semafora
true semantics

# ObjectLogic Reference Guide

semafora systems GmbH

Wilhelm-Leuschner-Str. 7
64625 Bensheim
Germany
www.semafora-systems.com

# Table of Content

# 1 Taxonomy

## Syntax

```
<concept A> :: <concept B>.
```

## Semantic

```
<concept A> is a subconcept of <concept B>
```

## Sample

```
person[].
man::person.
woman::person.
```

# 2 Query

```
?- ?X[].
```

delivers only a result for explicitly defined concepts (e.g. person[].).

```
?- _concepts(?X).
```

delivers all concepts known in the ontologie.

# 3 Transitivity of subclass relationship

```
?- ?X::?Y.
```

delivers all subconcepts ?Y of concept ?X– not only the asserted ones, but also the subconcepts of the subconcepts and so on

This means that if

C1::C2 and C2::C3 then also C1::C3 holds.

# 4 Query Options

**Definition:**

Query options control the execution and output of a query.

Syntactically query options are provided as annotations to the query. Annotations are written in front of the query with the syntax `@{<queryName>,options[<option1>,<option2>]}`.

- Query without id and options: `?- ?X::?Y.`
- Query with id "subconcepts": `@{subconcepts} ?- ?X::?Y.`
- Query with options "inferOff" and "profile": `@{options[inferOff,profile]} ?- ?X::?Y.`

| Query Option | Description | Example |
|---|---|---|
| sort | Sorts of the result of a query. | `@{q1, options[sort(asc(?X),desc(?Y)]} ?- ?X::?Y.`<br>This example will sort the query results in the ascending order of bindings for ?X and in descending order of bindings for ?Y. It is also possible to write:<br>`@{q1, options[sort(?X,desc(?Y)]} ?- ?X::?Y.`<br>The ?X will be be interpreted as asc(?X), so you can just leave out the asc(..). |
| outorder | Sequence and restriction of the result variables (output ordering).<br>Specifies the sequence of the output variables. | With the facts<br>`Man::Person.`<br>`Woman::Person.`<br>the output of the query<br>`@{q1, options[outorder(?X,?Y)]} ?- ?X::?Y.`<br>will be<br>`Man,Person`<br>`Woman,Person`<br><br>With<br>`@{q1, options[outorder(?Y,?X)]} ?- ?X::?Y.`<br>the output will be<br>`Person,Man`<br>`Person,Woman`<br><br>The "outorder" option can also be used for skipping some variable bindings in the result: with<br>`@{q1, options[outorder(?X)]} ?- ?X::?Y.`<br>the output will be<br>`Man`<br>`Woman` |
| limit | Limits the number of answers. | `@{q1, options[limit(1),outorder(?Y,?X)]} ?- ?X::?Y.`<br>In this example, only one answer will be given. Note that limiting the number of answers does not necessarily mean that the performance will improve. |
| offset | Answer offset. Skips some answers. | `@{q1, options[offset(10),limit(5),sort(?Y,?Y), outorder(?Y,?X)]} ?- ?X::?Y.`<br>In this example five answers will be given, starting with answers then. Using offset only makes sense if the answer order is specified. Otherwise it is not assured that the answers are returned in the same order. |

| | | |
|---|---|---|
| skipSendingAnswers | Skip sending answers.<br>You can execute a query without receiving the results. This can be useful for benchmarking, if the time for sending the answers to the client complicates time measurements. | `@{q1, options[skipSendingAnswers]} ?- ?X::?Y.` |
| trace | Tracing. OntoBroker will log trace information about the query execution | `@{q1, options[trace]} ?- ?X::?Y.` |
| profile | Profiling. Enables the logging of the performance characteristics. | `@{q1, options[profile]} ?- ?X::?Y.` |
| profileAll | Profiling. Enables the logging of the performance characteristics; executes the query with full profiling (including execution plans etc.). | `@{q1, options[profileAll]} ?- ?X::?Y.` |
| inferOff | Turning off inferencing. You may tell OntoBroker not to do inferencing for query answering. This will prohibit the evaluation of all rules and you will only get the given facts matching your query. | `@{q1, options[inferOff]} ?- ?X::?Y.` |
| userRulesOff | Turning off user rules. You may tell the OntoBroker to disable all user rules, i.e. all rules explicitly defined in any module. Internal axiom rules (like e.g. subclass and property hierarchy) are not influenced by this option. | `@{q1, options[userRulesOff]} ?- ?X::?Y.` |
| ignoreImports | Turning off imports. If module A imports module B, all facts of B are included on querying facts from module A. Using the "ignoreImports" query option you can query for facts only stored explicitly in module A. | Module A<br>`:- module A.`<br>`:- importmodule B.`<br><br>`John:person.`<br><br>Module B<br>`:- module B.`<br><br>`Ann:person.`<br><br>`@{q1} ?- ?X:person@A.  // returns John and Ann` |

| | | |
|---|---|---|
| | | `@{q1,options[ignoreImports]} ?- ?X: person@A.  // returns only John` |
| fillNull | Working with null values.<br><br>If a property is not set in ObjectLogic, it is just not there, i.e. querying for the property value returns no result. Under some circumstances, it is useful to get null values in these cases instead. In this case you can use the query option "fillNull". | `John[name->"John"].`<br>Then<br>`@{q1} ?- John[age->?X]. // no results`<br>`@{q1,options[fillNull]} ?- John[age->?X]. // returns ?X = null` |
| explain | Generates explanations for the query. See chapter "Explanations" in the OntoBroker manual for details. | `@{q1,options[explain]} ?- ?X::?Y.` |
| EvaluationMethod | This is an advanced option. To change the evaluation method for a single query, you can use the "EvaluationMethod" option. It takes one argument. The values for the argument are the same as in the OntoConfig.prp. | `@{q1,options[EvaluationMethod(BottomUp)]} ?- ?X::?Y.` |
| BottomUpEvaluator | This is an advanced option. To change the bottom up evaluation method for a single query, you can use the "BottomUpEvaluator" option. It takes one argument. The values for the argument are the same as in the OntoConfig. | `@{q1,options[BottmUpEvaluator(BottomUp)]} ?- ?X::?Y.` |

# 5 Property

**Definition:**

Properties describe a concept more precisely. A property has a domain concept and a range (concept or data type). A property has a minimum cardinality and a maximum cardinality (optionally). Properties are inherited along the taxonomy path (from super to sub concepts)

## Generic syntax

```
<domain>[<propertyname> {min:max} *=> <range>].
```

# 6 Attribute

**Definition:**

An attribute is a property with a basic data type as range.

