OntoBroker 6.3

semafora systems GmbH

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

Table of Content

About this document

There is an Online Help and a user manual available. The Online Help can be accessed via the Help / Help Contents menu. The user manual is available as a PDF on the Web page http://www.otc.semafora-systems.com/download and can be downloaded free of charge.

The documentation describes the software on a Windows operating system with the browser Mozilla Firefox. You can access the information using the table of contents, the index and the full text search. If you cannot find the answer to your question in the helpsystem, visit free training center at http://otc.semafora-systems.com, area "support", to get in touch with the support team.

1 General Information

1.1 Naming the Screen Elements

Screen-Element	Example
Tabs	
Navigation Tree	
Text Field (For Entering Text)	
Selection Field	 (Opens by clicking on the black triangle)
Radio Button	 (inactive)  (active)
Check Box	 (inactive)  (active)
Button	

1.2 Typographic Conventions

Use	Typographic Convention	Example
Definition or Note	Outlined in gray	**Definition:** "The RIF Framework for Logic Dialects (RIF-FLD) is a formalism for specifying all logic dialects of RIF, ..."
Instruction Prerequisite	Prerequisite: ✓ [Instruction prerequisite]	Prerequisite: ✓ You have the rights to read and to write to your current home directory and the installation directory.
Instruction	Numbered	1. Click on **File** -> **Import...**. 2. Click on **Ontology from Outlook**. 3. Click on **Next**.
Instruction Annotation	Intended	Outlook Import does not establish a connection to its source data during runtime but imports all of the data from Outlook during the import process.
Instruction Result	Not intended	A form appears.
Code Example	Courier New, grey background	`OntologyLanguage = RDF`
Term (Tab, Button or Link) on the user interface	Bold	**OK**
Technical term, e.g. command, web address, path or file name with file ending	Courier New font	`www.ontoprise.de`

1.3 Installation

Installing OntoBroker under Microsoft Windows

The installation of OntoBroker is fast and easy. After starting the installation file, the installation process is initiated. In the following, you have to confirm the prompts in the windows that appear. Before installing OntoBroker, you have to check whether your computer meets the system requirements for the recommended configuration. Before installing OntoBroker, close all of the programs and applications on the computer. The installation wizard will guide you through the further installation process. During the installation, you will be asked for the KeyFile which should be stored in the OntoBroker installation folder.

Note:
Keyfiles for different versions can be put into the same directory ONTOPRISE_LICENSE_HOME.

Installing OntoBroker under Linux

Before installing OntoBroker make sure you have an appropriate Java runtime environment installed (OntoBroker requires Java 1.6). Then:

- Unzip the OntoBroker distribution into a folder (e.g. /usr/shared/ontoprise/ontobroker),
- Copy the license key file (ob_prof.key.xml) into the same folder.

The setup process is now finished. You should be able to start OntoBroker using the commandline utilities.

Note:
The installer automatically installs the correct runtime dependending on the target machine's bit-version.

System Requirements

Application	System Requirements
Permitted operating systems	• Windows XP Professional • Windows 7 • Windows Vista • Windows Server 2003 • Linux (tested on SUSE, Debian and Ubuntu)
Java platforms	Sun/OpenJDK Java 6 and later
Hardware	• Intel or AMD processors, 32-bit or 64-bit • Minimum: 1 core, 2 GHz and better, 1 GB RAM, 200 MB free hard disk space • Recommended: Quad core, 2.4 GHz and better, 4 GB RAM, 400 MB free hard disk space

Important Notes

About the Key File
When installing the keyfile, during the installation process you must make sure that the keyfile has the original name given by semafora and the original extension »`*.key.xml`«. Otherwise the keyfile has to be renamed and copied manually to the installation directory.

About Windows Vista/Windows 7
OntoBroker is fully compatible with Windows Vista and later versions. This means that OntoBroker will not try to write into "Program Files" etc. We recommend that you install OntoBroker to the suggested default location of the operating system.

Installing JDBC-Driver

For database schema imports, an appropriate JDBC Driver for the database may be required. While OntoBroker supports Oracle, MSSQL Server and DB2 out of the box, other drivers, such as MySQL may not be redistributed. If you have purchased a valid license for the driver, you may configure OntoBroker in the following way:

Copy the database driver to [OntoBroker]/lib (e.g. mysql-connector-java-5.1.5-bin.jar)

Afterwards write the settings to OntoConfig.prp:

```
ExternalDB.OtherDatabase.Driver = com.mysql.jdbc.Driver
ExternalDB.OtherDatabase.ConnectionURL = jdbc:mysql://$host$/$name$
```

Installing MySQL Drivers

1. Change `OntoConfig.prp`.
   Add the following two lines to the configuration file:

```
ExternalDB.OtherDatabase.Driver = com.mysql.jdbc.Driver
ExternalDB.OtherDatabase.ConnectionURL = jdbc:mysql://$host$/$name$
```

2. Get the MySQL JDBC driver.
   Download a recent version of the JDBC driver (e.g. mysql-connector-java-5.1.10-bin.jar).
3. Convert to OSGi Bundle.
   Download `bnd.jar` (Bnd can be downloaded here: http://www.aqute.biz/Code/Download).
   a) Run from the command line: `java -jar bnd.jar wrap mysql-connector-java-5.1.10-bin.jar`
      As output you get `mysql-connector-java-5.1.10-bin.bar`
   b) Rename `.mysql-connector-java-5.1.10-bin.bar` to `mysql-connector-java-5.1.10-bin.jar`
   For more information, see:
   jonas.ow2.org/JONAS_5_2_0_M1/doc/doc-en/pdf/howto_install_jdbc_driver.pdf
4. Copy to Extensions Directory
   Copy the OSGi bundle, generated in the last step to the `extensions` directory of your OntoBroker or OntoStudio installation.
   `extensions/mysql-connector-java-5.1.10-bin.jar`
5. Start OntoBroker
   Finally start OntoBroker or OntoStudio.

1.4 Ontology Languages

OntoBroker supports multiple ontology languages:

- ObjectLogic 64
- RDF/RDFS 63
- OWL 63
- RIF 75

OntoBroker provides native support for RDF/RDFS and OWL. This makes OntoBroker applicable for a wide range of semantic applications and scenarios. It is possible to use OntoBroker as a high-performance RDF triple store (or quad store), for typical OWL reasoning tasks (e.g. consistency checks) for conjunctive queries against an OWL ontology, for high performance ObjectLogic reasoning for evaluating and debugging rules. It is possible to use different storage systems with all ontology languages. This means that you can use the persistent storage H2 for storing OWL axioms, RDF triples and ObjectLogic facts and rules.

You can use SPARQL (W3C standard for RDF query languages) for executing queries in combination with all other ontology languages (see the "Query Languages" chapter for further details). OntoBroker also provides a powerful Java-based API which can be used to open, modify and store ontologies, convert ontology languages (e.g. from OWL to ObjectLogic) and to execute queries against an ontology. This API was designed to be independent from the ontology language. This makes it easy to access OntoBroker from your own applications and to write applications which can deal with all ontology languages.

The ontology language can be set in the OntoConfig.prp configuration file of the OntoBroker server:

```
OntologyLanguage = ObjectLogic
```

or

```
OntologyLanguage = RDF
```

or

```
OntologyLanguage = OWL
```

2 Using Ontobroker as RDF Triple Store

2.1 Introduction

OntoBroker RDF is an RDF triple store, query and inferencing engine. It is well integrated into the general OntoBroker suite, allowing close interaction between the other supported knowledge representation formats OWL and ObjectLogic. Staying close to the official Semantic Web standards like RDF, RDF Schema, and SPARQL is one of the primary design goals. However, the implementation on a horn logic-based model imposes some restrictions in standard compliance -- and some unique features too. This document will give a pragmatic overview of OntoBroker RDF, focusing on RDF-related options and features. For more general issues, see the provided locations in the main OntoBroker documentation.

2.2 New and Noteworthy

A quick overview of the major changes in OntoBroker RDF 6.2:

- Support for ASK, DESCRIBE, and CONSTRUCT query forms
- SPARQL Update implementation
- SPARQL endpoint for queries and updates (including JSON serialization)
- Use ObjectLogic built-ins to create user-defined extension functions or reuse existing built-ins

2.3 Configuration

This section covers specific configuration options related to RDF. For all other settings such as evaluation methods or persistency, see the main OntoBroker documentation.

For OntoBroker RDF the ontology language has to be RDF.

```
OntologyLanguage = RDF
```

If necessary, you can activate additional RDF(S) entailment rules. There are two optional groups defined to infer implicit defined classes and/or properties. The default setting for both options is off.

```
RDF.Infer.Classes = on|off
RDF.Infer.Properties = on|off
```

Finally, if query performance is more important than storage consumption, all of the RDF(S) entailment rules can be materialized. By default, materialization is switched off.

```
MaterializeAxioms = on|off
Materialize.RDF.Entailment.Domain = on|off
Materialize.RDF.Entailment.Range = on|off
Materialize.RDF.Entailment.PropertyTransitivity = on|off
Materialize.RDF.Entailment.PropertyInheritance = on|off
Materialize.RDF.Entailment.ClassInheritance = on|off
Materialize.RDF.Entailment.ClassTransitivity = on|off
Materialize.RDF.Helper.Subclass = on|off
Materialize.RDF.Helper.Subproperty = on|off
Materialize.RDF.Helper.Generic = on|off
Materialize.RDF.Helper.TypeTemp = on|off
Materialize.RDF.Helper.Type = on|off
Materialize.RDF.PropertyDef.ByPredicate = on|off
Materialize.RDF.PropertyDef.ByDomain = on|off
Materialize.RDF.PropertyDef.ByRange = on|off
Materialize.RDF.PropertyDef.BySubpropertySubject = on|off
Materialize.RDF.PropertyDef.BySubpropertyObject = on|off
Materialize.RDF.ClassDef.ByType = on|off
Materialize.RDF.ClassDef.ByDomain = on|off
Materialize.RDF.ClassDef.ByRange = on|off
Materialize.RDF.ClassDef.BySubclassSubject = on|off
Materialize.RDF.ClassDef.BySubclassObject = on|off
```

The best evaluation method to choose is highly dependent on the characteristics of the given data and queries. The default settings should work fine for most use cases. For further optimization, the following settings may be tested alternatively:

- Set evaluation method to Dynamic Filtering 2:

```
EvaluationMethod = DynamicFiltering2
```

- Set evaluation method to Disjunctive Magic Set in combination with bottom-up evaluator Bottom-Up 2:

```
EvaluationMethod = DisjunctiveMagicSet
```

BottomUpEvaluator = BottomUp2

- Activation of module names ground:

```
ModuleNamesGround = on
```

Please note that some restrictions apply for concept and attribute names ground. See the FAQ 24 for more details. Note in addition that some filter expressions are not bottom-up save. If such filter expressions are required, use some top-down based evaluation like Dynamic Filtering 2 since bottom-up evaluation is not possible in such a case.

2.4 API

This chapter gives a short introduction on working with the OntoBroker Java API. All main tasks like creating ontologies and running queries are covered in the following explanations and code fragments. The source code of this example is located in RDFDemo.java (class com.ontoprise.examples.rdf.RDFDemo) within the OntoBroker installation directory.

