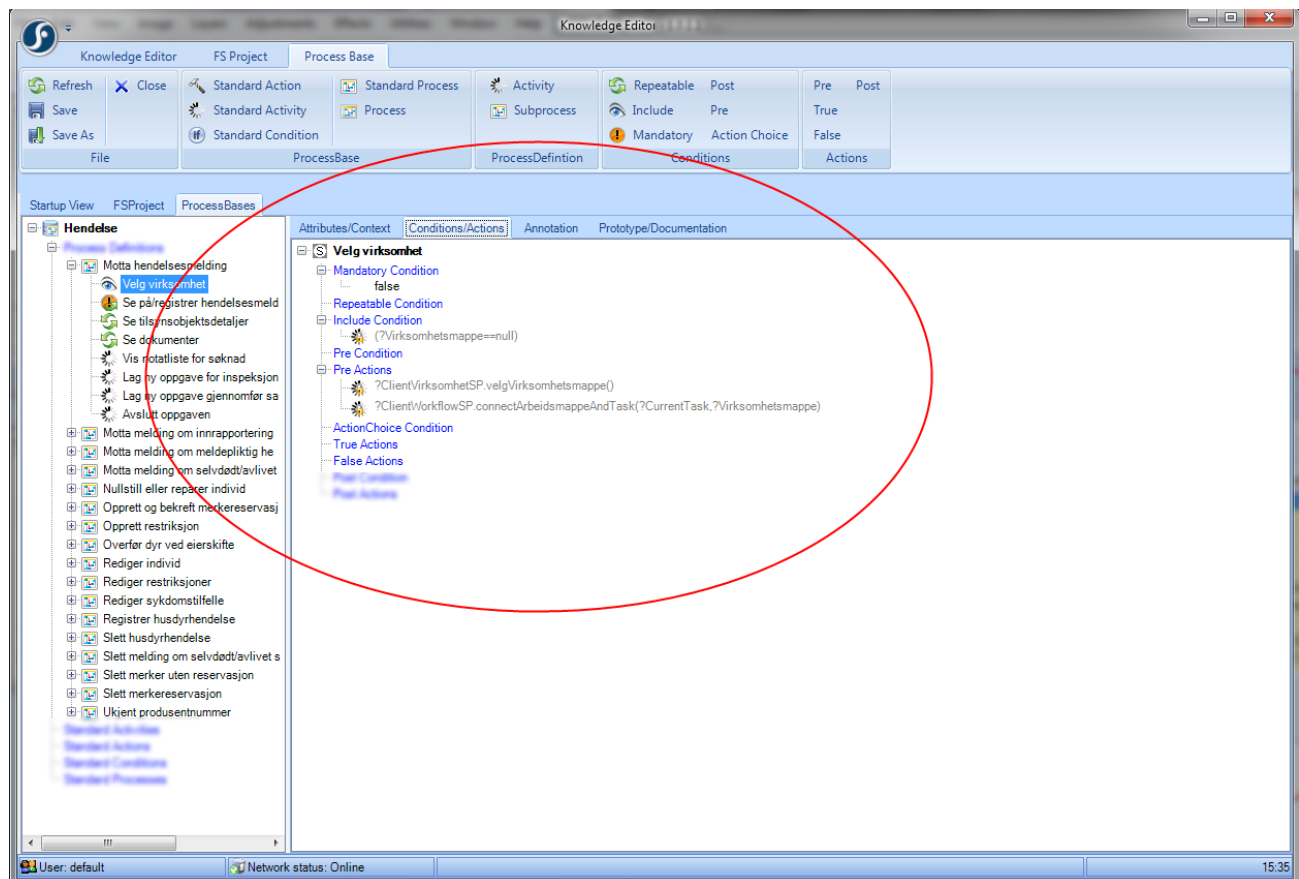
Starting from the process base xml files and the prototypes associated with task steps in these definitions, a set of html files are generated. The generated html documentation will present the defined processes in an easy-to-understand and easy-to-follow manner with simple navigation. Inheritance from standard definitions is taken into account in the same manner as it is in a FrameSolutions workbench client, in other words the documentation is presented in the same manner as presented for users executing a task and its steps.