An attribute describes a characteristic of an individual of a concept.

## Example

```
person[birthDate {1:1} *=> _dateTime].
person[firstname {1:*} *=> _string].
person[weight {1:1} *=> _decimal].
person[nickname {0:*} *=> _string].
```

## Cardinalities

```
person[birthDate {1:1} *=> _dateTime].   Exact 1
person[firstname {1:*} *=> _string].     At least 1
person[nickname {0:*} *=> _string].      Any number
person[nickname *=> _string].            Any number
```

For Expressing attribute values different syntax is available.

```
"hallo"       == "hallo"^^_string
                           == "hallo"^^xsd#string
```

```
1.2           == "1.2"^^_double
                           == "1.2"^^xsd#double
```

```
1             == "1"^^_int
                           == "1"^^xsd#int
```

```
"2001-12-12"^^_date == "2001-12-12"^^xsd#date
```

# 7 Data types

ObjectLogic supports various data types from the XML schema definition language.

Old Flogic syntax: `xsd#string, xsd#int,…`

ObjectLogic abbreviations: `_string, _int,…`

| Content | Data type | More information |
|---|---|---|
| Date and time | _dateTime | http://www.w3.org/2001/XMLSchema#dateTime |
| | _date | http://www.w3.org/2001/XMLSchema#date |
| | _time | http://www.w3.org/2001/XMLSchema#time |
| | _duration | http://www.w3.org/2001/XMLSchema#duration |
| Numbers | _double | http://www.w3.org/2001/XMLSchema#double |
| | _decimal | http://www.w3.org/2001/XMLSchema#decimal |
| | _integer | http://www.w3.org/2001/XMLSchema#integer (any number) |
| | _long | http://www.w3.org/2001/XMLSchema#long |
| | _int | http://www.w3.org/2001/XMLSchema#int (32-bit) |
| Strings and text | _string | http://www.w3.org/2001/XMLSchema#string |
| | _PlainLiteral | http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral |
| | _XMLLiteral | http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral |
| Others | _boolean | http://www.w3.org/2001/XMLSchema#boolean |
| | _iri | http://www.w3.org/2007/rif#iri |

## 7.1 Custom datatypes

OntoBroker supports custom datatypes beside the already implemented sub-set of xsd-types. This support covers the syntax and round-trip save storage of the lexical value and the associated datatype. The semantic interpretation of such a constant's value is not defined, i.e. those datatypes cannot be used within the standard set of built-ins. If operations based on a custom datatype are required it is up to the user to define or extend appropriate built-ins.

For example, the datatype identified by the IRI `http://www.example.org/sense#taste` could be used in ObjectLogic

```
water[tastes->"salty"^^<http://www.example.org/sense#taste>].
```

The lexical value for every custom datatype is stored exactly as provided as string and could be used within built-ins for further processing.

## 7.2 Geographic Coordinates

OntoBroker supports geographic coordinates, represented by latitude and longitude in decimal notation. The format of the new datatype _geo (obl:reserved:geo) in OL syntax is

```
"<latitude>;<longitude>"^^_geo
```

where the semicolon separated <latitude> and <longitude> have to be replaced by the actual values. Those values are stored with a precision of six decimal places. If a value with higher precision is given rounding is applied. An exemplary ontology, showing instances for the cities Karlsruhe, Brisbane and Marrakech with their geographical location might look like

```
karlsruhe:City[location->"49.013964;8.404455"^^_geo].
brisbane:City[location->"-27.336738;153.250909"^^_geo].
marrakech:City[location->"31.625828;-7.989094"^^_geo].
```

Using the built-in `_geoDistance/3` the distance between those locations can be calculated. Please note that this built-in assumes a simplified model of the earth, i.e. a sphere, and therefore will not return the real exact distance. However this approximation should be sufficient for most applications. The arguments are first two geographic coordinates and finally the distance. The coordinates have to be ground, the distance might be. Based on the example from above he query

```
@{options[outorder(?city,?distanceToKarlsruhe),sort(?city)]} ?- karlsruhe[location-
>?lka] and ?city:City[location->?lx] and _geoDistance(?lka,?lx,?distanceToKarlsruhe).
```

will return the distance in kilometer from Karlsruhe to all cities of the ontology:

| ?city | ?distanceToKarlsruhe |
|---|---|
| brisbane | 16185.1272 |
| karlsruhe | 0.0 |
| marakech | 2372.835 |

Obviously the distance from Karlsruhe to itself is zero. All distances are calculated with a precision to 0.1 meter. Additional related built-ins are `_latitude/2` and `_longitude/2` with a geographic coordinate as first argument and the latitude/longitude as second argument.

# 8 Relations

**Definition:**

A relation is a property with a concept as range.

A relation describes the relationship between two concepts.

```
person[hasFather {1:1} *=> man].
person[hasParent {2:2} *=> person].
person[hasSibling {0:*} *=> person].
```

Relations may have special characteristics: symmetry, transitivity and inverseness.

| Characteristic | Example | Description |
|---|---|---|
| Symmetry | `person[relatedTo {0:*,symmetric) *=> person].` | Means if a person A is relatedTo a person B than also person B is relatedTo person A. The reasoner will give back {A, B} and {B,A} as a result. |
| Transitivity | `person[relatedTo {0:*, transitive} *=> person].` | Means if a person A is relatedTo a person B and person B is relatedTo C than also person A is relatedTo person C. The reasoner will give back {A,B}, {B,C} and {A,C} as a result. |
| Inverseness | `person[hasChild {0:*, inverseOf (hasParent)} *=> person].` | Inverseness is always defined between two different relations.<br>The example means if a person A is relatedTo a person B and person B is relatedTo C than also person A is relatedTo person C. The reasoner will give back {A,B}, {B,C} and {A,C} as a result. |
| Property hierarchies | `hasSon<<hasChild.`<br>`hasDaughter<<hasChild.`<br>`hasChild<<relatedTo.` | Sometimes it is possible to reason more generic relations from more special relations. This may be expressed by property hierarchies.<br>In the example stating person A hasSon person B will also lead to the statements person A hasChild person B and person A relatedTo person B. |

