Springles Ruleset Vocabulary 0.1

Namespace Document 11 Jan 2013

Document URL:
http://stettler.fbk.eu/ckr/springles/spr.html (HTML, PDF)
Vocabulary URL:
http://stettler.fbk.eu/ckr/springles/spr.owl (RDF/XML, TURTLE)
Authors:
Luciano Serafini, Loris Bozzato (FBK-Irst)
Francesco Corcoglioniti

Abstract

The Springles Ruleset Vocabulary defines the classes and the properties that can be used to specify a Springles ruleset in RDF, after which it can be loaded to control inference in a Springles repository. This document describes the vocabulary and provides general instructions and reference documentation for its usage.

Status of This Document

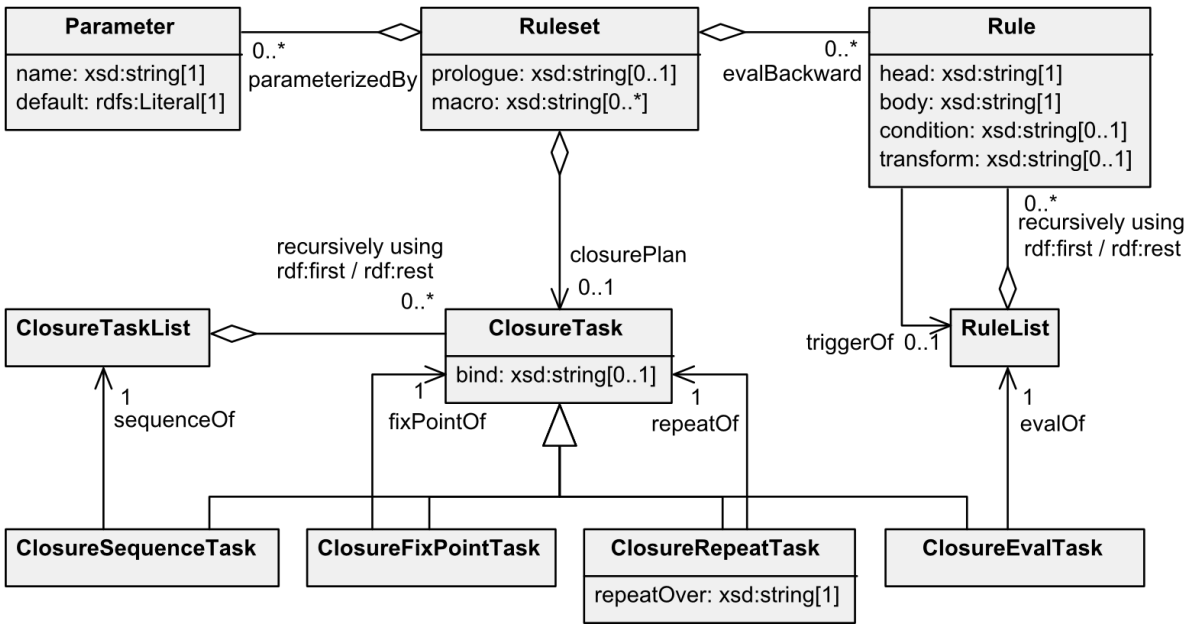
The Springles Ruleset Vocabulary is a work in progress. This document describes the latest version of the vocabulary as supported by the Springles platform.

Table of Contents

- Overview
- Rulesets
- Rules
- Closure plans
- Design considerations
- Terms reference

Overview

The following UML class diagram informally presents an overview of the vocabulary. Classes are rendered as UML classes, datatype properties as attributes and object properties as UML relations; minimum and maximum cardinalities and expected datatypes are also shown; note that spr:ClosureTaskList and spr:RuleList are refinements of RDF lists and as such are encoded using `rdf:first` and `rdf:rest` properties. The components of the vocabulary - namely rulesets, rules and closure plans - are detailed in the following sections.



The vocabulary namespace is `http://dkm.fbk.eu/springles/ruleset#`. The suggested prefix for referencing the vocabulary is `spr:`.