2.4.1 By Example

We start by initializing the ontology manager, either with a local connection or to a remote server instance. In this example a local connection is used, configured for RDF. In addition the module names ground is activated.

```
Properties properties = new Properties();
properties.setProperty(IConfig.MODULE_NAMES_GROUND, IConfig.ON);
OntologyManager manager = KAON2Manager.newOntologyManager(OntologyLanguage.RDF,
properties);
```

A new ontology can be created using the initialized manager. The first argument "module" in createOntology equals the RDF context or graph identifier. Simply set this value to a valid absolute IRI. The second argument allows you to specify further options for the new ontology. For our example we do not need them and set the argument to null.

```
Ontology ontology = manager.createOntology(module, null);
```

The created ontology is empty. There are several ways to add new statements, i.e. triples. Let's start by importing the content of a RDF ontology file, here referred to as "sample.rdf" in the current directory. In this example, all of the additional arguments of the import method are not required.

```
ontology.importContentsFrom(new File("sample.rdf"), null, null);
```

New statements can be also created using the API. The following lines of code show the construction of several RDF terms which are finally combined to a new RDF statement. This statement is then added to the ontology.´

```
KAON2Factory factory = KAON2Manager.factory();
Constant subject = factory.constantIRI("http://www.example.com/#cain");
Constant predicate = factory.constantIRI("http://www.example.com/#name");
Constant object = factory.constantRDFText("Kain", "de");
FPropertyMember stmt = factory.fpropertyMember(subject, predicate, object);
ontology.addAxiom(stmt);
```

Using Turtle syntax, which is compact and easy to read and also avoids some issues of RDF/XML serialization, new statements can be inserted by executing a SPARQL Update command.

```
manager.execute("sparqlUpdate \"\"\"PREFIX : <http://www.example.com/#> "
 + "INSERT DATA { GRAPH <urn:demo#> { :able :age 32; :knows :cain. } }\"\"\"");
```

Serializing a ontology is provided by the method saveOntology.

```
ontology.saveOntology(OntoBrokerOntologyFileFormat.RDF_TURTLE, new File("sample-
modified.ttl"));
```

Finally we query the ontology using SPARQL. This example specifies a simple query including a filter. The results are printed to the console. For details on queries see the SPARQL compatibility overview. Non-standard query extensions are also explained there. For example the 'debug' option generates a more verbose output and shows details of the internally used rules and query optimizations. A quite handy tool especially for the optimization of complex queries. For better legibility, exception and error handling has been excluded.

```
  // Execute SPARQL query
  reasoner = ontology.createReasoner();
  String queryText = "SELECT * WHERE { ?s ?p ?o . FILTER (?o > 30) } OPTIONS
'debug'";
  query = reasoner.createQuery(Namespaces.INSTANCE, queryText);
  query.open();

  // show variables used
  Variable[] vars = query.getDistinguishedVariables();
  for (Variable var: vars) {
      System.out.print(var.getVariableName() + "\t");
  }
  System.out.println();

  // show results
  Term[] buffer = query.tupleBuffer();
  Formatting format = ontology.getOntologyFormatting().getDefaultFormatting();
  while (!query.afterLast()) {
      for (Term term: buffer) {
          System.out.print(term.toString(format) + "\t");
      }
      System.out.println();
      query.next();
  }
  query.close();
```

2.4.2 SPARQL Query

SELECT can be used straightforwardly by iterating across the resulting variable bindings. However, for other SPARQL query forms the result is not defined as variable bindings. ASK returns a Boolean value, CONSTRUCT and DESCRIBE will produce an RDF graph. To remain compatible with the general OntoBroker query interface these result types are mapped to standard variable mappings.

2.4.2.1 Boolean Queries

For ASK queries, the boolean value false is represented as no query results, whereas true is mapped to exactly one empty solution.

2.4.2.2 Graph Queries

The resulting graph of CONSTRUCT and DESCRIBE queries is represented as a set of triples. Each triple maps to one solution, providing a triple instantiation: Variable $s for subject, variable $p for predicate, and variable $o for object position.

Example:

Given the RDF ontology

```
@prefix : <http://www.example.com/ontology#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
:alice :name "Alice" .
:bob :name "Bob" .
```

and the SPARQL query

```
PREFIX : <http://www.example.com/ontology#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?person vcard:FN ?name }
WHERE { ?person :name ?name }
```

will produce the resulting graph, represented as triples via variable bindings

$s	$p	$o
:alice	vcard:FN	"Alice"
:bob	vcard:FN	"Bob"

which equals the RDF ontology

```
@prefix : <http://www.example.com/ontology#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
:alice vcard:FN "Alice" .
:bob vcard:FN "Bob".
```

2.4.3 SPARQL Update

Update operations can be executed within the standard command framework of OntoBroker. The wrapper command **sparqlupdate** takes exactly one argument which specifies the SPARQL Update string.

Example:

```
ontologyManager.execute("sparqlupdate \"\"\"CLEAR DEFAULT; LOAD <http://www.w3.
org/1999/02/22-rdf-syntax-ns.rdf>\"\"\"");
```

The return value will be Boolean. TRUE in case of success, otherwise an exception is thrown. The update operations are also available via the SPARQL endpoint.

2.4.4 Factory Overview

RDF terms like resources and literals are created using the KAON2Factory factory provided by KAON2Manager. factory(). The table below shows all of the relevant methods for OntoBroker RDF.

Category	Element	Factory method
Resource	URI/IRI resource	constantIRI
	Blank node	constantBNode
Literal	untyped	constantSimpleLiteral
	language tagged	constantPlainLiteral
	boolean	constantBoolean
	date/time	constantDate constantTime constantDateTime
	duration	constantDuration
	double/float	constantDouble
	other numeric	constantDecimal constantInteger
	string	constantString
	XML literal	constantXMLLiteral
	other	typeCode=TypeRegistry.getTypeCode(typeURI) constant(value, typeCode)

For more details, see the JavaDoc delivered with OntoBroker RDF. Particularly the class com.ontoprise.files.rdf.shared.RDFUtils provides further useful RDF related methods.

2.5 RDF Compatibility

2.5.1 Supported Formats

RDF im-/export is realized using the Sesame RDF parsers. All common RDF serialization formats, i.e. RDF/XML, Turtle, N3, and N-Triple, are supported. Most RDF based applications support RDF/XML. However, note that RDF/XML imposes some restrictions on the models which can be serialized, since the URI of a predicate has to be serialized as a valid QName. In particular the local part of a QName has to be a valid LocalPart.

Example:

The triple

```
<my:subject> <my:predicate!> "object" .
```

cannot be serialized in RDF/XML since the last character in "predicate!" is a exclamation mark and thereby not a valid NCNameChar.

2.5.2 Internal Model and Round-Tripping

Inside OntoBroker RDF all statements are stored in a predicate-based, optimized structure. Depending on the RDF predicate of a statement, different "tables" are used. By activating one of the names ground options, this diversification will be even more sophisticated. Adapting the RDF graph model to the internal predicate-based model imposes some restrictions on data types. During reasoning and query answering, performance is of outmost importance. One of the limiting factors, e.g. for join operations, is the efficient comparison of RDF terms regarding equality. This particularly affects RDF literals. For example, comparing two numeric literals "42"^^xsd:short with "42"^^xsd:integer is more expensive compared to a simply object identity check due to necessary type conversions, assuming that the same objects share their identity, which is the case in OntoBroker RDF. On the decision between efficient reasoning and data type preservation we favored performance. As a result, some of the numeric XML Schema data types are irreversibly mapped to a unified internal representation. The table below shows those conversions.

Original data type	Mapped data type
xsd:double xsd:float	xsd:double
xsd:int xsd:long xsd:integer xsd:byte xsd:short xsd:decimal xsd:negativeInteger xsd:positiveInteger xsd:nonNegativeInteger xsd:nonPositiveInteger xsd:unsignedByte xsd:unsigendInt xsd:unsignedLong xsd:unsigendShort	xsd:int xsd:long xsd:integer xsd:decimal

`xsd:double` and `xsd:float` are mapped to `xsd:double`. The differentiation between `xsd:int/.../xsd:decimal` is based on the value of the literal. For a uniform mapping every numeric literal is cast to the data

type with the minimal value range containing the value. Therefore "32"^^xsd:integer and "32"^^xsd:long are both internally represented by "32"^^xsd:int. This conversion also affects the round-tripping behavior for RDF ontologies. Any other RDF terms like literals with user-defined data types are preserved. Blank nodes are also supported. The blank node identifiers are adapted during import in order to ensure uniqueness even in the case of graph merging. However, this will not affect the proper semantics of blank nodes.

2.6 Inferencing

OntoBroker RDF supports a configurable sub-set of RDF/RDFS inferencing. As default a sub-set is choosen to achieve good performance. This includes these RDFS rules

- RDFS2: RDF.Entailment.Domain
- RDFS3: RDF.Entailment.Range
- RDFS5: RDF.Entailment.PropertyTransitivity
- RDFS7: RDF.Entailment.PropertyInheritance
- RDFS9: RDF.Entailment.ClassInheritance
- RDFS11: RDF.Entailment.ClassTransitivity
- RDF.PropertyDef.ByHierarchy
- RDF.ClassDef.ByHierarchy

in which the last two rules implement a sub-set of the axiomatic facts covered by the internal concept/property hierarchy.

If more detailed control is required inferencing using the hierarchies could be disabled and selectively (additional) rules switched on/off. Simply activate the configuration settings

```
RDF.Infer.Properties = on
RDF.Infer.Classes = on
```

and the additional rules

- RDF1: RDF.PropertyDef.ByPredicate
- RDF.PropertyDef.ByDomain
- RDF.PropertyDef.ByRange
- RDF.PropertyDef.BySubpropertySubject
- RDF.PropertyDef.BySubpropertyObject
- RDF.ClassDef.ByType
- RDF.ClassDef.ByDomain
- RDF.ClassDef.ByRange
- RDF.ClassDef.BySubclassSubject
- RDF.ClassDef.BySubclassObject

can be used instead of the internal hierarchies.

Finally some internal rules are used to mediate between asserted and inferred facts.

- RDF.Helper.Subclass
- RDF.Helper.Subproperty
- RDF.Helper.TypeTemp
- RDF.Helper.Type

All given rules can be incrementally materialized by setting the option "Materialization.<identifier>" to "on", e.g.

```
Materialization.RDF.Entailment.ClassTransitivity = on
```

2.7 SPARQL

OntoBroker RDF supports SPARQL (pronounced "sparkle"), the W3C recommendation for querying RDF data. This chapter describes details on the implementation, compatibility issues and extensions. If you are not familiar with SPARQL we recommend that you first of all read the official specification.