## Query for properties

In order to gather relational properties you should query:

```
?- ?X[?R {?P} *=> ()].
```

If you have the schema:

```
Person[hasAge {1:7, symmetric} *=>_duration].
```

then it is possible to execute:

```
?- ?X[?R {?MIN:?MAX, symmetric} *=>?Y].
```

but it is not possible to execute:

```
?- ?X[?R {?MIN:?MAX, ?PROP} *=>?Y].
```

The reason for this behaviour is the way in which relational properties are stored internally in OntoBroker.

# 9 Individuals

**Definition:**

Individuals are classified using the taxonomy defined in the schema. An individual may be instance of one or more concepts

## Syntax

```
<instance A> : <concept B>.
```

## Semantic

```
<instance A> is an instance of <concept B>
```

## Sample

```
Rob:man.
```

# 10 Property values

**Definition:**

As in the schema definition we distinguish properties in attributes and relations. Attributes will have values of basic datatypes as range. Relations will have instance identifiers as range.

## Sample

```
Rob:Man[hasName->"Rob Wroblewski"].
Rob[hasFather->Henryk:Man].
```

# 11 Rules

**Definition:**

Rules are similar to queries but they declare a conclusion in the head. In this way we can derive new facts from given facts. Rules are described by the vocabulary of the schema but work on the facts that are given according to the schema.

## Sample

*What is the definition of an uncle?*

„An uncle is the brother of one of my parents“

In ObjectLogic:

```
?A[hasUncle->?B] :- ?A:Person AND ?A
[hasParent->?P] AND ?P:Person AND ?P
[hasBrother->?B] AND ?B:Mann.
```

# 12 Predicates

**Definition:**

In ObjectLogic, predicates are used in the same way as in predicate logic, e.g. in Datalog. Thus, preserving upward-compatibility from Datalog to ObjectLogic. A predicate symbol followed by one or more terms separated by commas and included in parentheses is called a P-atom to distinguish it from F-atoms. The example below shows some P-atoms. The last P-atom consists solely of a 0-ary predicate symbol. Those are always used without parentheses.

```
owner(car74, paul).
adult(paul).
true.
```

Information expressed by P-atoms can usually also be represented by F-atoms, thus obtaining a more natural style of modeling. For example, the information given in the first two P-atoms could also be expressed as follows:

```
car74[owner->paul].
paul:adult.
```

Note that the expressions in the two examples above are alternative but disjoint representations. They cannot be used in a mixed manner, i.e. a query for owner(X,Y) does not retrieve any results for facts represented in the object-oriented way with F-Atoms.

# 13 Namespaces

**Definition:**

Namespaces are used to distinguish same terms with different semantic. E.g. if two different domains speak about „Jaguar“ (in this case the domain cars and animals) you must be able to use the term „Jaguar“ for both but still be able to distinguish one from the other

## Sample

```
<http://www.carsoftheworld.com#Jaguar>[].
<http://www.animalsoftheworld#Jaguar>[].
```

In ObjectLogic files you may define a default prefix and any number of named prefixes

```
:- default prefix = "http://www.carsoftheworld.com#".
:- prefix bio = "http://www.animalsoftheworld#".

//no leading prefix means it is in the default prefix
Jaguar : Car.

//otherwise you put the prefix name with a ‘#’ in
//front
bio#Jaguar : bio#animal.
```

It is possible to query for namespace and local

```
:- default prefix = "http://www.carsoftheworld.com#".
:- prefix bio = "http://www.animalsoftheworld.com#".

Jaguar:Car.
bio#Jaguar:bio#animal.

?- ?Z:bio#animal
       AND ?Z[_localName->?L]
       AND ?Z[_namespace->?X].
```

# 14 Paths

An expression like
```
o[A->?Y],?Y[B->?Z],?Z[C->?U]
```

can be reduced to

```
o.A.B.C
```

You can insert the shortened expression as a term wherever you want.

# 15 Modules

**Definition:**

Modules help to organize knowledge of different domains. You may define for each statement in which module it is valid. In one obl file only content for one module is allowed.

## Sample

File one

```
:- default prefix = "http://www.carsoftheworld.com#".
:- module = <http://www.carsoftheworld.com/british>.

Jaguar:Car.
```

File two

```
:- default prefix = "http://www.animalsoftheworld.com#".
:- module = <http://www.animalsoftheworld.com/asia>.

Jaguar:animal.
```

Querying modules:

```
?- $module(?M).
?- ?X:?Y@?M.
?- ?X:<http://www.carsoftheworld.com#Car>@?M.
?- ?X:Car@<http://www.carsoftheworld.com/british>.
```

# 16 Expressions

In expressions like ?X is <expression> or ?X = <expression>, you can use some predefined constants and functions.