A list of classes and properties is reported below, with links to their reference documentation:

Classes: | ClosureEvalTask | ClosureFixPointTask | ClosureRepeatTask | ClosureSequenceTask | ClosureTask | ClosureTaskList | Parameter | Rule | RuleList | Ruleset |

Properties: | bind | body | closurePlan | condition | default | evalBackward | evalForward | evalOf | fixPointOf | head | macro | name | parameterizedBy | prologue | repeatOf | repeatOver | sequenceOf | transform | triggerOf |

Rulesets

A **ruleset** consists of a set of rules that are evaluated either in forward- or backward- chaining and jointly specify the type of inference provided by a Springles repository. A ruleset is an instance of class Ruleset; this must be explicitly stated in order for Springles to recognize a ruleset. An example of ruleset definition is shown below.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix spr: <http://dkm.fbk.eu/springles/ruleset#> .
@prefix :    <http://dkm.fbk.eu/springles/rdfs-merged#> .

:ruleset a spr:Ruleset ;
    spr:parameterizedBy
        [ spr:name "enable_tbox_rules" ; spr:default "true"^^xsd:boolean ] ;
    spr:prologue """
        PREFIX rdf:
        PREFIX rdfs:
        PREFIX xsd:
        BASE  """ ;
    spr:macro "HOME = <>" ;
    spr:macro "DIFF(a,b,c) = FILTER ($a != $b && $a != $c && $b != $c)" ;
    spr:evalBackward :r1 ;
    spr:closurePlan :plan .
```

Backward-chaining rules (property spr:evalBackward) such as `r1` in the example are evaluated at query time through query rewriting (this feature is currently unsupported in Springles). Forward-chaining rules are evaluated each time a repository is modified in order to materialize a repository logical closure. The closure computation process is specified through a **closure plan** (property closurePlan), which decomposes the overall process in tasks and sub-tasks and allows to fine-tune and optimize the process.

A ruleset is optionally associated to a number of ruleset parameters, a 'SPARQL prologue' and a set of 'macros':

- **Parameters** (property spr:parameterizedBy) permit to inject user-supplied constant values to customize the evaluation of rules, such as parameter `enable_tbox_rules` in the example used to enable/disable RDFS rules for TBox reasoning. Each parameter is an instance of class spr:Parameter and is characterized by a name (property spr:name) and a default value (property spr:default), both mandatory. When a ruleset is loaded in Springles, the values of parameters provided by the user (or their default value, when not specified) are used together with the binding inheritance mechanism (see below) to fix the values of certain SPARQL variables in rules and other SPARQL expressions, thus allowing to customize their behaviour.
- The **SPARQL prologue** (property spr:prologue) is a SPARQL expression that defines the namespaces and/or the base URIs for parsing SPARQL expressions in the ruleset. The syntax is the one of SPARQL: prefixes are declared as '`PREFIX pref: <uri>`', (as with prefixes `rdf:`, `rdfs` and `xsd:` in the example), while the base uri is declared as '`BASE <uri>`'.
- **Macros** (property spr:macro) are constants or functions (such as `HOME` and `DIFF(a,b,c)` in the example) which can be used inside SPARQL expressions in the ruleset to factorize SPARQL snippets and make ruleset definitions less verbose. The syntax for macro definitions is '`macro(arg1, ..., argN) = template`'. Name and arguments are identifiers, i.e., letter followed by letters, numbers or underscore. Arguments are optional; if absent, parenthesis can be omitted. If a macro accepts a parameter `P`, its template may reference `P` using syntax `#P`; at the same time, the macro cannot use a SPARQL variable `?P` with the same name (to avoid hard to spot errors caused by wrong typing). When a ruleset is loaded, macros are expanded by replacing their references in SPARQL expressions with the associated template, whose parameter placeholders are substituted with the values specified in the macro invocation.

Additional ruleset properties such as a `rdfs:label` can be specified: while ignored by Springles, they can be used by tools to display or manipulate metadata about rulesets.

Rules

An inference **rule** specifies using a SPARQL-based syntax how inferred statements can be deduced through the

matching of statements already contained in the repository. An example of rule is reported below.