### 10.2 Overview of the generated documentation

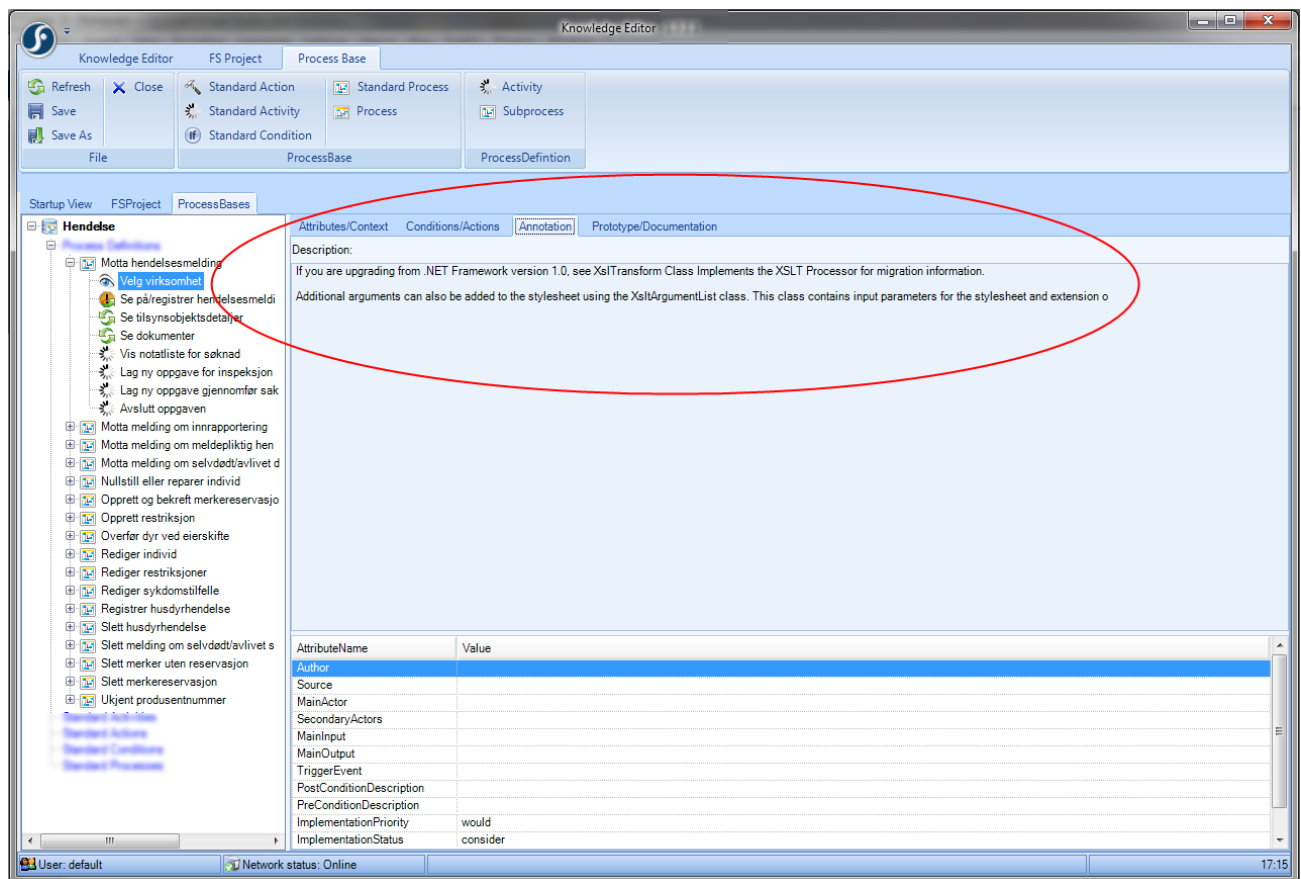
The Attributes/Context info is included, taking into account the inheritance of standard definitions.



The information from Conditions/Actions is included, also taking into account the inheritance of standard definitions.