The operator '=' and 'is' are aliases.

| Constant | Example |
|---|---|
| PI,pi | ?- ?X = PI. // 3.1415.... |
| E,e | ?- ?X = E. // Euler's constant 2.71... |
| RANDOM | ?- ?X = RANDOM. // random number |

| Function | Example |
|---|---|
| + | ?- ?X is 6 + 3. // ?X = 9 |
| - | ?- ?X is 6 - 3. |
| * | ?- ?X is 6 * 3. |
| / | ?- ?X is 6 / 3. |
| mod | ?- ?X is 7 mod 4. |
| abs(_integer),<br>abs(_double) | ?- ?X = abs(-2). |
| max(_integer,_ integer),<br>max(_double, _double) | ?- ?X = max(12, 34). |
| min(_integer,_ integer) | ?- ?X = min(12, 34). |
| round(_double) | ?- ?X = round(4.4). |
| ceil(_double) | ?- ?X = ceil(4.4). |
| floor(_double) | ?- ?X = floor(4.4). |
| rint(_double) | ?- ?X = rint(3.5). |
| tan(_double) | ?- ?X = tan(PI / 4). |
| atan(_double) | ?- ?X = atan(1). |
| sin(_double) | ?- ?X = sin(PI / 4). |
| asin(_double) | ?- ?X = asin(1). |
| cos(_double) | ?- ?X = cos(0). |
| acos(_double) | ?- ?X = acos(1). |
| exp(_double) | ?- ?X = exp(1). |
| log(_double) | ?- ?X = log(E). // natural logarithm, ?X = 1.0 |
| pow(_double,_double) | ?- ?X is pow(4.5, 2). |
| sqrt(_double) | ?- ?X is sqrt(16). |

Of course you can build more complex expressions using brackets.

```
?- ?X = 3 * (4 + sin(pi * ?Y)), ?Y = 0.5. // ?X = 15.0, ?Y = 0.5
```

And you can use the expressions at places where a term is expected.

```
?- ?X[age -> 3 * (4 + sin(pi * 0.5))]. // same as ?- ?X[age -> 15.0].
```

The plus operator also work on strings

```
?- ?X = "a" + "b". // ?X = "ab"
```

| **Comparator** | **Example** |
|---|---|
| < | ?- 3 < 6. |
| <= | ?- 3 <= 6. |
| > | ?- 6 > 3. |
| >= | ?- 6 >= 3. |
| == | ?- 6 == 6.0. |
| != | ?- 3 != 6. |

These comparators do not need any variable or is-statement; the result is simply “true” or “false”.

# 17 Constraints

The syntax of constraints is similar to the syntax of queries. A constraint is considered violated if a result is yielded when the constraint is posed as a query.

Syntax: `!- constraintBody`

Example of constraints:

```
!- ?X:person[worksAt ->  ?C:company, hasJob -> false].
```

This constraint asserts that if a person works at a company, then it is not possible that this person has no job, namely that the value of the attribute “has job” is “false”.

# 18 Annotations

**Hint:**

The two main goals of annotations are:

- to define rule and query Ids
- to specify query options

Rules, queries and constraints can have an Id, which is used in the annotation syntax.

## Syntax

```
@{ID, frame, options[…],…} body.
```

- `ID` represents the rule/query/constraint label,
- `frame` represents the metadata associated with the rule/query/constraint to which the annotation is attached,
- `options`[…] controls the output of a query; the different possible options are:
  - sort(asc(?X),desc(?Y),...): sort the output of a query in the ascending order of bindings for ?X, descending order of bindings for ?Y, etc.
  - outorder(?Y,?X): the output of a query should be a set of tuples (?Y,?X), i.e. the bindings for ?Y go first,
  - maxnumber(NUMBER): specifies that only (at most) the first NUMBER of answers needs to be computed.
  - timeout(NUMBER): stop the computation after this NUMBER of seconds.
- `body` is a rule, a query or a constraint.

## Example of an annotation for a query

```
@{ID1,
  ID1[author -> “John”, date ->"2009-10-12"^^_dateTime],
  options[sort(desc(?Name)), maxnumber(2)]}
?- ?:person[name -> ?Name, age -> ?Age].
```

The query identifier is “ID1”, its author is “John” and it was created on the 12th of October 2009. The results will be sorted in the descending order by the attribute “name” of the class “person” and the returned result will be restricted to the first two results.

## 18.1 Annotations for ObjectLogic Module

The ObjectLogic syntax has been extended to support the metadata for the module itself. You can use this feature to store annotations to the ontology. The ontology annotations must follow the directives directly. They begin with "@module {" and end with "}.". Similarly to the metadata for rules, queries and constraints, these annotations are stored as facts of the $metaatt/4 predicate with the module term as the first argument.

### Example ObjectLogic ontology

```
:- default prefix = "http://yourcompany.com/".
:- prefix dc = "http://purl.org/dc/elements/1.1/".
:- module ontology1.

@module {
   // adding Dublin core metadata to the module
   ontology1[dc#creator->"Martin Weindel", dc#date -> "2010-03-
04T13:35:30"^^_dateTime],
   myobject[foo->bar]
}.
... // rest of ontology
```

# 19 Terms

## 19.1 Constants

Constants are atomic ground terms and they represent the value of a data type.

In general, constants can be written in the following way:

```
"charseq"^^datatype
```

The charseq supports escaping for Unicode and some other special characters.

### 19.1.1 String

| Criterion | | Example |
|---|---|---|
| Usage | For character sequences | |
| Data type | http://www.w3.org/2001/XMLSchema#string | |
| Shortcut | _string | |
| Syntax alternatives | double-quoted strings | "charseq" |
| | triple double-quoted strings | """charseq""" |
| | string representation with data type | "charseq"^^_string |

Double quoted strings without a data type (i.e. ^^datatype) are automatically interpreted as constants of the XML schema data type http://www.w3.org/2001/XMLSchema#string.

ObjectLogic supports the Java-like escape sequences starting with the character '\' for Unicode characters and some special characters. Characters in the range ASCII 0x00 to 0x1f are not allowed as characters in quoted string but can be entered as escape sequence. New lines are directly allowed in double quoted strings.

| Escape sequence | Description |
|---|---|
| \u1a3b | UTF-16 character (here code point 1A3B)<br>A Unicode escape sequence consists of<br>1. a backslash character '\'<br>2. a 'u'<br>3. four hexadecimal digits |
| \\ | backslash character \ (ASCII 0x5c) |
| \" | double quote character " (ASCII 0x22) |
| \' | single quote character ' (ASCII 0x27) |
| \n | line feed character (ASCII 0x10) |
| \r | carriage return character (ASCII 0x13) |
| \t | horizontal tab character (ASCII 0x09) |
| \f | form feed character (ASCII 0x0c) |

Examples:

```
       "foo"  ° "foo"^^_string
                     ° "foo"^^<http://www.w3.org/2001/XMLSchema#string>
       """multi-line
second line
third"""                    ° "multi-line\nsecond line\nthird"
       "M\u00fcller"        ° "Müller"^^_string
```