```
:rdfs11 a spr:Rule ;
    spr:triggerOf ( :rdfs2 :rdfs3 :rdfs7 :rdfs9 :rdfs11 ) ;
    spr:condition "?enable_tbox_rules" ;
    spr:head """ GRAPH ?inference_graph { ?x rdfs:subClassOf ?z } """ ;
    spr:body """ GRAPH ?graph { ?x rdfs:subClassOf ?y .
                                ?y rdfs:subClassOf ?z . }
                 $DIFF(?x, ?y, ?z)
                 FILTER NOT EXISTS { GRAPH ?graph { ?x rdfs:subClassOf ?z } } """ .
```

A rule has a mandatory body, that has to be matched by statements in the repository, and a mandatory head, that specifies which statements have to be inferred if the body is matched:

- The **body** (property spr:body) is a SPARQL expression of the kind used in WHERE clauses (`GroupGraphPattern` production in SPARQL 1.1 grammar). The body is evaluated against statements stored in the repository, setting all variables in the expression for which a ruleset parameter or an inherited binding (see below) is available with the corresponding value. In general, the body has the tasks of (1) implementing the premises of the rule; and (2) checking that the statements produced by the head are not already stated explicitly in the repository. Note that it is not necessary (and it may be detrimental to performances) to remove duplicates at this level, as this is already done efficiently by Springles engine. Note, also, that if the body is empty, the head will be always inferred and the rule will behave as an axiom.
- The **head** (property spr:head) is a SPARQL expression (a `GroupGraphPattern` too) restricted to use only triple patterns (as in a SPARQL `CONSTRUCT` clause), plus the `GRAPH` keyword that permits to generate statements inside specific graphs (differently from a `CONSTRUCT` query). The head expression may reference variables, whose values derive either from the matched rule body or from ruleset parameters and/or inherited bindings (as `?inference_graph` in the example, see binding inheritance later).

A rule is optionally associated to a condition, a transformer and a list of triggered rules:

- The **condition** (property spr:condition), if specified, is a generic SPARQL boolean expression (Constraint production in SPARQL 1.1. grammar) that must be satisfied by ruleset parameters and inherited bindings in order for the rule to be possibly evaluated. In the example above, if ruleset parameter `enable_tbox_rules` is `false`, then rule `:rdfs11` will never be considered for evaluation. Note that the expression cannot query for statements stored in the repository.
- The **transformer** (property spr:transform), if specified, is executed after the evaluation of the rule body and before the materialization of the rule head. It permits to invoke external Java code to implement specialized processing not possible or difficult/inefficient to realize in SPARQL. More precisely, the stream of solutions (tuples of variable bindings) extracted from the evaluation of the rule body is fed into the transformer, which produces another stream of solutions which is the one actually used to materialize the head. The transformer is identified by a SPARQL function call (`FunctionCall` production in SPARQL 1.1 grammar). Its URI identifies a Java static method implementing the transformer and is either in the form `<springles:builtin_transformer>` or `<java:class_name.method_name>`. Its arguments are used to configure the transformer and are SPARQL expressions evaluated based based on ruleset parameters and inherited bindings (access to statements in the repository is disallowed).
- **Triggered rules** (property spr:triggerOf) are the rules that may fire as a consequence of the subject rule being fired, i.e., they are the successor of the rule in a *rule dependency graph*. This is an optional property that permit users to supply the inference engine with the result of a static analysis of rule dependencies, which is then exploited to avoid unnecessary rule evaluations. If this property is not supplied, the engine will conservatively assume that every rule in the plan may fire as a result of the subject rule being fired. Note that the value of the property is a RuleList implemented as a specialization of RDF lists: a list is used in order to differentiate between the case the property is not supplied (e.g., because unknown) and the case there are no triggered rules (empty list); the order of rules in the list is irrelevant.

A rule's head, body and condition expression may use the macros defined in the ruleset. This is shown in the rule body of the example, where macro `DIFF` is called. When loading the ruleset, it will be expanded with the string `FILTER ($x != $y && $x != $z && $y != $z)`, with macro parameters `a`, `b` and `c` replaced with supplied expressions `?x`, `?y` and `?z`. Note that, should one of those expression contain a comma, it must be surrounded by parenthesis `( ... )`, or it is interpreted as multiple parameter assignments.

Closure plans

A closure plan instructs the system on how to compute and materialize a logical closure. A plan is decomposed into a number of **closure tasks**, arranged in a hierarchy. Execution of a plan starts from the *root* task identified by property closurePlan in the ruleset. Children tasks can then be executed as part of the execution of the parent task, that controls their execution according to a specific strategy (e.g., sequential vs fix-point execution); the process is repeated recursively for child tasks. Note that the topology of the task hierarchy is actually a DAG with a single root, i.e., it is allowed for a task to be included as a child in multiple positions of the hierarchy.

An example of closure plan (implementing per-graph RDFS inference) is shown below. In the example, `:plan` is the root task referenced by the ruleset. It is a spr:ClosureRepeatTask that executes a child task `:graph-closure` for each graph in the repository. In turn, task `graph-closure` is executed as the sequence of two tasks: `:add-axioms` and `:do-closure`. Note that the RDF definition of a task may be restricted by stating the minimum possible amount of statements (as happen in the example for task `:do-closure`), relying on the system to infer missing information when loading a ruleset based on the semantic constraints in the Springles Ruleset Vocabulary (e.g., to infer that the task type is `spr:ClosureFixPointTask`); this principle applies in general to the whole ruleset RDF definition.