2.7.1 Error Handling

Using the API and working with SPARQL queries is straightforward. If invalid (in terms of syntax or supported by OntoBroker) queries are specified, one of the following issues should be checked:

- General syntax/parser issues: The obvious case.
- Reuse of blank node in different graph patterns: According to SPARQL definition, it is prohibited to share blank nodes in different graph patterns.
- Unsupported expression: The arithmetic comparison of boolean sub-expressions is not supported.
- Variables not bound by graph pattern: Variables which are not bound inside a basic graph pattern will raise an exception. This is a design decision to prevent the rather uncommon case of intensionally unbound variables and should be seen as a error indicator, for example, typos in query variables.
- Invalid query options syntax: A syntax violation or unknown option in the OPTIONS argument.
- Conflicting namespace definitions: Namespaces can be specified inside the query string and using the namespace argument for query creation. If both places provide conflicting information, i.e. same prefix but different namespace, an exception indicates this mismatch.
- Ordering on boolean expressions: Expressions used in ORDER blocks always have to return non-boolean values.

2.7.2 Standard Conformance

General

- Implicit REDUCED mode: Using optimized evaluation strategies, an early elimination of duplicates is important. Therefore per default all SPARQL queries operate in REDUCED mode. This might also affect the selection part of CONSTRUCT and DESCRIBE queries.
- Lexical literal value matching: As described in RDF Compatibility only the values, but not their original lexical representations are stored. Resulting queries for literals like "0001"^^xsd:integer will always be translated to "1" of type xsd:int.
- Handling of unbound variables: This feature of SPARQL logic is implemented using a special NULL constant which represents an unbound variable. In combination with UNION patterns, a variable might be "actually bound" with NULL, which could prevent some of the valid solutions found outside the UNION pattern.
- Default graph: In OntoBroker RDF also the default graph is labeled by an identifier (obl:reserved: defaultModule).
- Namespace prefixes: Those definitions are only supported for a limited Unicode range (characters below 0x0300).
- Blank nodes in WHERE clause: If explicitly named blank nodes are used in WHERE clause they are matched against those blank nodes in the store – and do not act as variables. If the standard W3C semantics are required, either an anonymous blank node or a variable has to be used.

OPTIONAL and UNION patterns

- Multiple OPTIONAL patterns may decrease the overall query performance.
- For limitations on UNION have a look at section General/Handling of unbound variables.

Expressions for FILTER and ORDER

- Locality of filter expressions: The locality principle of filter expressions for variable bindings is not supported. Instead variable bindings are globally visible on all levels of nested structures.
- Implementation of ternary logic: SPARQL filter expressions are based on ternary logic, which extends the boolean logic by the value 'error'. Inside OntoBroker RDF the mapping to horn logic enforces a

"compatibility mode" where the value 'error' is mapped to 'false'. While most of the filter expressions work as expected, the result will differ especially for negation of errors (in SPARQL a negation of an error is an error, where in OntoBroker RDF the negation of false will be true).

- Comparison of Boolean sub-expressions, such as (?a < 3) > (4 != ?b) is not supported.
- XSD type casts are implemented for xsd:string, xsd:boolean, xsd:date, xsd:time, xsd:dateTime, xsd:duration, as well as the supported numeric values xsd:decimal, xsd:long, xsd:integer, xsd:int, xsd:double. For the numeric conversions, please consider OntoBroker's implementation of the RDF data model.
- Extended date/time support: For convenience date/time values are already taken into consideration in compare operations.
- Queries using filter expressions accessing variables (not only in the current graph pattern) may not be evaluated bottom-up. Please choose an alternative evaluation method such as Dynamic Filtering instead.

Ordering

- Order expressions: Only expressions not returning Boolean values are supported.

CONSTRUCT queries

- Construction patterns using blank nodes are memory-intensive and should only be used for small to medium result sets.
- Solution modifiers will operate on the result graph, not on the intermediate query results. See DESCRIBE below for details and use cases.

DESCRIBE queries

- As it is up to the SPARQL query processor to define descriptions, OntoBroker will describe every resource by all associated properties, i.e. statements with the resources to describe in subject position.
- Solution modifiers for DESCRIBE like ordering are ignored. Using LIMIT and OFFSET modifiers the resulting graph will be sliced, but due to missing sorting the actual selection might be arbitrary. However, LIMIT could be used to restrict the resulting graph to a maximum number of statements, for example, for protection from denial-of-service attacks which might use queries with low selectivity to return large result sets.

Update operations

- The SPARQL Update implementation is based on the current SPARQL Update W3C draft (14 October 2010), using the SPARQL 1.0 Query definition for the where-clauses. Minor differences in comparison to the W3C examples and grammar are explained in the following lines.
- BASE and PREFIX definitions allowed at the beginning of an operation sequence and shared among the operations, and subsequent operation-local prefix definitions are not expected.

2.7.3 Extension Functions

Expressions might contain calls to extension functions. In OntoBroker those functions are provided by built-ins, available in the context of FILTER expressions:

- Filter built-ins are mapped to functions with boolean return value. There is one limitation to this approach: If you need the boolean return value outside of boolean operations, e.g. as an argument for another function, you have to use a functional built-in (see next paragraph) returning an explicit boolean term. However, a built-in filter will be sufficient for filtering in most use cases. For example, you might want to write an filter built-in with signature "even/1" to check if the argument is an even number. Another example is the already existing ObjectLogic built-in contains/2 (with namespace obl:reserved:) which checks if a string is contained in another string. This built-in might be used to filter for triples with their object containing the word 'house':

```
PREFIX obl: <obl:reserved:>
SELECT * WHERE { ?s ?p ?o. FILTER (obl:contains(?o, "house") }
```

- Functional (and relational) built-ins allow the returning of an arbitrary RDF term. Mapping in the SPARQL function space is based on one simple assumption: The last argument of the built-in will always be the return value. Therefore the last argument has to be omitted if the built-in is called as a function. For example, a function to return the distance between two geographic coordinates, represented as a user-defined data type with latitude and longitude as one literal, might have the function signature

"geoDistance/2" (coordinate1, coordinate2). However, the associated OntoBroker built-in has to increase the arity by one to provide the return value: "geoDistance/3" (coordinate1, coordinate2, distance). An exemplary SPARQL query using this function to find all cities with distances larger than 2000 km:

```
 PREFIX obl: <obl:reserved:>
 PREFIX : <http://www.example.com/geo#>
 SELECT * WHERE { ?city1 a :City; :location ?l1. ?city2 a :City; :location ?l2.
FILTER (obl:geoDistance(?l1, ?l2) > 2000) }
```

- Any other built-in type like aggregations or connectors cannot be used inside SPARQL expressions.

Out of the box there are several built-ins available for various operations (mathematical, strings, time etc.).

2.7.4 Extended Query Options

Beside the standard SPARQL syntax and semantics, OntoBroker RDF offers additional query options:

- debug: Enables debug mode with more detailed output on the rules and optimizations.
- visualDebug: Start visual debugger, a Swing based GUI for debugging.
- inferOff: Disable RDF(S) inferencing.
- profile: Information about possible performance issues.

Simply extend the query by the keyword OPTIONS (not case sensitive) and add all of the query options enclosed inside a SPARQL string, separated by comma. An example:

```
 SELECT * { ?s ?p ?o } OPTIONS "inferOff, debug"
```

For more detailed information on the query options and interpretation of the output, see the main OntoBroker documentation.

2.8 SPARQL Endpoint

OntoBroker RDF supports an endpoint providing access to SPARQL Query and SPARQL Update. Therefore OntoBroker RDF can be used in context of Linked Open Data (LOD).

To activate this web service, either use the OntoConfig option

```
 SparqlEndpoint = on
```

or start the server with option `-sparqlEndpoint`. If left unchanged in the settings (option `HttpPort`), the default port is 8267. The path for the service is `/services/sparql/query` for queries and `/services/sparql/update` for updates. For example, if the OntoBroker web service is running on "localhost" standard port 8267, the query service is accessible via `http://localhost:8267/services/sparql/query` and the update service via `http://localhost:8267/services/sparql/update`. If no parameters are supplied the debug mode showing a simple HTML based UI is activated.

To verify if the service is started you can access `http://localhost:8267/services` which provides an overview of all deployed REST or REST-style (and SOAP) services. Additionally via `http://localhost:8267/services/sparql?_wadl` a Web Application Description Language (WADL)-based specification of the service parameters and supported MIME types can be derived. Please be aware that this generated description does not reflect every detail, e.g. parameter qualifiers like required and repeating are not included.

For testing and debugging purposes the debug mode might be used. It is activated either by directly accessing the service without any arguments or explicitly setting the parameter debug to true. The lightweight HTML based UI allows you to switch between update and query mode. For queries the URI-results are linked to basic SPARQL queries thus supporting a basic navigation through the loaded data. Both the query and update services are documented in detail in the next sections.

About Linked Open Data (LOD):

While the traditional web links a huge amount of human readable web pages to each other the great vision of the Semantic Web creates a web of data. A light-weight version of this web of data are linked open data (LOD) sources. They collect structured information and provide this information in LOD servers. These LOD servers are accessible with the http-protocol and thus by ordinary browsers. Objects are identified by URIs. For instance DBpedia is a system extracting information from Wikipedia and providing this information in a structured way as an LOD server.

Ontobroker can be exposed as such an LOD endpoint. Together with its integration capabilities this provides a large amount of possibilities to bring data into the linked open data cloud.

2.8.1 Queries

To invoke the query service, at least one query has to be specified. For example, using the default endpoint configuration the query

```
SELECT * {?s ?p ?o}
```

will result in the encoded URI

```
http://localhost:8267/services/sparql/query?query=SELECT%20*%20%7B?s%20?p%20?o%7D
```

Further examples are available at W3C SPARQL Protocol HTTP Examples.

More technically, the service supports both GET and POST requests. An overview of all of the parameters is provided in the following table:

Parameter	Type	Description
query	xsd:string	The SPARQL query (required)
default-graph-uri	xsd:anyURI	Explicitly set default graphs (0..n)
named-graph-uri	xsd:anyURI	Explicitly set named graphs (0..n)
debug	xsd:boolean	Switch to debug mode if true, i.e. show HTML based UI (optional)

If both by SPARQL query and via arguments default-graph-uri or named-graph-uri graphs are selected, the argument's definitions are used.

The query result will be returned in different formats, depending on the query form and HTTP content negotiation. In all cases UTF-8 is used as encoding.

SPARQL Query Form	Result Type	Result Format (MIME)
SELECT	Variable bindings	SPARQL Query Results XML Format (application/sparql-results+xml) as default or JSON (application/json)
ASK	Boolean value	
CONSTRUCT	Graph	Turtle (text/turtle) as default or RDF/XML (application/rdf+xml)
DESCRIBE		

2.8.2 Updates

One or more update operations has to be specified for invoking the update service. For example, the operation

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

will result in the GET-encoded URI

```
http://localhost:8267/services/sparql/update?update=LOAD%20%3Chttp://www.w3.
org/1999/02/22-rdf-syntax-ns.rdf%3E
```

Besides the update parameter, the debug option can be specified. The following table shows an overview of

all service arguments, both available for HTTP GET and POST:

Parameter	Type	Description
update	xsd:string	The SPARQL Update operation(s) (required)
debug	xsd:boolean	Switch to debug mode if true, i.e. show HTML based UI (optional)

The result is returned as plain text. Either with a simple status message that everything went fine, i.e. all operations have been executed without an error, or an description of the error that occurred.