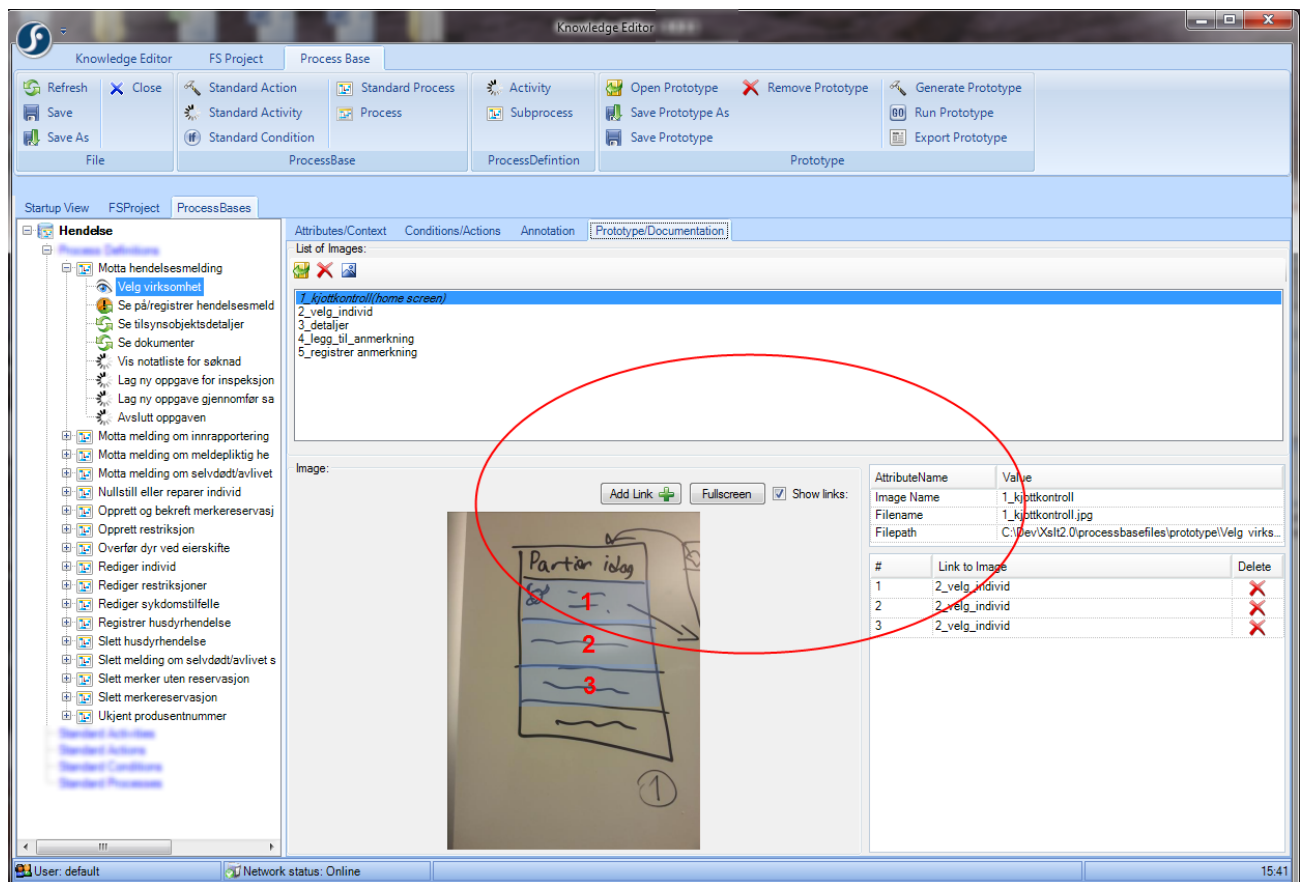


Step annotations are included.