```
:plan a spr:ClosureRepeatTask ;
    spr:repeatOver """ SELECT ?graph WHERE { GRAPH ?graph { ?s ?p ?o } } " ;
    spr:repeatOf :graph-closure .

:graph-closure a spr:ClosureSequenceTask ;
    spr:bind "?inference_graph = IRI(concat(str(?graph), '-inf'))" ;
    spr:sequenceOf ( :add-axioms :do-closure ) .

:add-axioms a spr:ClosureEvalTask ;
    spr:evalOf ( :rdf_axioms :rdfs_axioms ) .

:do-closure spr:fixPointOf [ spr:evalOf (
        :rdf1 :rdfs2 :rdfs3 :rdfs4a :rdfs4b :rdfs5 :rdfs6
        :rdfs7 :rdfs8 :rdfs9 :rdfs10 :rdfs11 :rdfs12 :rdfs13 ) ] .
```

Binding inheritance mechanism

When a closure task is executed, it *inherits* a set of **variable bindings** (i.e., `variable = value` pairs) that are used to fix the values of matching variables in referenced rules and SPARQL expressions. Variable bindings are propagated in a top-down mode. This starts with the root task, which inherits a binding for each ruleset parameter, where the variable is the parameter name and the value is the one supplied by the user at configuration time (or the default value of the ruleset parameter). Each tasks can then define its own bindings, which are merged with (and possibly override) the ones inherited and are then propagated down the hierarchy. Binding definition is either implicit or explicit:

- **Implicit binding definition** is associated to certain kinds of tasks, such as the spr:ClosureRepeatTask task, which defines the bindings of loop variables in each iteration. This is shown in the example, where task `:plan` retrieves all the graphs in the repository with a query and then executes the child task `:graph-closure` binding variable `?graph` to a different (retrieved) graph URI each time.
- **Explicit binding definition** is controlled by the user through the use of the spr:bind property, which is supported for every type of closure task. The property takes as value a string expression of the form `variable = SPARQL expression`, where the SPARQL expression is evaluated based on all the bindings received in input by the task. This occurs in the example for task `:graph-closure`, which generates the URI of the graph where to store inferences and binds it to variable `?inference_graph`; this binding, together with the one for variable `?graph`, are then inherited by child tasks `:add-axioms` and `:do-closure`, and from them inherited by referenced rules `:rdf_axioms`, `:rdfs_axioms`, `:rdf1` ... `:rdfs13`.

Types of closure tasks

The following types of closure task are defined, each one with its own semantics:

- **Eval** (class spr:ClosureEvalTask). It consists in the parallel evaluation of zero or more rules, specified using property spr:evalOf. In case only a limited number `N` of rules can be evaluated in parallel (this is a configuration option of Springles which should be fixed based on the number of available CPU cores), rules are queued according to the order specified by spr:evalOf and the first `N` rules are evaluated, starting the evaluation of another rule waiting in the queue each time the evaluation of a preceeding rule terminates. Inferred statements are buffered and written back to the repository only after all the rules have been evaluated, thus implying that each evaluated rule will not 'see' the inferences produced by other evaluated rules.
- **Sequence** (class spr:ClosureEvalTask). It consists in the sequential execution of zero or more child tasks, specified using property spr:sequenceOf. Note that execution is strictly sequential, in the sense execution of a child task is started only after the execution of the previous task completes.
- **Fix point** (class spr:ClosureEvalTask). It consists in the fix-point execution of a child task, i.e., the child task is repeatedly executed until it produces no more inferred statements. The child task is specified using property spr:fixPointOf.