2.9 FAQ

What is the difference between RDFTerm-equal and sameTerm?

Comparing two terms by equality (operator '=') checks for value equality, e.g. "1.0"^^xsd:double is equal to "1"^^xsd:int. sameTerm in contrast checks for identity, e.g. "1.0"^^xsd:double is different from "1"^^xsd:int. Please note that due to the implicit type casting e.g. "1"^^xsd:long and "1"^^xsd:int will be identical (and of course equal too). Depending on your data, if the check sameTerm is sufficient, a huge performance benefit might be the result. Especially in combination with a persistent data model sameTerm is much more efficient.

Default and named graphs?

The dataset is defined by the query using a combination of FROM and FROM NAMED. If one or more named graphs are specified but no default graph the default graph is empty. If only one or more default graphs are given but no named graphs the set of named graphs is empty. Finally, if neither a default nor named graphs are specified the default dataset is used. This contains the current, i.e. queried, ontology as a default graph and all other graphs as named graphs.

ObjectLogic support?

SPARQL functionality might also be used in combination with a ObjectLogic data model. Care has to be taken for the slightly different axiomatization of ObjectLogic and differences in handling/representation of property domain/range information. Therefore the result of SPARQL queries might differ. For loading new ontologies via SPARQL Update the ontology format has to match the data model ontology language, e. g. for ObjectLogic only ObjectLogic ontologies can be loaded. Finally if non-RDF-terms like functional terms are returned by SPARQL queries these are converted to special URIs (obl:term#ObjectLogic) similar to the ontology transformation.

Names ground optimization?

Depending on the chosen names ground configuration not all of the SPARQL/RDF features are available. A safe option without any restriction is to activate module names ground. This might increase the overall performance. If you choose concept names ground or attribute names ground the merging of the default graph in SPARQL queries is not supported. In addition, no sub-properties may be used for attribute names ground. Those single restrictions also apply for the combination of different names ground settings.

3 OntoBroker GUI

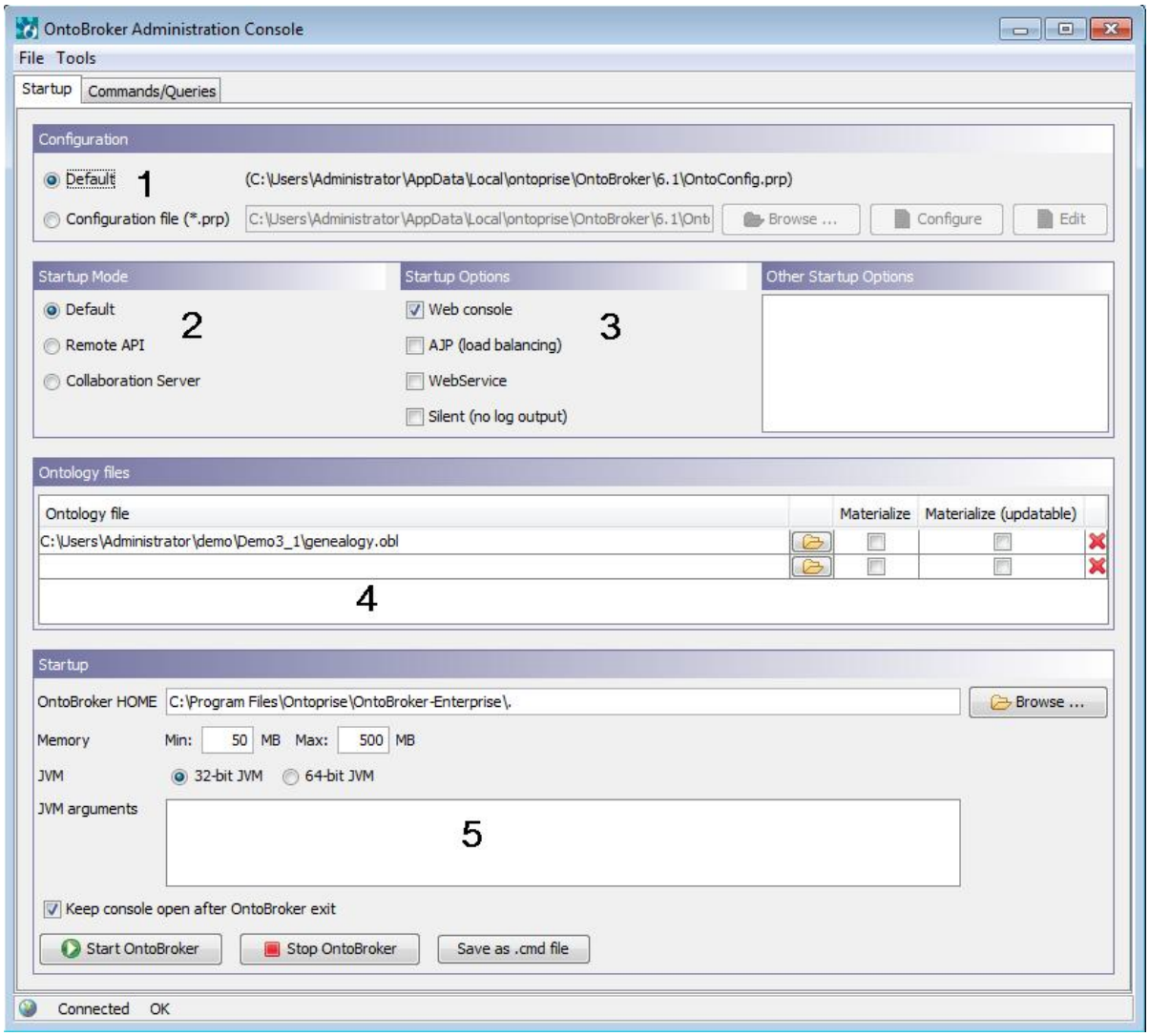
The OntoBroker Administration Console is an easy to use graphical user interface for configuring and executing OntoBroker as a server and serves as a client for demonstration and test purposes. This chapter describes how to use the OntoBroker Administration Tool.

3.1 Administration Console GUI

The OntoBroker Administration Console is launched by clicking on **Start** -> **Programs** -> **OntoBroker** -> **Administration Console.**

"Startup" tab

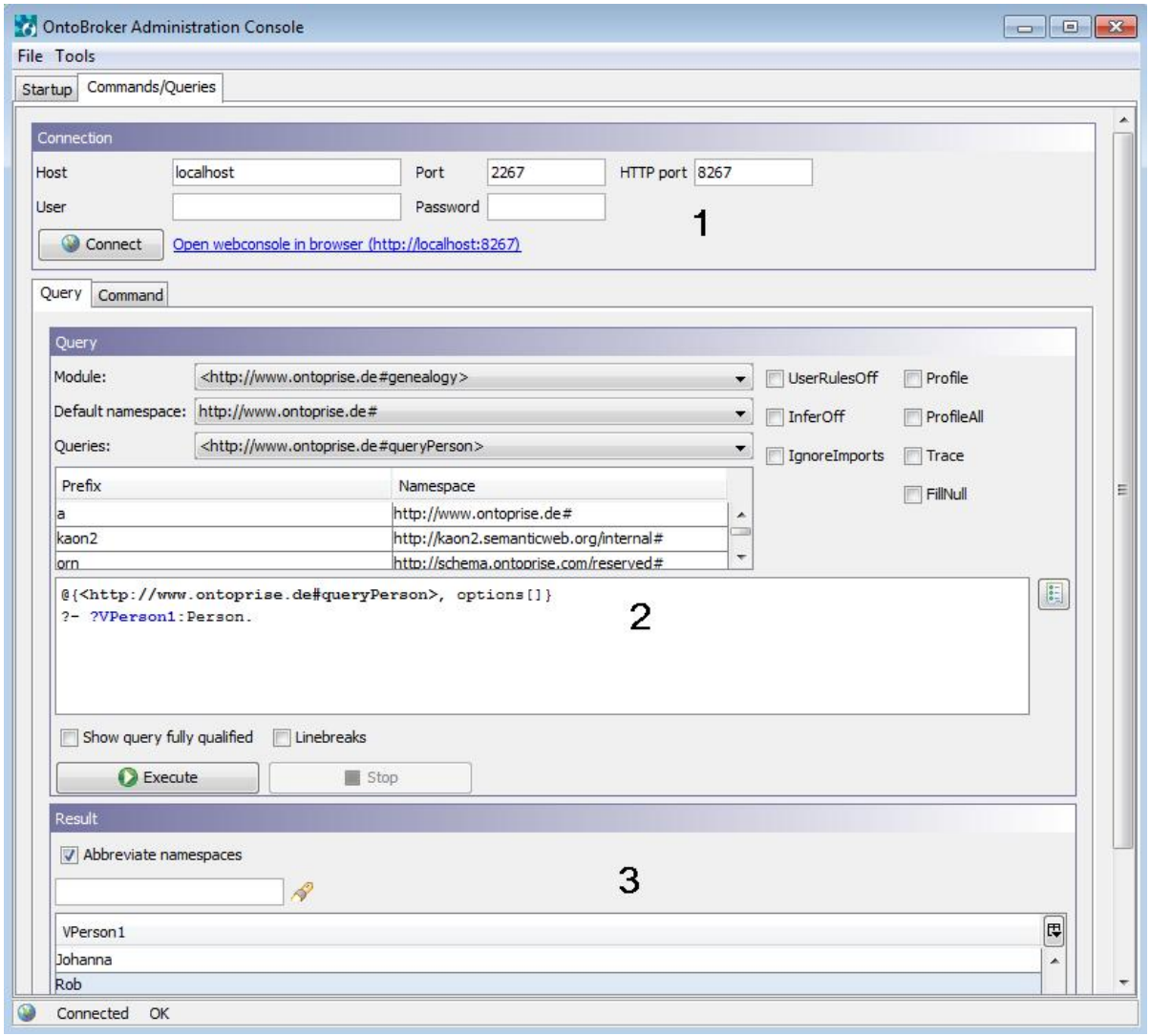
The user interface of the Administration Console consists of 5 areas in the Startup tab:



Nr.	Area	Description
1	**Configuration**	You can use the default values or load your own configuration file (see OntoConfig.prp at the OntoBroker home directory).
2	**Startup Mode**	Within the startup mode OntoBroker can be launched in default mode or it can either be configured as a collaboration server, or accessing Remote API.
3	**Startup Options**	OntoBroker is also started as a 'WebService' , if ' WebService' is activated. If 'Silent' is activated there will be no output to the OntoBroker console. The Web console is the default value. For more information on the Webconsole click here 36
4	**Ontology files**	In the Ontology files-area , files of folders can be added with which OntoBroker should start. These files must contain facts and rules in an input format like ObjectLogic, RDF (S), OWL or N-Triple (n3, nt).
5	**Startup**	Within the startup area, you can configure the heap size of OntoBroker and JVM arguments. The configuration can be saved by clicking <Save as .cmd file> in the configurations-field. Furthermore, you can start or finish OntoBroker here.

