Core SWRL Built-ins (supported by Pellet)

swrlb:greaterThan

swrlb:replace

swrlb:stringConcat

swrlb:cos

swrlb:yearMonthDuration

swrlb:substringBefore

swrlb:lessThan

swrlb:substringAfter

swrlb:divide

swrlb:stringLength

swrlb:upperCase

swrlb:normalizeSpace

swrlb:substring

swrlb:round

swrlb:notEqual

swrlb:greaterThanOrEqual

swrlb:equal

swrlb:dateTime

swrlb:stringEqualIgnoreCase

swrlb:dayTimeDuration

swrlb:matches

swrlb:anyURI

swrlb:mod

swrlb:tokenize

swrlb:time

swrlb:subtract

swrlb:ceiling

swrlb:lowerCase

swrlb:resolveURI

swrlb:multiply

swrlb:integerDivide

swrlb:lessThanOrEqual

swrlb:abs

swrlb:endsWith

swrlb:pow

swrlb:sin

swrlb:startsWith

swrlb:translate

swrlb:booleanNot

swrlb:unaryMinus

swrlb:contains

swrlb:containsIgnoreCase

swrlb:add

swrlb:floor

swrlb:roundHalfToEven

swrlb:tan

swrlb:date

swrlb:unaryPlus

Custom Internal SWRL Built-ins:

Name	Arguments	Description
no	(individual *ind1*, literal *rel*, individual *ind2*)	Checks if *ind1* does not have a relation of type *rel* with *ind2*
relGT **relGE** **relEQ** **relLE** **relLT**	(literal *value*, individual *ind1*, literal *rel*, [literal *cls*])	Checks if the number of relations of type *rel* that *ind1* has is (Greater Than, Greater or Equal, Equal, Less or Equal, Less Than) the number provided by *value*. *cls* is **optional** and can be used to specify the class range of that relation. If the target of a relation is not contained in the provided range, it is ignored. By default, the range is **owl:Thing**, if *cls* is not specified.

Prefix:

The Custom Internal SWRL Built-ins have the same prefix (**ro**) because the ontology in which they were created is called **ro** (e.g.: **ro**:no(arg1, arg2, arg3)). When the user creates a new **Builtin** instance, its prefix will be the same as the user ontology's name.

Arguments:

rel – String which specifies the full IRI of an existing Object Property.
cls – String which specifies the full IRI of an existing Class.
ind1, *ind2* – Variable which represents an existing individual.
value – Integer which specifies the number of relations between two individuals.

Example:

Check if an individual of class **DisplayController** has exactly 0 relations of type **uses** with individuals of class **Backlight**.

```
DisplayController(?dc) ^ ro:relEQ(0, ?dc, "esrg:upper#uses", "esrg:calculator#Backlight") -> ...
```

Note that it is not the same thing as:

```
DisplayController(?dc) ^ Backlight(?b) ^ ro:no(?dc, "esrg:upper#uses", ?b) -> ...
```

The **no** built-in only works with existing individuals. If the ontology only contains one individual, it is not suitable.

Steps to create a custom external SWRL built-in:

1. Create or open user Ontology in Protégé.
2. Make sure that both **upper** and **ro** ontologies are imported.
3. Create an instance of **swrl:Builtin** named after the custom built-in.
4. Specify the rule's arguments.

The **swrlArguments** class contains arguments groups. Each Built-in can have one or more arguments groups. These are specified with the **hasArguments** object property.

Built-in (Individual)	**(Object Property)**	**Arguments Group (Individual)**
relEQ	hasArguments	relationClassCounter
		relationCounter

Each argument group is composed of one or more arguments, which are specified through annotations.

Arguments Group (Individual)	**(Annotation Property)**	**Argument type (Data Property)**
relationClassCounter	Argument_1	literal
	Argument_2	individual
	Argument_3	literal
	Argument_4	literal
relationCounter	Argument_1	literal
	Argument_2	individual
	Argument_3	literal

The user can use an existing arguments group or create a new one. There are 3 types of arguments:

- **literal** – A literal is composed of a string (lexical form) and a datatype specifying how to interpret this string. It can be used to represent many data type such as strings and integers.
- **individual** – variable which represents an existing individual
- **unbound** – variable which has a null value before the built-in is executed. This variable is assigned during the built-in's execution.

5. Create a new SeML project and import the user ontology
6. Open the project folder and then open "[project_name]_template"
7. Copy the custom built-in file to "[project_name]_swrl" and customize it
8. Edit the SeML file and the built-in will be automatically detected, compiled and loaded

Note: the ontology should be saved as "OWL/XML Syntax" or "OWL Functional Syntax" to avoid compatibility issues on Protégé. Individuals of the **Builtin** class won't be kept, when saving in other file formats.

Note 2: the SWRL built-ins receive a Pellet Reasoner instance, which has no support for SWRL rules. That support was purposely disabled to avoid rule execution loops. However, the user must be aware of this limitation when implementing the desired behavior.

Note 3: by default, a built-in template prints a "." each time it is executed. The user can change this symbol or print any desired information to the standard output/error.