- **Repeat** (class spr:ClosureEvalTask). It realizes a sort of 'for-each' primitive that consists in the repeated execution of a child task, specified by property spr:repeatOf, for each tuple obtained from the evaluation of a `SELECT` query, specified by property spr:repeatOver. The query is executed once when the task is executed; query results are collected and bufferend and the child task is executed for each result tuple in the order they are returned by the query. Note that the SPARQL bindings in the result tuple are propagated to the child task as additional variable bindings, providing a way to influence the execution of the child task.

Design considerations

Concerning the ruleset conceptual model, a point worth of consideration is the relation between closure plans and rule dependencies encoded using property spr:triggerOf. The two mechanisms are orthogonal. The Springles engine will always stick to the closure plan when performing inference. Each time the plan asks for the execution of a rule, the engine checks whether the rule can possibly produce some inferred statement by consulting rule dependencies as encoded by spr:triggerOf properties and by checking which rules previously fired. If it is determined that no inference can be produced, then the rule is not executed at all, possibly leading to substantial time savings due to the rule body not being evaluated.

Another aspect worth discussion is the relation between binding inheritance and macros. While bindings can be used in place of constant macros, they are though as a way to dinamically adapt a ruleset to inputs supplied from the user at configuration time or to the outcome of the execution of some closure task at inference time. Macros, on the other hand, are a static mechanism that can be used to save some key-stroke when writing a ruleset; in fact, the engine expands macros when a ruleset is loaded, and then gets rid of them.

Concerning the way the model has been encoded in a vocabulary, it is worth noting that the Springles Ruleset Vocabulary has been designed with the goal to be easy and comfortable to use when writing a ruleset directly without the help of a tool (whose implementation as part of Springles is not envisioned). This justifies certain modelling solutions, such as: (1) the use of different properties to relate a task to its child tasks, so to allow the inference of the type of task and make ruleset specification less verbose; and (2) the use (by specialization) of RDF lists, which are handy in RDF syntaxes but surely not examples of good ontological modelling.

Terms reference

Classes: | ClosureEvalTask | ClosureFixPointTask | ClosureRepeatTask | ClosureSequenceTask | ClosureTask | ClosureTaskList | Parameter | Rule | RuleList | Ruleset |

Properties: | bind | body | closurePlan | condition | default | evalBackward | evalForward | evalOf | fixPointOf | head | macro | name | parameterizedBy | prologue | repeatOf | repeatOver | sequenceOf | transform | triggerOf |

Classes and Properties (full detail)

Classes

Class: spr:ClosureEvalTask

A closure task consisting in the evaluation of zero or more rules, specified using mandatory property spr:evalOf.

Properties include:	spr:evalOf
Sub class of	spr:ClosureTask
Restriction(s):	The property spr:evalOf must be set *at least* 1 time(s)

Class: spr:ClosureFixPointTask

A closure task consisting in the fix-point execution of a child task, specified using mandatory property spr:fixPointOf.

Properties include:	spr:fixPointOf
Sub class of	spr:ClosureTask
Restriction(s):	The property spr:fixPointOf must be set *at least* 1 time(s)

Class: spr:ClosureRepeatTask