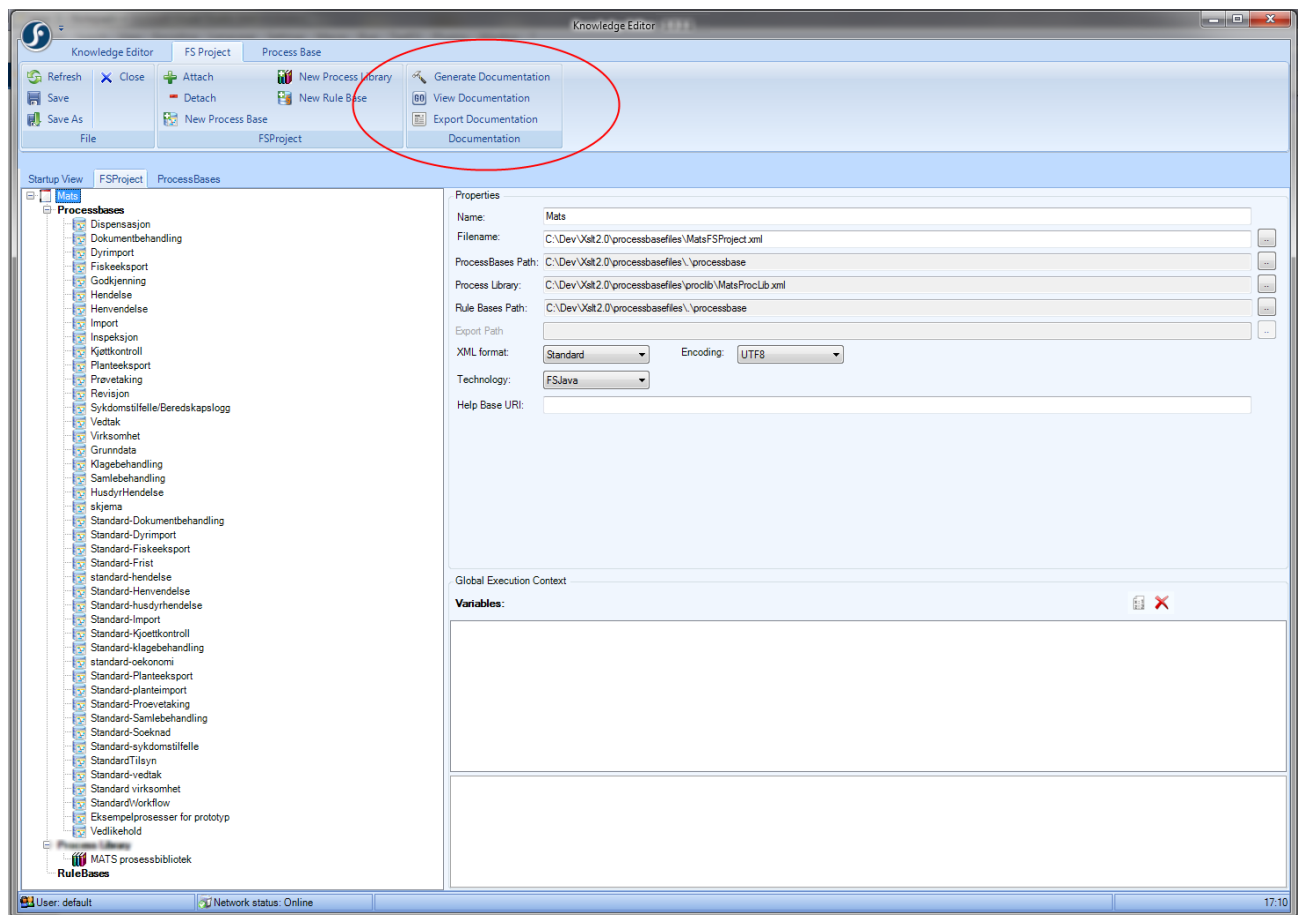
Prototypes linked to steps are included, as documentation (associated screens), but also as "live" running prototypes that you can navigate into from the documentation.





### 10.2.1 The ribbon menu to generate, view and export documentation:

- Generate Documentation - Runs transformations to generate the collection of html-files for the documentation.
- View Documentation - Open generated documentation in the your default webbrowser
- Export Documentation - Copies generated documentation with all necessary files to a chosen folder.



### Additional information

Generating documentation presupposes that there is an FS Project defined for the process bases you want to document. The menu choices for documentation generation are only shown in the FS Project ribbon menu. In order to generate process documentation from the command line, for instance as part of a software build process, the project structure should be a main folder for the FrameSolutions project and sub folders with helptext, process base and prototype:

- \fsfolder
- \fsfolder\fsproject.xml
- \fsfolder\processbase\
- \fsfolder\helptext\
- \fsfolder\prototype\

# 11 APPENDIX – OBJECT EXPRESSIONS IN THE KNOWLEDGE EDITOR

Object expressions are FrameSolutions means of connecting tasks to framework and application services. They are cascaded message-sends that use parameters and operators in the same way as the programming language you are using to develop your application. Object expression syntax is, with a few restrictions and extensions, the same as that of your development language.

An overview is provided below. The complete BNF syntax of object expressions is given in section "BNF" below.