"Commands/Queries" > "Queries" tab

The user interface of the Administration Console consists of 3 areas in the main window of the **Commands/ Queries** tab:



Nr.	Area	Description
1	**Connection**	After the configuration is complete, the port-number (standard port is <2267>) on which OntoBroker should run has to be chosen. The IP-address can also be specified.
2	**Query**	On the "Query" - tab, queries can be sent to OntoBroker and the answers are shown. Choosing the "Default Namespace" and the "Module" - entering and changing the query - will affect the result view. By activating the options e.g. "fill null" the query will be extended accordingly. To create queries in an easier manner a default module can be chosen from the drop down menu. This module will be used to receive answers from OntoBroker. The given prefix, shown within the "Prefix/Namespace" field, can be used instead of the namespace.
3	**Result**	The result of your query is shown in the result area.

The **History** button provides a history of your executed queries.

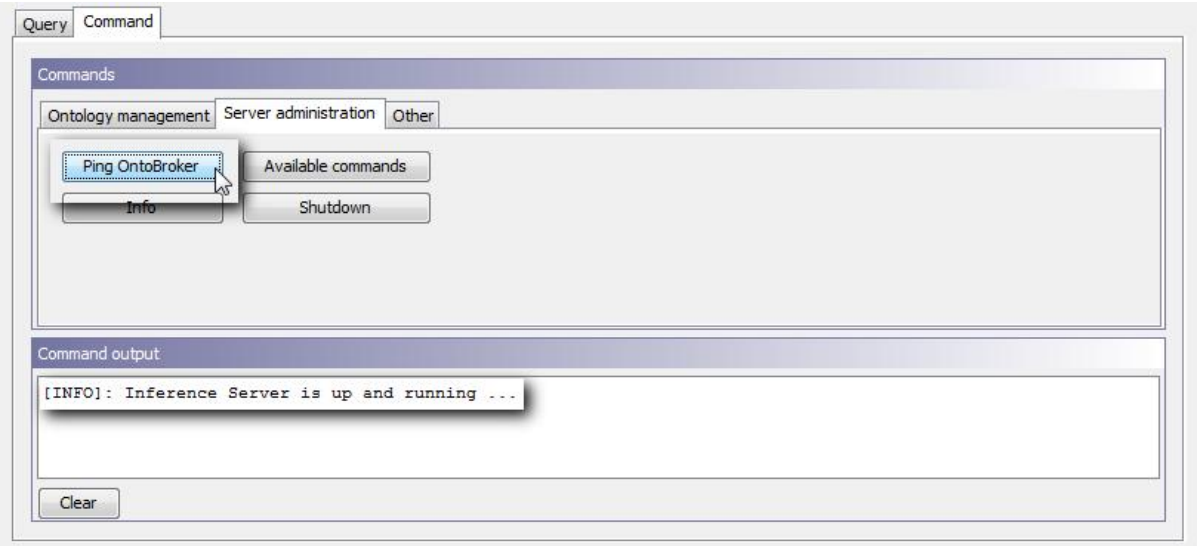
"Commands/Queries" > "Command" tab

Within the main window of the **Commands/Queries** tab there are several sub-areas. The screenshot below shows the **Command** tab again with its tabs and functions.

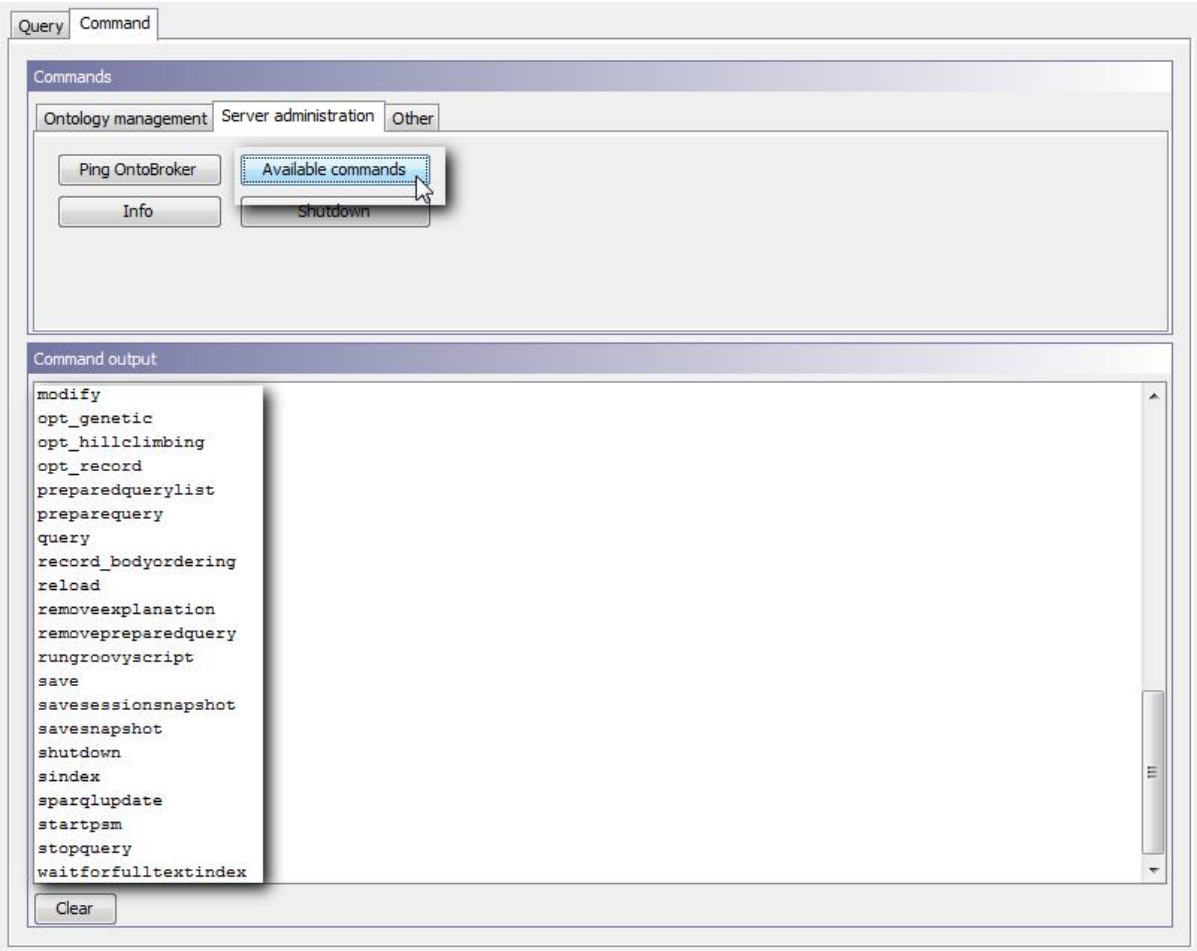
Nr.	Description
1	**Ontology Management**
2	**Command Output**

"Commands/Queries" > "Server administration" tab

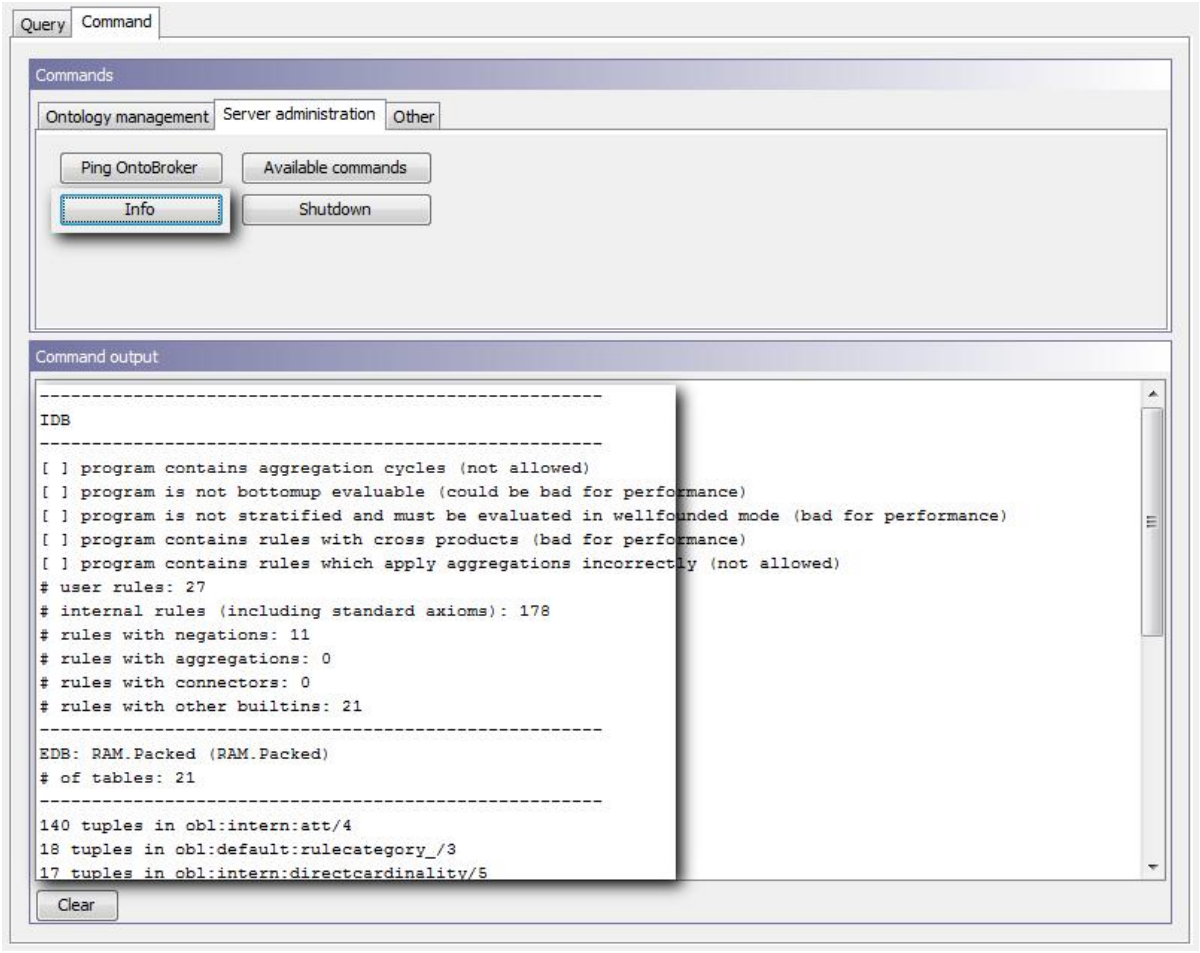
The **Server administration** tab offers several options for handling commands or getting basic information on OntoBroker. A list of all of the available commands is also available.