A closure task consisting in the repeated execution of a child task, specified by property spr:repeatOf, for each tuple obtained from the evaluation of a `SELECT` query, specified by property spr:repeatOver.

Properties include:	spr:repeatOver spr:repeatOf
Sub class of	spr:ClosureTask
Restriction(s):	The property spr:repeatOver must be set *at least* 1 time(s) The property spr:repeatOf must be set *at least* 1 time(s)

Class: spr:ClosureSequenceTask

A closure task consisting in the sequential execution of zero or more child tasks, specified using mandatory property

spr:sequenceOf.

Properties include: spr:sequenceOf
Sub class of spr:ClosureTask
Restriction(s): The property spr:sequenceOf must be set *at least* 1 time(s)

Class: spr:ClosureTask

A generic task of the closure plan. Explicit definition of variable bindings is supported for any type of task using property spr:bind.

Properties include: spr:bind
Used with: spr:closurePlan spr:fixPointOf spr:repeatOf
Has sub class spr:ClosureSequenceTask spr:ClosureFixPointTask spr:ClosureRepeatTask spr:ClosureEvalTask

Class: spr:ClosureTaskList

A list of closure tasks, defined by specializing and constraining a generic RDF list.

Used with: spr:sequenceOf
Sub class of rdf:List
Restriction(s): The property rdf:rest must be set http://dkm.fbk.eu/springles/ruleset#ClosureTaskList time(s)
The property rdf:first must be set http://dkm.fbk.eu/springles/ruleset#ClosureTask time(s)

Class: spr:Parameter

A ruleset parameter characterized by a name (property spr:name) and a default value (property spr:default), whose value may be supplied by users at repository configuration time.

Properties include: spr:name
Used with: spr:parameterizedBy
Restriction(s): The property spr:name must be set *at least* 1 time(s)
The property spr:default must be set *at least* 1 time(s)

Class: spr:Rule

An inference rule expressed using SPARQL, with mandatory spr:head and spr:body, and optional spr:condition, transformer (property spr:transform) and list of triggered rules (property spr:triggerOf).

Properties include: spr:head spr:transform spr:condition spr:triggerOf spr:body
Used with: spr:evalBackward spr:evalForward
Restriction(s): The property spr:body must be set *exactly* 1 time(s)
The property spr:head must be set *exactly* 1 time(s)

Class: spr:RuleList

A (possibly empty) list of rules, defined by specializing and constraining a generic RDF list.

Used with: spr:triggerOf spr:evalOf
Sub class of rdf:List
Restriction(s): The property rdf:first must be set http://dkm.fbk.eu/springles/ruleset#Rule time(s)
The property rdf:rest must be set http://dkm.fbk.eu/springles/ruleset#RuleList time(s)

Class: spr:Ruleset

A ruleset consists of a set of rules, evaluated either in backward- or forward- chaining. Backward evaluation (currently unsupported) affects rules listed by property spr:evalBackward. Forward evaluation is specified by supplying a closure plan with property spr:closurePlan. A ruleset is optionally associated to a number of ruleset parameters via property spr:parameterizedBy, a 'SPARQL prologue' via property spr:prologue and a number of 'macros' via property spr:macro.

Properties include: spr:evalBackward spr:closurePlan spr:parameterizedBy spr:evalForward spr:macro spr:prologue
Restriction(s): The property spr:closurePlan must be set *at least* 1 time(s)

Properties

Property: spr:bind

Declares a binding for the subject closure task. The value is binding is a string of the form 'variable = SPARQL expression'. The expression is evaluated only on the basis of inherited bindings.

Domain: spr:ClosureTask
Range: xsd:string

Property: spr:body

Specifies the body of the rule. The value is a SPARQL spr:GroupGraphPattern.

Domain: spr:Rule
Range: xsd:string

