Named Node Expressions
and Reifier Expressions

A Proposal

By Kurt Cagle
kurt.cagle@gmail.com
Copyright 2024-05-22

# Named Node Expression

 A Named Node Expression is a syntactical notation in Turtle that provides a way to provide a “blank” node that nonetheless can be treated as a named node, allowing for a more compact notation.

Named node expressions in RDF allow for a compact and readable representation of complex relationships. The expansion of these expressions into standard RDF triples involves transforming the shorthand notation into multiple triples that convey the same information. Here are the detailed expansion rules for named node expressions when the named node appears in different positions (subject, predicate, or object).

## 1. Named Node in the Object Position

When a named node is in the object position of a triple, the expansion involves creating a series of RDF triples where the named node acts as the subject for additional properties.

**Syntax:**

:subject :predicate [ :namedNode | :property1 :value1; :property2 :value2 ] .

**Expansion:**

:subject :predicate :namedNode .
:namedNode :property1 :value1 .
:namedNode :property2 :value2 .

**Example:**

:person :hasSpouse [ :marriage | :start "2000"; :end "2010" ] .

**Expands to:**

:person :hasSpouse :marriage .
:marriage :start "2000" .
:marriage :end "2010" .

## 2. Named Node in the Predicate Position

When a named node is in the predicate position of a triple, the named node becomes the predicate between the subject and object, with additional properties defined for the named node.

**Syntax:**

:subject [ :namedNode | :property1 :value1; :property2 :value2 ] :object .

**Expansion:**

:subject :namedNode :object .
:namedNode :property1 :value1 .
:namedNode :property2 :value2 .

**Example:**

:liz [ :marriage1 | :start "1965"; :end "1974"; :reason :divorce ] :richard .

**Expands to:**

:liz :marriage1 :richard .
:marriage1 :start "1965" .
:marriage1 :end "1974" .
:marriage1 :reason :divorce .

## 3. Named Node in the Subject Position

When a named node is in the subject position of a triple, the named node acts as the subject, with the properties of the named node expanded into separate triples.

**Syntax:**

[ :namedNode | :property1 :value1; :property2 :value2 ] :predicate :object .

**Expansion:**

:namedNode :predicate :object .
:namedNode :property1 :value1 .
:namedNode :property2 :value2 .

**Example:**

[ :marriage | :start "2000"; :end "2010" ] :involves :person .

**Expands to:**

:marriage :involves :person .
:marriage :start "2000" .
:marriage :end "2010" .

# Summary of Expansion Rules

* **Named Node in Object Position**:
	+ Compact Form: :subject :predicate [ :namedNode | :property1 :value1; :property2 :value2 ] .
	+ Expanded Form:
	+ :subject :predicate :namedNode .
	:namedNode :property1 :value1 .
	:namedNode :property2 :value2 .
* **Named Node in Predicate Position**:
	+ Compact Form: :subject [ :namedNode | :property1 :value1; :property2 :value2 ] :object .
	+ Expanded Form:
	+ :subject :namedNode :object .
	:namedNode :property1 :value1 .
	:namedNode :property2 :value2 .
* **Named Node in Subject Position**:
	+ Compact Form: [ :namedNode | :property1 :value1; :property2 :value2 ] :predicate :object .
	+ Expanded Form:
	+ :namedNode :predicate :object .
	:namedNode :property1 :value1 .
	:namedNode :property2 :value2 .

By following these expansion rules, named node expressions in RDF can be transformed into a set of standard RDF triples, ensuring clarity and consistency in representing complex relationships within the RDF data model.

# Expansion Rules for a Reifier Expression

Reifier expressions in RDF are used to make statements about statements, enabling the attachment of metadata to RDF triples. These expressions follow specific rules for transforming into a set of standard RDF triples that describe the original statement. Here are the detailed expansion rules for reifier expressions:

## Basic Syntax

**Reifier Expression**:

<< ?r | :subject :predicate :object >>

**Expanded Form**:

?r rdf:subject :subject .
?r rdf:predicate :predicate .
?r rdf:object :object .

Where ?r is a variable that gets replaced by a specific URI, effectively creating metadata about the statement :subject :predicate :object.

# Example of Reifier Expression

**Reifier Expression**:

:john :said << :reification1 | :jane rdf:type :Person >>

**Expanded Form**:

:john :said :reification1 .
:reification1 rdf:subject :jane .
:reification1 rdf:predicate rdf:type .
:reification1 rdf:object :Person .

# Detailed Expansion Rules

1. **Identifying Components**:
	* The reifier expression contains a variable or URI (?r) which will be used as the subject of the reification triples.
	* The expression also includes the actual statement components: :subject, :predicate, and :object.
2. **Creating Reification Triples**:
	* For each reifier expression, three RDF triples are generated to describe the original statement.
	* These triples use rdf:subject, rdf:predicate, and rdf:object to indicate the components of the reified statement.
3. **No Assertion of the Original Triple**:
	* The reification does not assert the existence of the original triple :subject :predicate :object. It merely provides metadata about what could be a triple.