A very useful function is the **Available commands** button. One click provides a list of all commands. Select a command and copy and paste it:



The **Info** function shows all of the important information about a running OntoBroker instance and the loaded ontology:



"Commands/Queries" > "Other" tab

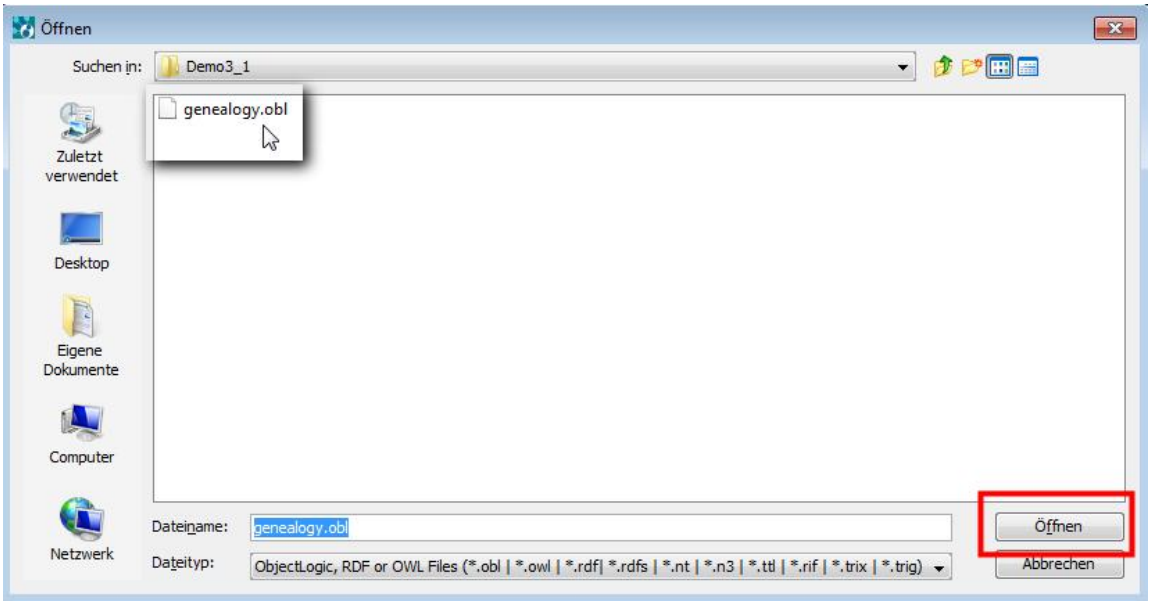
The **Commands/Other** tab allows you to execute commands in a comfortable way. Type in a command or paste a command from the list of commands and choose **Execute**. The OntoBroker-feedback is shown in the **Command output** area.

3.2 Starting OntoBroker

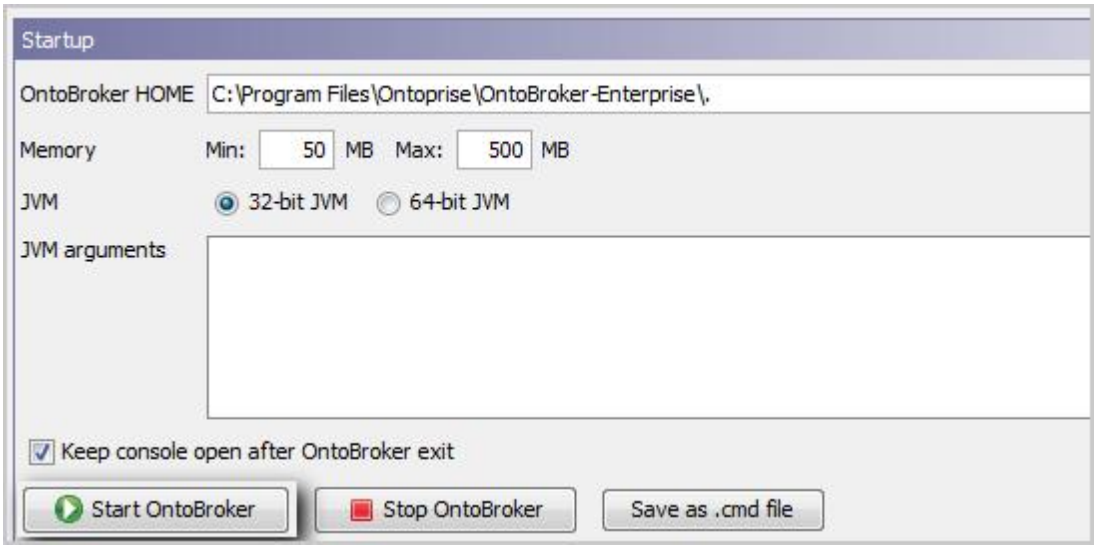
This is a short introduction on how to load an ontology, start Ontobroker and make a query.

1. First of all you have to browse through your directory for your ontology file:

2. Select your ontology (here: companies.obl).

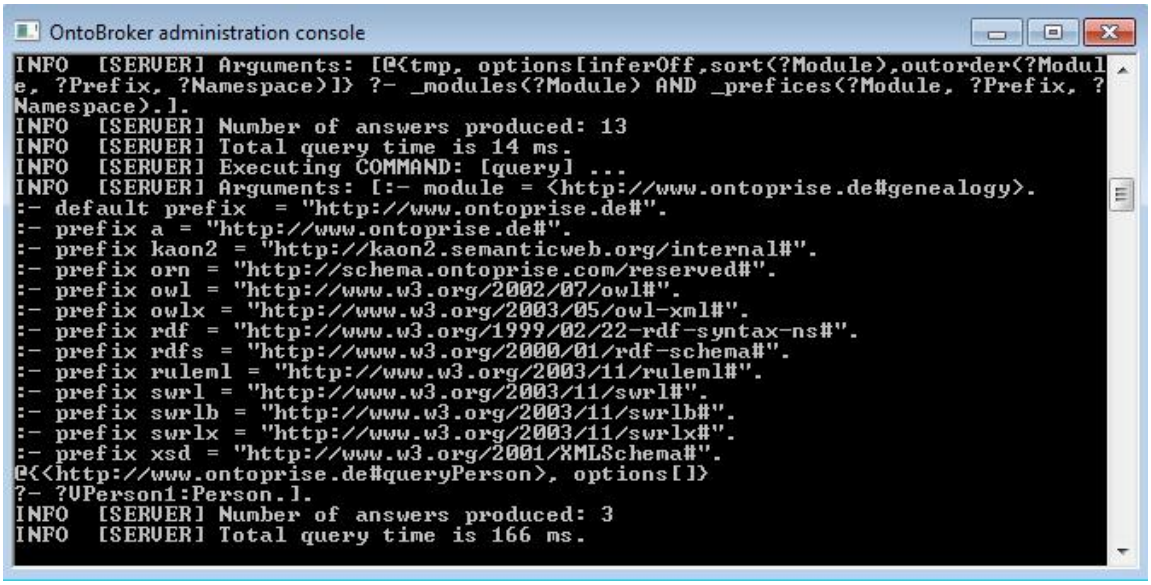


3. Start OntoBroker (your ontology file will be loaded).



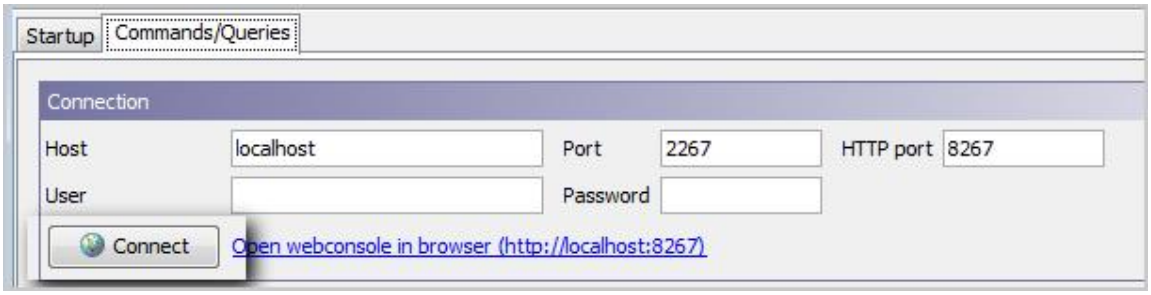
The console window shows any information about the starting OntoBroker (e.g. configuration).

After OntoBroker has been started, a command line box should appear (see below). After all of the files have been successfully loaded, the OntoBroker Administration Tool shows the message <OntoBroker up and running...>.



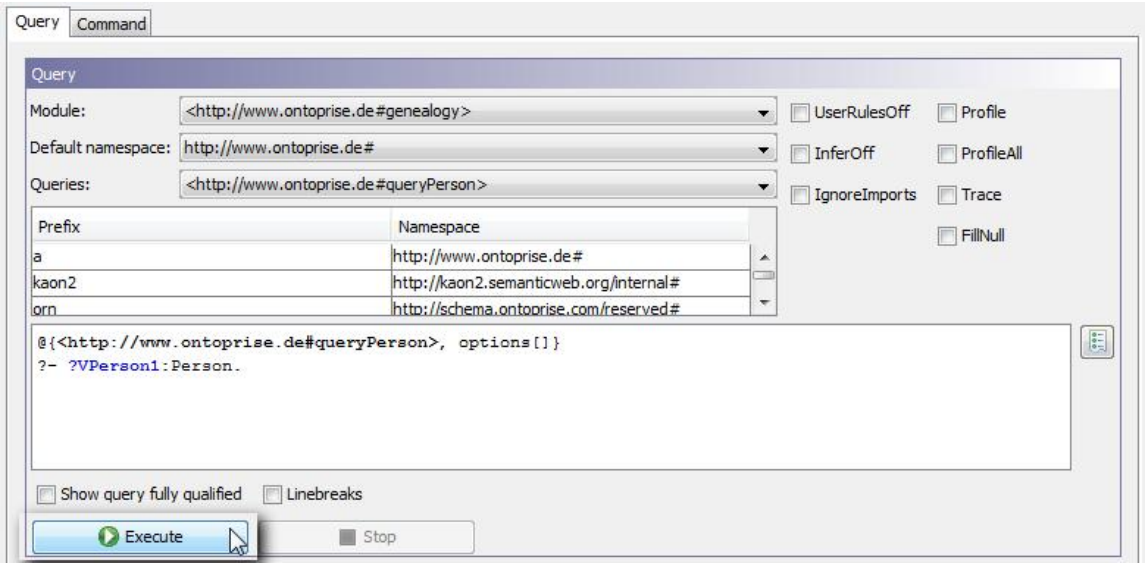
4. Switch to the **Command/Queries** tab and connect your OntoBroker to the webconsole.

   In addition to the configuration and execution of OntoBroker, the Administration console also serves as a client for demonstration and test purposes. Under the **Command/Queries** tab you first have to connect to the OntoBroker. Select the server (host) and the port and then click on the **Connect** button. When authorization is required the user and password must be entered in the relevant fields. If the Administration console was able to connect to OntoBroker a suitable message appears.