### 19.1.2 IRI

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2007/rif#iri | |
| Shortcut | _iri | |
| Syntax alternatives | full qualified IRI reference | '<' IRI '>' |
| | local part identifier (implicitly carries default namespace) | identifier |
| | single quoted local part (implicitly carries default namespace) | '\'' charseq '\'' |
| | prefix plus local part | (identifier \| '\'' charseq '\'') '#'<br>(identifier \| '\'' charseq '\'') |
| | IRI in ObjectLogic reserved namespace | '_'(identifier \| '\'' charseq '\'') |
| | IRI in ObjectLogic internal namespace | '$' (identifier \| '\'' charseq '\'' |
| | string representation with data type | "charseq"^^_iri |

The following examples assume that these namespace prefixes are defined:

```
:- default prefix = "http://foo#".
:- prefix a = "urn:ietf:rfc:".
:- prefix 'i!' = "urn:isbn:".
```

Examples:

```
<http://www.acme.com/project1#ontology1>
       ° "http://www.acme.com/project1#ontology1"_iri

Müller                  ° <http://foo#Müller>
'Jörg Müller'           ° <http://foo#Jörg%20Müller>
a#'2648'                ° <urn:ietf:rfc:2648>
a#2648 (invalid syntax)
'i!'#'0451450523'       ° <urn:isbn:0451450523>
_bar                    ° <obl:reserved:bar>
$bar                    ° <obl:intern:bar>
```

### 19.1.3 Double

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2001/XMLSchema#double | |
| Shortcut | _double | |
| Syntax alternatives | number containing decimal point | ['+'\|'-']? digit+ '.' digit+ Exponent? FloatSuffix?<br>['+'\|'-']? '.' digit+ Exponent? FloatSuffix?<br>Exponent := ['E'\|'e'] ('+'\|'-')? Digit+<br>FloatSuffix := ['d'\|'D'\|'f'\|'F'] |
| | number containing exponent | ['+'\|'-']? digit+ Exponent FloatSuffix? |
| | number with float type suffix | ['+'\|'-']? digit+ FloatSuffix |
| | string representation with data type | "charseq"^^_double |

All numbers containing a decimal point, or an exponent or a float type suffix are constants of the _double data type.

Examples:
```
1.2            ° "1.2"^^_double
               ° "1.2"^^<http://www.w3.org/2001/XMLSchema#double>
.12            ° "0.12"^^_double
-0.12          ° "-0.12"^^_double
12e-34         ° "1.2e-33"^^_double
13d            ° 13.0  ° "13.0"^^_double
```

### 19.1.4 Int

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2001/XMLSchema#int | |
| Shortcut | _int | |
| Syntax alternatives | numbers between -231 and 231-1 without decimal point | digit+ |
| | string representation with data type | "charseq"^^_int |

All constants of `type _int` are the same term if it was defined with the same value but with either data type `_int`, `_long`, `_integer` or `_decimal`.

Examples:
```
1234567        ° "1234567"^^_int
               ° "1234567"^^<http://www.w3.org/2001/XMLSchema#int>
-98765         ° "-98765"^^_int
```

### 19.1.5 Long

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2001/XMLSchema#long | |
| Shortcut | _long | |
| Syntax alternatives | numbers between -263 and 263-1 without decimal point | digit+ |
| | string representation with data type | "charseq"^^_long |

All constants of type _long are the same term if it was defined with the same value but with either data type _long, _integer or _decimal. If the value is between -231 and 231-1, it is also the same as the constant with the same value and data type _int.

Examples:

```
123456789012345       ° "123456789012345"^^_long
                      ° "123456789012345"^^<http://www.w3.org/2001/XMLSchema#long>
-123456789012345      ° "-123456789012345"^^_long
1234567               ° "1234567"^^_long ° "1234567"^^_int
```

### 19.1.6 Integer

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2001/XMLSchema#integer | |
| Shortcut | _integer | |
| Syntax alternatives | integer numbers less than -263 or greater than 263-1 without decimal point | digit+ |
| | string representation with data type | "charseq"^^_integer |

Examples:

```
123456789012345678901234567890
        ° "123456789012345678901234567890"^^_integer
        ° "123456789012345678901234567890"^^<http://www.w3.org/2001/XMLSchema#integer>
```

### 19.1.7 Decimal

| Criterion | | Example |
|---|---|---|
| Data type | http://www.w3.org/2001/XMLSchema#decimal | |
| Shortcut | _decimal | |
| Syntax alternatives | string representation with data type | "charseq"^^_decimal |

If the decimal value is an integer, the term is the same as if it was defined with an integer data type. Constants of data type double and of data type decimal do not unify.

Examples:

```
"12.3456"^^_decimal  ° "12.3456"^^<http://www.w3.org/2001/XMLSchema#decimal>
12.3456              ° "12.3456"^^_double

"123"^^_decimal
       ° "123"^^_integer
       ° "123"^^_long
       ° "123"^^_int
       ° 123
```

## 19.1.8 Date

| Criterion | | Example |
|---|---|---|
| Usage | To specify a date | |
| Data type | http://www.w3.org/2001/XMLSchema#date | |
| Shortcut | _date, _d | |
| Syntax alternatives | lexical representation according the XMLSchema data type | "YYYY-MM-DD"^^_date<br>"YYYY-MM-DDZ"^^_date<br>"YYYY-MM-DDShh:mm"^^_date<br>"YYYY-MM-DD"^^_d<br>"YYYY-MM-DDZ"^^_d<br>"YYYY-MM-DDShh:mm"^^_d<br><br>YYYY indicates the year, MM indicates the month (1-12), DD indicates the day (1-31) |

To specify a date in UTC time is marked with the suffix 'Z'

- Z character 'Z' indicates that this date is in UTC time

Alternatively, to specify the timezone you can add an offset from the UTC time by adding a positive or negative time behind the date.