## 11.1 Overview

Object expressions are cascaded message-sends that use parameters and operators much as your development language does. However, unlike programming language statements, which are compiled, object expressions are stored as data and evaluated at runtime.

An object expression can call instance methods on a class. The following expression calls the instance method `askUser` on the global object `GUI` to display a prompt:

```
GUI.askUser("Have you received confirmation?")
```

An object expression can also call class methods. The following expression invokes the class method `getWorkflowManager` on the class `Factory` to return an instance of the class `WorkflowManager`:

```
Factory.getWorkflowManager()
```

Messages can be cascaded. The following expression sends the instance method `sendTask` to the instance of `WorkflowManager` returned by the previous expression:

```
Factory.workflowManager.sendTask("Prepare court hearing", ?Case.getOfficer(),  
?Case)
```

As the expression above reveals, object expressions can contain variables. A variable is represented by a name prefixed with a question mark:

```
?Case
```

Access object attributes by appending a dot and the attribute name. Assuming that `Case` is bound to an object representing a court case having an attribute named `involvedParties`, the following expression returns the involved parties:

```
?Case.involvedParties
```

You can assign a value to a variable by telling the context to bind the name to the value:

```
?Context.bind ("Officer", "Stan Mann")  
?Context.bind ("Officer", ?Case.getOfficer())
```

Object expressions can also access atomic values, such as the string in the expression above.

Create instances using the keyword `new`:

```
new com.computas.RandomGenerator()
```

Use operators, such as in this comparison:

```
new com.computas.RandomGenerator().getRnd() <
    new com.computas.RandomGenerator().getRnd()
```

Access classes using the keyword `class`. The following expression computes the length (in characters) of the requested class name and compares it with the atomic integer 12.

```
com.computas.Person.class.getName().length() >= 12
```

## 11.2 BNF

In addition to the operators and expressions described below, you can use the operators and expressions available in Java. The object expression resolver automatically casts between primitives and their corresponding classes.

```
ObjectExpression =
    UnaryExpression ObjectExpressionRest
UnaryExpression =
    UnaryOperator UnaryExpression Modifiers |
    "(" ObjectExpression ")" Modifiers |
    StaticClass ClassOperator |
    AtomicValue Modifiers
ObjectExpressionRest =
    BinaryExpressionRest ObjectExpressionRest
BinaryExpressionRest =
    BinaryOperator UnaryExpression
UnaryOperator =
    Plus |
    Minus |
    Not
BinaryOperator =
    Equal |
    NotEqual |
    Or |
    And |
    LT |
    LTEq |
    GT |
    GTEq |
    Plus |
    Minus |
    Mult |
    Div |
    Remain
```

```

Modifiers =
    "." Identifier Arguments Modifiers
Identifier =
    Word |
    Name
NumberIdentifier =
    PosInteger
Arguments =
    "(" ObjectExpression RestArguments |
    "("
RestArguments =
    ")" |
    ", " ObjectExpression RestArguments
AtomicValue =
    PosNumber |
    String |
    True |
    False |
    Null |
    Variable |
    StaticClass |
    Constructor
PackageName =
    Word "." PackageName |
    ""
Constructor =
    New StaticClass Arguments
StaticClass =
    PackageName Name
Variable =
    ("?" | "$") (Identifier | NumberIdentifier)
PosNumber =
    PosFloat |
    PosInteger
Not = "!"
Equal = "=" ("=")?
NotEqual = "!="
Or = "|" ("|")?
And = "&" ("&")?
LT = "<"
LTEq = "<="
GT = ">"
GTEq = ">="
Plus = "+"
Minus = "-"
Mult = "*"
Div = "/"
Remain = "%"
ClassOperator = ".class"
Null = ("null" | "nil")

```

```

True = "true"
False = "false"
New = "new"
String = (String1 | String2)
String1 = "\"" (~["\""])* "\""
String2 = "'" (~["'"])* "'"
Name = Big (Char | Digit)*
Word = Small (Char | Digit)*
Char = (Small | Big)
Small = ["a"-"z", "E", " ", "A"]
Big = ["A"-"Z", "f", "y", ".", "_"]
PosFloat = (PosExp | PosReal)
PosExp =
    (PosReal | PosInteger) ["E", "e"] (["-", "+"])? PosInteger
PosReal = PosInteger "." Digits
PosInteger = (Digits)
Digits = (Digit) +
Digit = ["0"-"9"]

```