## Another Example

**Reifier Expression**:

:reporter :reported << :statement1 | :event rdf:type :Accident >>

**Expanded Form**:

:reporter :reported :statement1 .
:statement1 rdf:subject :event .
:statement1 rdf:predicate rdf:type .
:statement1 rdf:object :Accident .

##

## Summary of Expansion Rules

* **Reifier Expression Syntax**:
	+ << ?r | :subject :predicate :object >>
	+ This represents a statement about a statement, using ?r as the reification node.
* **Transformation into RDF Triples**:
	+ For each reifier expression, generate the following triples:
	+ ?r rdf:subject :subject .
	?r rdf:predicate :predicate .
	?r rdf:object :object .
	+ These triples describe the components of the reified statement.
* **Example**:
	+ **Reifier Expression**: :john :said << :reification1 | :jane rdf:type :Person >>
	+ **Expanded Form**:
	+ :john :said :reification1 .
	:reification1 rdf:subject :jane .
	:reification1 rdf:predicate rdf:type .
	:reification1 rdf:object :Person .

## Important Points

* **Metadata Attachment**:
	+ Reifier expressions are particularly useful for attaching metadata, such as provenance or trust information, to statements.
* **Non-Assertion of Original Statement**:
	+ Reification provides information about a statement without asserting that the statement itself is true or exists in the dataset.

By following these expansion rules, reifier expressions can be transformed into a set of RDF triples that describe the original statement, allowing for rich metadata representation in RDF data models.

# SPARQL ASK Query Using Reifiers

To create a SPARQL ASK query that uses reifiers to check if a given triple exists, we need to look for the reification triples that describe the given triple. Here is how you can structure the query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

ASK WHERE {
 ?reification rdf:subject ?subject ;
 rdf:predicate ?predicate ;
 rdf:object ?object .
}

## Example Query

Suppose we want to check if the triple :jane rdf:type :Person exists using reifiers. We will replace ?subject, ?predicate, and ?object with the specific URIs or literals.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://example.org/>

ASK WHERE {
 ?reification rdf:subject ex:jane ;
 rdf:predicate rdf:type ;
 rdf:object ex:Person .
}

## General Template

Here is a generalized template for the ASK query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

ASK WHERE {
 ?reification rdf:subject <subjectURI> ;
 rdf:predicate <predicateURI> ;
 rdf:object <objectURI> .
}

## Explanation

* **PREFIX**: Declares namespaces for RDF and any other prefixes you may need.
* **ASK WHERE**: The ASK form returns true or false based on the existence of the pattern within the WHERE clause.
* **?reification**: A variable representing the reified statement.
* **rdf:subject, rdf:predicate, rdf:object**: Standard properties used to reify a statement in RDF.

## Usage

Replace <subjectURI>, <predicateURI>, and <objectURI> with the actual URIs of the subject, predicate, and object you want to check. For example:

* <subjectURI>: http://example.org/jane
* <predicateURI>: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
* <objectURI>: http://example.org/Person

This query checks if the reified statement describing the triple exists in the RDF dataset. If it does, the query returns true; otherwise, it returns false.

# SPARQL CONSTRUCT Query

To create a SPARQL CONSTRUCT query that generates a triple from a given reifier, we need to extract the subject, predicate, and object from the reified statement and then construct the corresponding triple.

Here’s how you can structure the query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
 ?subject ?predicate ?object .
}
WHERE {
 ?reification rdf:subject ?subject ;
 rdf:predicate ?predicate ;
 rdf:object ?object .
}

## Example Query

Suppose we want to reconstruct the triple from a specific reifier (e.g., :reification1). We will specify the reifier in the WHERE clause.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://example.org/>

CONSTRUCT {
 ?subject ?predicate ?object .
}
WHERE {
 ex:reification1 rdf:subject ?subject ;
 rdf:predicate ?predicate ;
 rdf:object ?object .
}

## General Template

Here is a generalized template for the CONSTRUCT query:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
 ?subject ?predicate ?object .
}
WHERE {
 <reifierURI> rdf:subject ?subject ;
 rdf:predicate ?predicate ;
 rdf:object ?object .
}

## Explanation

* **PREFIX**: Declares namespaces for RDF and any other prefixes you may need.
* **CONSTRUCT**: Specifies the pattern of triples to be constructed.
* **?subject, ?predicate, ?object**: Variables representing the components of the reified statement.
* **WHERE**: Extracts the subject, predicate, and object from the given reifier.

## Usage

Replace <reifierURI> with the actual URI of the reifier you want to use. For example:

* <reifierURI>: http://example.org/reification1

This query will generate the triple described by the reified statement, reconstructing the original RDF triple from its reified form.