5. Type in your query and press **Execute**.

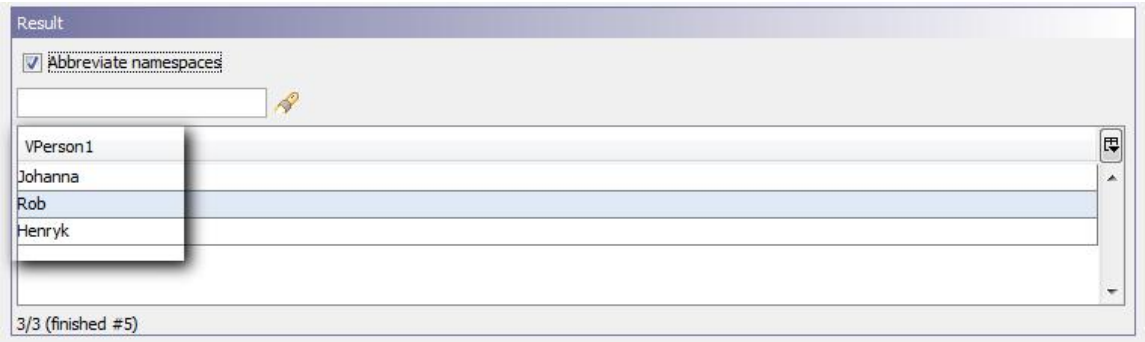
   On the Query tab, queries may be sent to OntoBroker and the answers are shown. Choosing the "Default Namespace" and the "Module" - entering and changing the query - will affect the result view. By activating the options e.g. "fill null" the query will be extended accordingly. To create queries in an easier way a default module can be chosen from the drop down menu. This module will be used to receive answers from OntoBroker. The given prefix, shown within the "Prefix/Namespace" field, can be used instead of the namespace.



Special characters may be used on the query interface:

- \n newline: Line break after temporary facts
- \" backslash: Mark quotes to be a quote inside of queries
- \\ double backslash: Mark a backslash to be a backslash inside of queries

The result of a query is shown in the Result area.



3.3 Specifying the VM

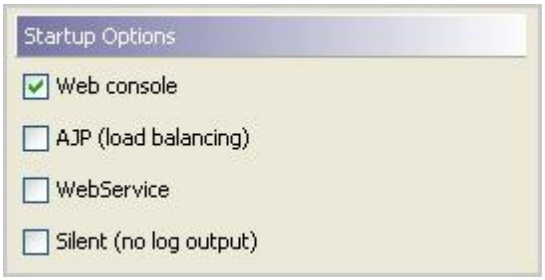
How to specify the vm OntoBroker should run with:

1. Rename `OntoBroker.exe` to `OntoBroker2.exe`.
2. Create an `OntoBroker.bat` with

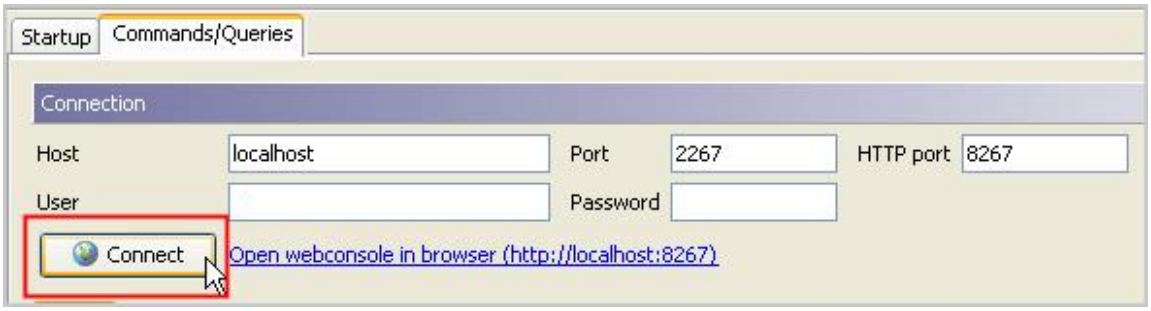
```
Ontobroker2 -vm "C:\Program Files\Java\jdk1.6.0_10\bin\java.exe" %*
```

3.4 Starting the Webconsole

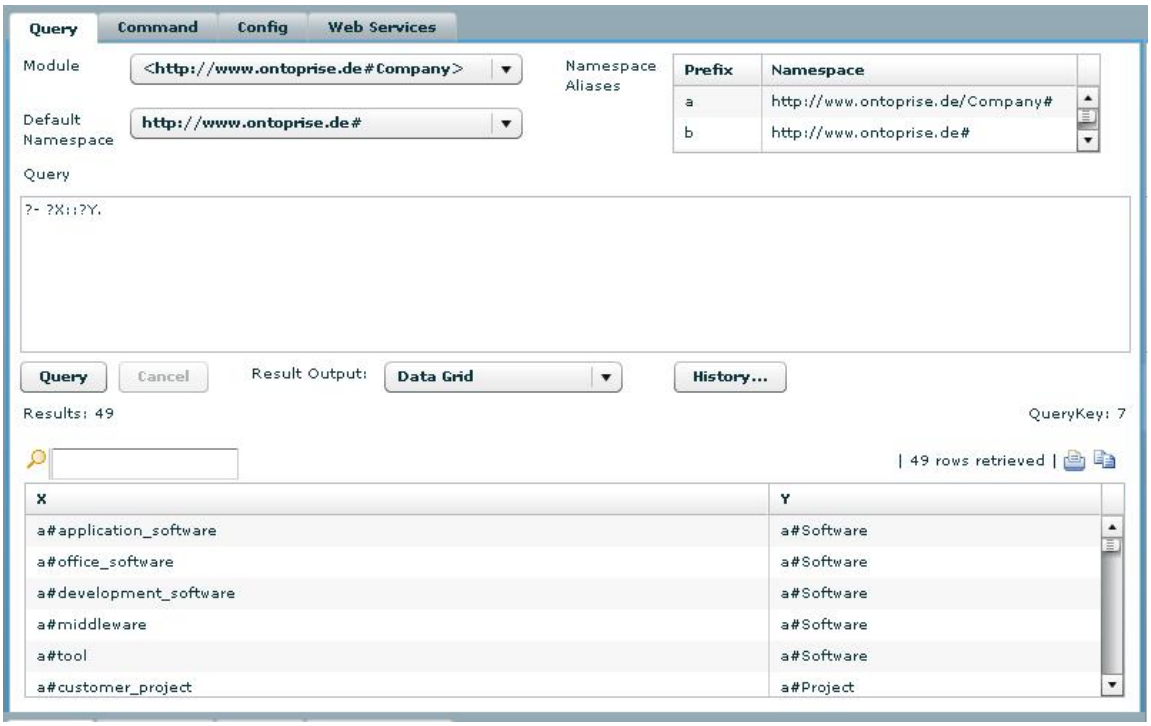
1. Flag the relevant checkbox as shown on the screenshot.



2. Press "Connect".
   The OntoBroker Webconsole is starting on localhost on port 8627:



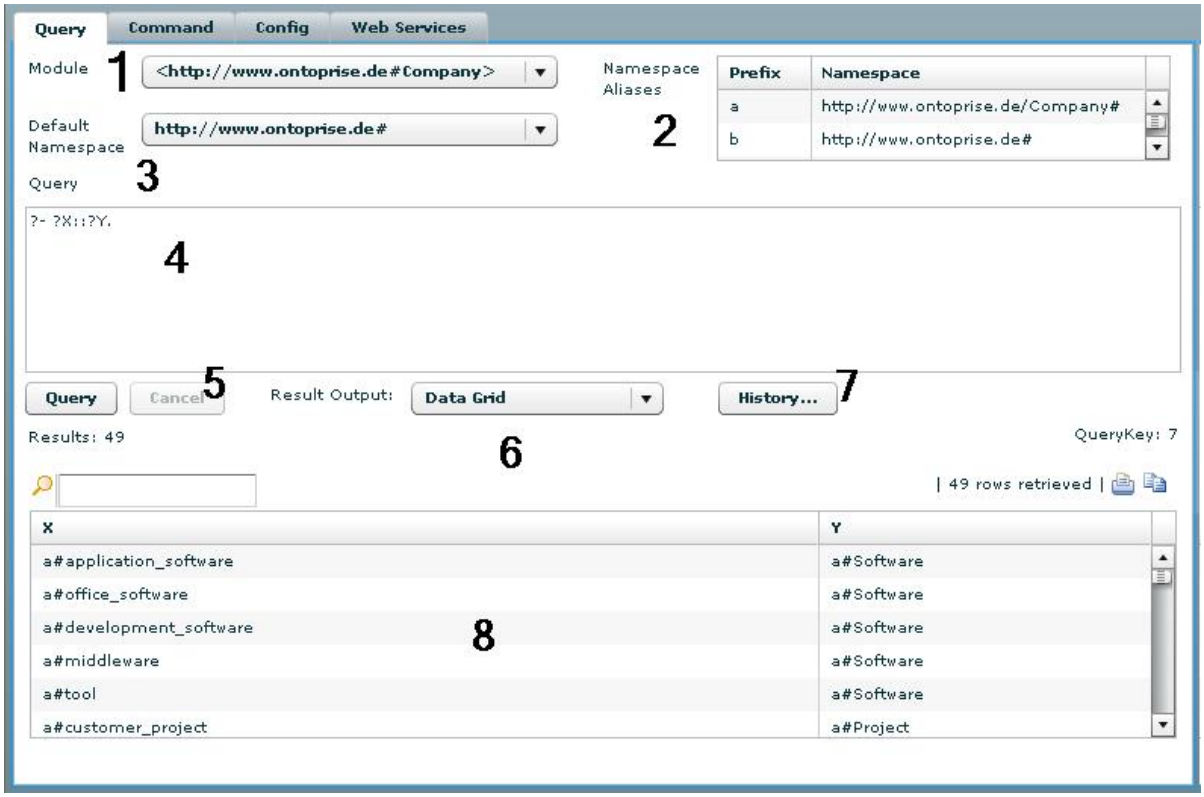
3. Type `localhost:8267` in the input window of your browser and press <ENTER>.
   The Webconsole is now displayed. Now queries 64 and commands 188 can be executed in a comfortable way.



3.5 Webconsole GUI

"Query" tab

Within the webconsole, there are several options for handling a query:



Nr.	Area	Description
1	Module	specifies which module is loaded
2	Namespace Aliases	Choose a prefix for the accordant namespace
3	Default Namespace	Choose the default namespace
4	Query	Type in the query here
5	Query/Cancel	Run the query or cancel the request
6	Result Output	Define the output format of the query.
7	History	The history of requests (queries) is displayed here
8	Result	The result of the query is displayed here

"Command" tab

On the **Command** tab, commands can be executed.

A list of all of the server commands can be found in the relevant chapter.