Property: spr:closurePlan

Specifies the mandatory closure plan associated to the ruleset.

Domain: spr:Ruleset
Range: spr:ClosureTask

Property: spr:condition

Specifies the optional rule condition. The value is a SPARQL boolean expression satisfying the grammar production spr:Constraint. The expression is evaluated only considering inherited variable bindings.

Domain: spr:Rule
Range: xsd:string

Property: spr:default

Specifies the default value of a parameter. Literal values of type xsd:anyURI are injected as IRIs (instead of literals) in SPARQL expressions.

Range: rdfs:Literal

Property: spr:evalBackward

Specifies the rules to be evaluated in backward-chaining. Note that forward- and backward- evaluation of a rule are mutually exclusive.

Domain: spr:Ruleset
Range: spr:Rule

Property: spr:evalForward

Specifies the rules to be evaluated in a forward-chaining mode. If a closure plan is supplied, then this property can be omitted; otherwise, the rules listed here are evaluated according to a basic plan performing fix-point evaluation of all forward-chanining rules.

Domain: spr:Ruleset
Range: spr:Rule

Property: spr:evalOf

Specifies the list of rules to be evaluated in parallel as part of the execution of a spr:ClosureEvalTask.

Domain: spr:ClosureEvalTask
Range: spr:RuleList

Property: spr:fixPointOf

Specifies the sub-task to be executed in a fix-point mode as part of the execution of a spr:ClosureFixPointTask.

Domain: spr:ClosureFixPointTask

Range: spr:ClosureTask

Property: spr:head

Specifies the head of the rule. The value is a SPARQL spr:GroupGraphPattern restricted to triple patterns and the `GRAPH` keyword.

Domain: spr:Rule
Range: xsd:string

Property: spr:macro

Defines a macro using syntax `macro(arg1, ..., argN) = template`. Arguments are optional, in which case parenthesis can be omitted. Arguments can be referenced in the template with the syntax `#arg`. As a safety measure, if an argument `arg` is defined, no SPARQL variable `?arg` can be used in the template. Macros can be used in SPARQL expressions being the objects of spr:head, spr:body, spr:condition and spr:bind properties.

Domain: spr:Ruleset
Range: xsd:string

Property: spr:name

Specifies the name of a ruleset parameter.

Domain: spr:Parameter
Range: xsd:string

Property: spr:parameterizedBy

Specifies the (optional) parameters of the ruleset.

Domain: spr:Ruleset
Range: spr:Parameter

Property: spr:prologue

Specifies the optional SPARQL prologue, consisting of namespace declarations (syntax `PREFIX pref: <uri>>`) and base URI declaration (syntax `BASE <uri>>`).

Domain: spr:Ruleset
Range: xsd:string

Property: spr:repeatOf

Specifies the sub-task to be repeatedly executed as part of the execution of a spr:ClosureRepeatTask.

Domain: spr:ClosureRepeatTask
Range: spr:ClosureTask

Property: spr:repeatOver

Specifies the query whose results control the execution of a child task in a spr:ClosureRepeatTask.

Domain: spr:ClosureRepeatTask
Range: xsd:string

Property: spr:sequenceOf

Specifies the list of child tasks to be executed sequentially as part of the execution of a spr:ClosureSequenceTask.

Domain: spr:ClosureSequenceTask
Range: spr:ClosureTaskList

Property: spr:transform

Specifies the optional transformer associated to the rule. The value is a SPARQL spr:FunctionCall expression whose URI is the transformer URI (either `<springles:builtin function>` or `<java:class name.method name>`) and

arguments are evaluated based only on inherited variable bindings.

Domain: spr:Rule
Range: xsd:string

Property: spr:triggerOf

Specifies the (possibly empty) list of rule(s) that may fire if the subject rule is fired. The property is optional and the order of rules in the list is irrelevant.

Domain: spr:Rule
Range: spr:RuleList