- S +/- sign of the time zone offset relative to UTC
- hh hours of time zone offset
- mm minutes of time zone offset (0-59)

Examples:

```
"2009-03-31"^^_date
"1999-12-24Z"^^_date
"2009-03-31-06:00"^^_date
```

### 19.1.9 Time

| Criterion | | Example |
|---|---|---|
| Usage | To specify a time | |
| Data type | http://www.w3.org/2001/XMLSchema#time | |
| Shortcut | _time, _t | |
| Syntax alternatives | lexical representation according the XMLSchema data type | "hh:mm:ss.f"^^_time<br>"hh:mm:ss.fZ"^^_time<br>"hh:mm:ss.fShh:mm"^^_time<br>"hh:mm:ss.f"^^_t<br>"hh:mm:ss.fZ"^^_t<br>"hh:mm:ss.fShh:mm"^^_t<br><br>hh indicates the hour (0-23), mm indicates the minute (0-59), ss indicates the second (0-59), f is an optional part to indicate the fractions of a second. |

To specify a date in UTC time is marked with the suffix 'Z'

- Z  character 'Z' indicates that this date is in UTC time

Alternatively, to specify the timezone you can add an offset from the UTC time by adding a positive or negative time behind the date

- S  +/- sign of the time zone offset relative to UTC
- hh  hours of time zone offset
- mm  minutes of time zone offset (0-59)

Examples:

```
"13:57:00"^^_time
"13:57:01.124"^^_time
"23:15:00Z"^^_time
"00:00:00.001+09:00"^^_time
```

### 19.1.10 DateTime

| Criterion | | Example |
|---|---|---|
| Usage | To specify a date and a time | |
| Data type | http://www.w3.org/2001/XMLSchema#dateTime | |
| Shortcut | _datetime, _dt | |
| Syntax alternatives | lexical representation according the XMLSchema data type | "YYYY-MM-DDThh:mm:ss.f"^^_dateTime<br>"YYYY-MM-DDThh:mm:ss.fZ"^^_dateTime<br>"YYYY-MM-DDThh:mm:ss.fShh:mm"^^_dateTime<br>"YYYY-MM-DDThh:mm:ss.f"^^_dt<br>"YYYY-MM-DDThh:mm:ss.fZ"^^_dt<br>"YYYY-MM-DDThh:mm:ss.fShh:mm"^^_dt<br><br>YYYY indicates the year, MM indicates the month (1-12), DD indicates the day (1-31), the character 'T' indicates the start of the required time section, hh indicates the hour (0-23), mm indicates the minute (0-59), ss indicates the second (0-59), f is an optional part to indicate the fractions of a second. |

To specify a date in UTC time is marked with the suffix 'Z'

- Z character 'Z' indicates that this date is in UTC time

Alternatively, to specify the timezone you can add an offset from the UTC time by adding a positive or negative time behind the date

- S +/- sign of the time zone offset relative to UTC
- hh hours of time zone offset
- mm minutes of time zone offset (0-59)

Examples:

```
"2009-03-31T13:57:00"^^_dateTime
"2009-03-31T13:57:01.124"^^_dt
"2009-03-31T23:15:00Z"^^_dt
"2000-01-01T00:00:00.001+09:00"^^_dt
```

### 19.1.11 Duration

| Criterion | | Example |
|---|---|---|
| Usage | To specify a time interval | |
| Data type | [http://www.w3.org/2001/XMLSchema#duration](http://www.w3.org/2001/XMLSchema#duration) | |
| Shortcut | _duration | |
| Syntax alternatives | lexical representation according the XMLSchema data type | "sPnYnMnDTnHnMn.fS "^^_duration<br><br>• s +/- sign of time interval (optional)<br>• P the character 'P' indicates the period (required)<br>• nY indicates the number of years (optional)<br>• nM indicates the number of months (optional)<br>• nD indicates the number of days (optional)<br>• nH indicates the number of hours (optional)<br>• nM indicates the number of minutes (optional)<br>• n.fS indicates the number of seconds (optional) with optional fractions of seconds |

Examples:

```
"-P1Y"^^_duration
"PT1004199059S"^^_duration
"P1DT2S"^^_duration
"P1Y2M3DT5H20M30.123S"^^_duration
```

The following values are invalid: 1Y (leading P is missing), P1S (T separator is missing), P-1Y (all parts must be positive), P1M2Y (parts order is significant and Y must precede M), or P1Y-1M (all parts must be positive).

### 19.1.12 Boolean

| Criterion | | Example |
|---|---|---|
| Usage | To specify a true or false value | |
| Data type | http://www.w3.org/2001/XMLSchema#boolean | |
| Shortcut | _boolean | |
| Syntax alternatives | lexical representation according the XMLSchema data type | "sPnYnMnDTnHnMn.fS "^^_duration<br><br>• s +/- sign of time interval (optional)<br>• P the character 'P' indicates the period (required)<br>• nY indicates the number of years (optional)<br>• nM indicates the number of months (optional)<br>• nD indicates the number of days (optional)<br>• nH indicates the number of hours (optional)<br>• nM indicates the number of minutes (optional)<br>• n.fS indicates the number of seconds (optional) with optional fractions of seconds |
| | Objectlogic keywords | true º "true"^^_boolean<br>false º "false"^^_boolean |

### 19.1.13 Plain Literal

| Criterion | | Example |
|---|---|---|
| Usage | To specify a text string with an attached language tag | |
| Data type | http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral | |
| Shortcut | _PlainLiteral, _text | |
| Syntax alternatives | lexical representation according RIF-DTB syntax | "str@lang"^^_PlainLiteral<br>• "abc"{de} == "abc@de"^^_PlainLiteral<br>• str character sequence (escaping conventions are same as for _string)<br>• @ the character '@' separates value and language tag<br>• lang language tag (can be empty) |

Examples:

```
"Wein@de"^^_PlainLiteral
"vine@en"^^_PlainLiteral
"vin@fr"^^_PlainLiteral
"UNO@"^^_PlainLiteral
```

See also: `http://www.w3.org/2005/rules/wiki/DTB#ref-rdf-PlainLiteral`

### 19.1.14 XMLLiteral

| Criterion | | Example |
|---|---|---|
| Usage | To specify well-formed fragment of XML text | |
| Data type | http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral | |
| Shortcut | _XMLLiteral | |
| Syntax alternatives | XML text with data type | "xmltext"^^_XMLLiteral |

Examples:

```
"<foo>bar</foo> "^^_XMLLiteral
"<order id=\"12345\"/>"^^_XMLLiteral
```

## 19.2 Variables

All variables are denoted as ?var

- ? a variable always starts with the character '?'
- var variable name (can contain letters, digits und most unicode characters) - See grammar in appendix for details.

The variable ? denotes an anonymous variable. Every occurance denotes a new variable that does not occur anywhere else.

Example:

```
?X
?y1
?1
?aPerson
?Übung
```

## 19.3 Functions

Functions are complex terms that consist of a function symbols (which is a constant as defined above) and a list of one or more terms (enclosed in parenthesis) representing the arguments.

Example:

```
f(?X)
<http://racer.com#maximumSpeed>(germany, autobahn)
1.2(?X,?Y)
"c"(1,2,3)
```

## 19.4 Lists

A special kind of terms are lists. In ObjectLogic lists of terms can be represented as in Prolog. A list containing the constants a to e looks like this:

```
[a,b,c,d,e]
```

Internally a list is represented by recursively nesting the binary function symbol $l (º <obl:intern:l>). Its first argument represents the first element of the list and its second argument represents the rest of the list (i.e. head and tail in Prolog-speak, or car and cdr in Lisp-speak).

The example list presented above has the internal represenation:

```
$l(a, $l(b, $l(c, $l(d, $l(e, []))))))
```

Note the empty list to represent the end of the list. Due to the canonical mapping even open lists with no fixed length can be represented, e.g.

```
[a, b, c, d | ?Tail]
```

The variable `?Tail` represents the currently not bound list, following the fourth element of this list. Note the "|"-symbol after `d`. This symbol separates the remainder of the list of the lists firsts element.

Example:

```
[]
[1,2,3]
[1,[2,3],4]
[1,2|?T]
```

## 19.5 map data type

### Mapping between Java objects and ObjectLogic

Example:
BLZ Service of the Deutsche Bundesbank, see
http://www.thomas-bayer.com/axis2/services/BLZService?wsdl
Generating the Java client for this web service results in (simplified here):

Java

```
interface BLZService {
   Details getDetails(String blz);
}
class Details {
   String bezeichnung;
   String bic;
   String ort;
   String plz;
}
```

A web service call from ObjectLogic could look like this:

```
?- _callWebservice("http://bundesbank/services/BLZService",
       "getDetails", [blz->"66010075"],?Result).
```

Internally the input parameters, here the single parameter "blz" is extracted from the input map and the web service is called via the generated Java client. The result is returned in a Details object. This object is mapped into a map constant term:

```
?Result = [bezeichnung->"Postbank Karlsruhe",
  bic->"PBNKDEFF",ort->"Karlsruhe",plz->"76127"]
```

## Mapping between XML and ObjectLogic

Using both maps and lists, it would become possible to map XML directly into an ObjectLogic constant term without loosing the hierarchical structure and sequence order. This would be very similar to the mapping between XML and JSON.
Example:
XML

```
    <animals>
       <dog id="1">
             <name>Rufus</name>
             <breed>labrador</breed>
        </dog>
        <dog id="2">
             <name>Marty</name>
             <breed>whippet</breed>
        </dog>
        <cat name="Matilda"/>
   </animals>
```

ObjectLogic

```
?X = [animals->[
       [dog->[
              [id->1,name->"Rufus",breed->"labrador"],
              [id->2,name->"Marty",breed->"whippet"]
        ],
        cat->[
              [name->"Matilda"]
        ]
       ]
    ]
```

## Options parameter for built-ins

Example:

```
?- _queryIndex(module,[return(type),return(title),includeAll,stringMetric("Jaro")],
  "searchString",0,10,?OBJ,?TC,?SCORE,?ORDER,?OPT).
```

Using the map data type this becomes more readable to

```
?- _queryIndex(module,
 [return->{type,title},includeAll->true,stringMetric-> "Jaro")],
 "searchString",0,10,?OBJ,?TC,?SCORE,?ORDER,?OPT).
```

## Result parameter for built-ins with variable content structure

Sometime built-ins need to return variable result sets depending on the options. This has already been needed in the above web service example. Another example is here also the _queryIndex/10 built-in, where the ?OPT variable returns variable a list of functions, which is hard to pass directly in ObjectLogic.
For the above example, the ?OPT variable currently returns things like

```
?OPT = [type("c"),title("Reparaturanleitung")]
```

Using the map data type, this would be instead

```
?OPT = [type->"c",title->"Reparaturanleitung"]
```

## Specification Details

The syntax is just an overloading of the list syntax. The square brackets would contain key/value pairs instead of terms. Mixing terms and key/value pairs is forbidden, also the head/tail syntax used for lists.

## Assignments

| ObjectLogic syntax | Description |
|---|---|
| ?X=[] | Empty list (special constant) |
| ?X=[->] | Empty map (special constant) |
| ?X=[1,2,3] | list |
| ?X=[1\|?Y] | List head / tail |
| ?X=[a->1,b->2] | Map |
| ?X=[a->{1,2,3},b->[1,2]]<br>equivalent to<br>?X=[a->1,a->2,a->3,b->[1,2]] | Multi-valued map. Note the difference between the values for a and b.<br>a is multi-valued, but b has a list as value. |
| ?X = [1,a->b] // invalid | Mixing of list terms and key value pairs is not allowed. ObjectLogic compiler must throw exception |

## Facts

| ObjectLogic syntax | Description |
|---|---|
| p([a->1,b->2]). | Predicate p/1 with a map term as first argument |

## Rules , Queries, Built-ins

| ObjectLogic syntax | Description |
|---|---|
| p([?X->?Y]) :- r(?X,?Y). | map in rule head |
| p(?Z) :- ?Z = [?X->?Y], r(?X,?Y). | same as above |
| ?- ?M = collectmap { ?X,?Y \| r(?X,?Y) }.<br><br>translates into:<br>?- _aggr_collectmap([],[?X,?Y], ?X, ?Y, ?M), r(?X,?Y)). | Aggregation collectmap to collect key/value pairs in one map |
| ?- [a->{1,2},b->3][_memberAt(?X)->?Y].<br><br>returns:<br>?X = a, ?Y = 1<br>?X = a, ?Y = 2<br>?X = b, ?Y = 3 | Extended built-in _memberAt/3 to extract keys and/or values from a map |
| ?- [a->{1,2},b->3,c->4][_intersection([a->1, c->5])->?X].<br><br>returns:<br>?X = [a->1] | Built-in _intersection/3 to create intersection of two maps |
| ?- [a->{1,2},b->3,c->4][_union([a->1, c->5])->?X].<br><br>returns:<br>?X = [a->{1,2},b->3,c->{4,5}] | Built-in _union/3 to create union of two maps |
| ?- [a->{1,2},b->3,c->4] [_difference([a->1, c->5])->?X].<br><br>returns: | Built-in _difference/2 to remove all key/ value pairs of argument1 from argurment 0 |

| **ObjectLogic syntax** | **Description** |
|---|---|
| ?X = [a->2,b->3,c->4] | |
| ?- [b->1,a->4,a->2][_normalize->?X].<br><br>returns:<br>?X = [a->{2,4}, b->1] | Built-in _normalize/2 to bring map in internal order |
| ?- _map[_toType(a,2)->?X].<br><br>returns:<br>?X = [a->2] | Extended built-in _toType/4 to generate singleton map |
| ?- ?M = [a->[b->[1,2], c->3], d->4], _memberByPath(?M,_path(a, b),?V).<br><br>returns<br>?V = [1,2] | Built-in _memberByPath/3 to filter with xpath expression |
| ?- ?M = [a->[b->[1,2], c->3], d->4], _map2table(?M,?P,?I,?V).<br><br>returns<br>P I V<br>---------------<br>"a.b" 0 1<br>"a.b" 1 2<br>"a.c" 0 3<br>"d" 0 4<br><br>?- ?M = [a->[b->{1,2}, c->3], d->4], _map2table(?M,?P,?I,?V).<br><br>returns<br>P I V<br>---------------<br>"a.b" 0 1<br>"a.b" 1 2<br>"a.c" 0 3<br>"d" 0 4 | Built-in _map2table/4 to convert a map to flat table. Note that list values are converted to flat table too. |
| ?- [a->1,b->2] == [b->2,a->1].<br><br>returns:<br>true | _equals/2 should be extended to compare maps independently from the order |

**Side remark:**
Mapping ObjectLogic instances to and from maps: Interestingly, you can also extract all attribute values of an ObjectLogic instance into a map. It “strips” off the instance and leaves the blank attribute map.

Example 1 (flat):

```
juergen[name->"Jürgen", age->28].
?- ?M = collectmap { ?Z | ?Z=[?X,?Y], juergen[?X->?Y] }.
```

results in:

```
?M = [name->"Jürgen", age->28]
```

Example 2 (hierarchical):

```
juergen:Person[name->"Jürgen", age->28, address->ja].
ja:Address[street->"An der RaumFabrik", ort->"Durlach"].
attribute(?P,?X,?Y) :- ?P:Person[?X->?Y] AND NOT (EXIST ?C ?Y:?C).
attribute(?P,?X,?Y) :- ?P:Person[?X->?Z] AND ?Z:?C, ?Y = collectmap {?X1,?Y1 [?Z]| ?Z
[?X1->?Y1]}.
?- ?P:Person, ?M = collectmap {?X,?Y [?P] | attribute(?P,?X,?Y)}.
```

results in:

```
?P = juergen
?M = [address->[street->"An der RaumFabrik",ort->"Durlach"],name->"Jürgen",age->28]
```

## Implementation

Maps are stored as functions internally. The function symbol is $m and the arguments are a sequence of key value pairs. This implies that the key,value pairs are ordered and that a key can have multiple values. Therefore [a->b,c->d] does not unify with [c->d,a->b]. If you need to unify maps, use the normalize built-in to bring the key/value pairs in a the same order.

| ObjectLogic term | Description | Internal Representation |
|---|---|---|
| [] | Empty list (special constant) | [] |
| [->] | Empty map (special constant) | [->] |
| [1,2,3] | list | $l(1,$l(2,$l(3,[]))) |
| [1\|?Y] | List head / tail | $l(1,?Y) |
| [a->1,b->2] | Map | $m(a,1,b,2) |
| [a->{1,2,3},b->[1,2]] | Multi-valued map. Note the difference between the values for a and b. a is multi-valued, but b has a list as value. | $m(a,1,a,2,a,3,b,$l(1,$l(2,[]))) |

## 19.6 Null value

The null value is a type-less constant which is used to denote the NULL value of relational databases.

Example:

```
?- ?X = null.
```

## 19.7 Blank nodes

Blank nodes denote anonymous resources. Sometimes they are also named anonymous OIDs, . Without digits every occurance means a completely new identifier. With the numbered syntax _#1,_#2,… occurances of the same blank node (e.g. _#2) means the same identifier, but only within a clause. I.e. if _#2 occurs in two facts, they are different identifiers.

Syntax: `_# digit*`

Example:

```
q(1,_#).
q(2,_#).
?- q(?X,?Y). // returns two different blank nodes
```

```
tmp(1).
tmp(2).
?- ?X = p(?Y,_#,_#1,_#,_#1,_#2), tmp(?Y). // returns same blank nodes for both
results
```

In rare cases if you need to reference an existing blank nodes use the following syntax:

`_# '<UUID>'`

Example:

```
?- ?X = _#'n6c5245b2-de51-445a-9b14-5b3dc23ac239-000d'.
```

## 19.8 Current module

The term @_ is a short cut for the current module.

Example:

```
?- ?X = @_.
```

A more useful example is the usage within a rule using a built-in with an module argument.

```
:- module m1.

// define predicate to access rule text for all user rules of this module
@{ruleText} ruleText(?ID,?TXT) :- _ruleByID(@_, ?ID, ?TXT).
```