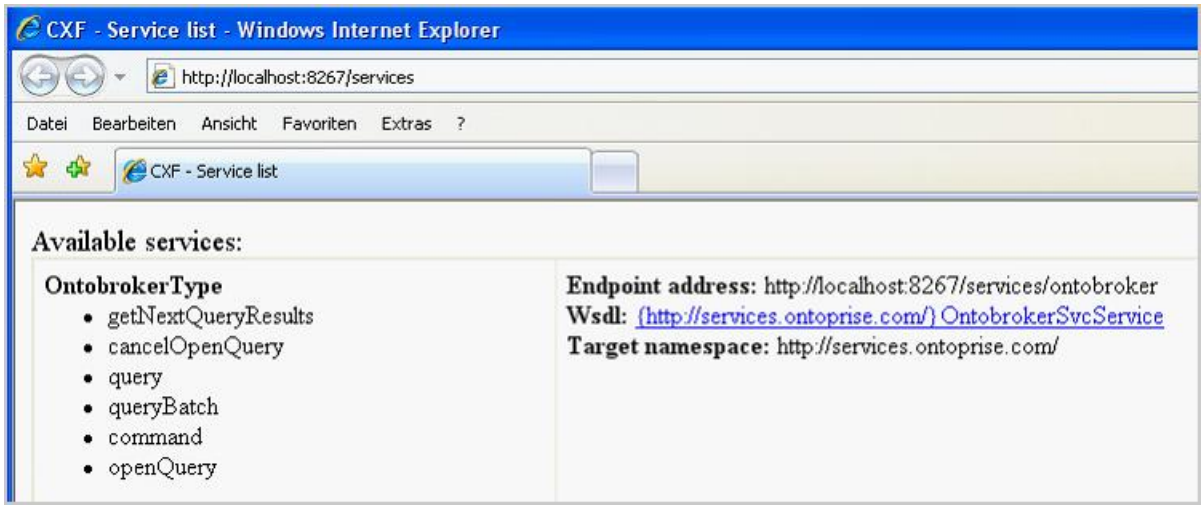
"Config" tab

On the **Config** tab all of the OntoBroker settings are displayed at a glance. Change your settings in the OntoConfig.prp.

"Web Services" tab

On the Web Services 105 tab, information on any Webservice started is displayed.

Just click on **Click here to see...** to get a list of all of the available Web services.

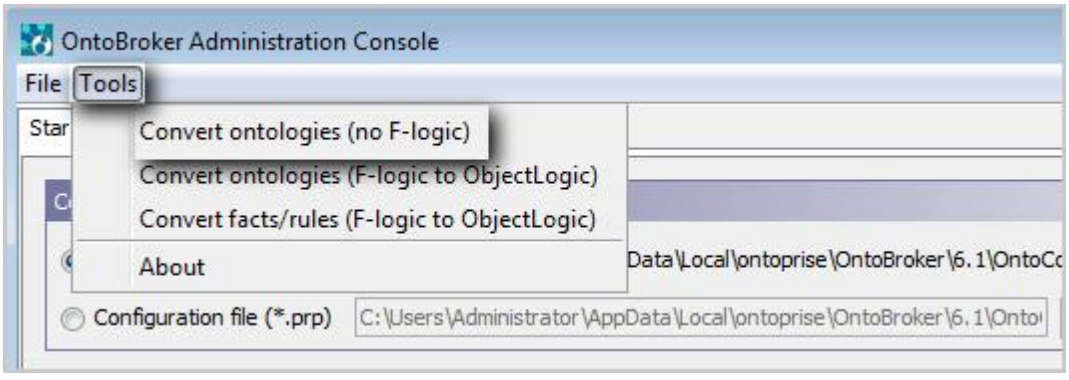


In the next step you get information on the Web service methods, the endpoint address and a link to the WSDL.

3.6 Converting Ontologies

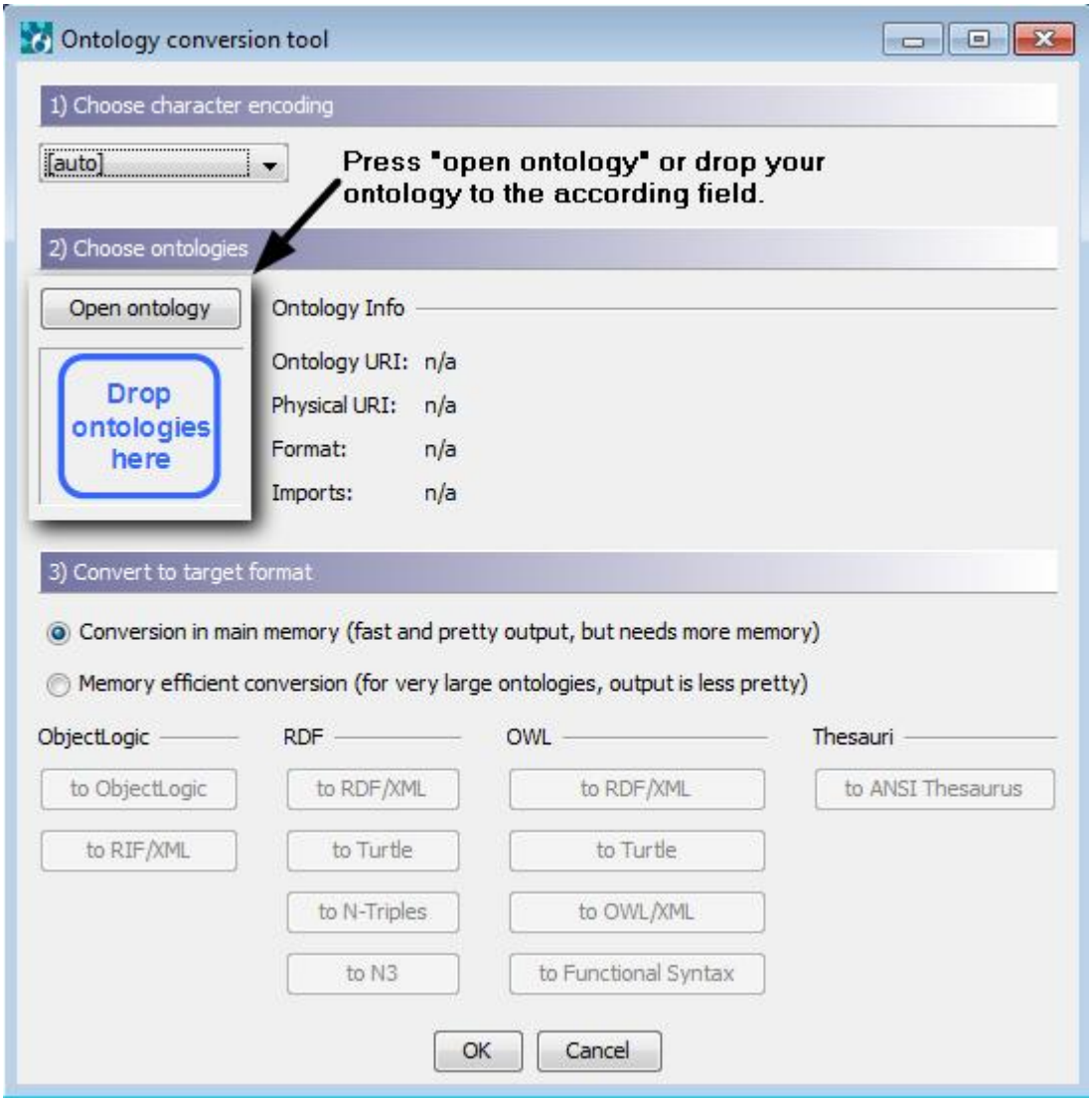
OntoBroker provides an easy and manageable way of converting ontologies to different formats.

1. Start the OntoBroker GUI by clicking on: **Start** -> **Programs** -> **OntoBroker** -> **OntoBroker Administration Console**.
2. From the top level menu select: **Tool** -> **Convert ontologies**.

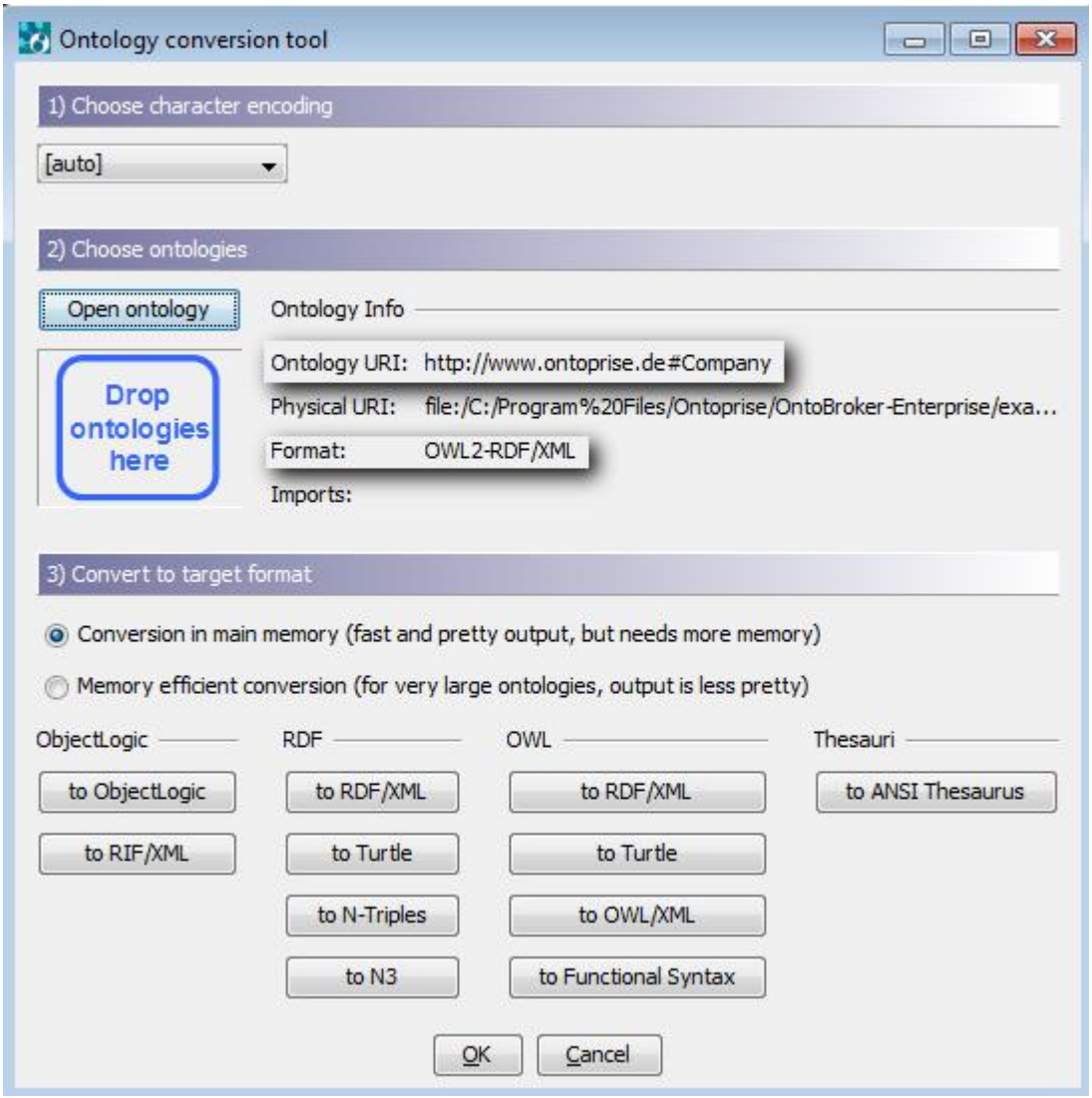


The "Convert Tool" of OntoBroker opens.

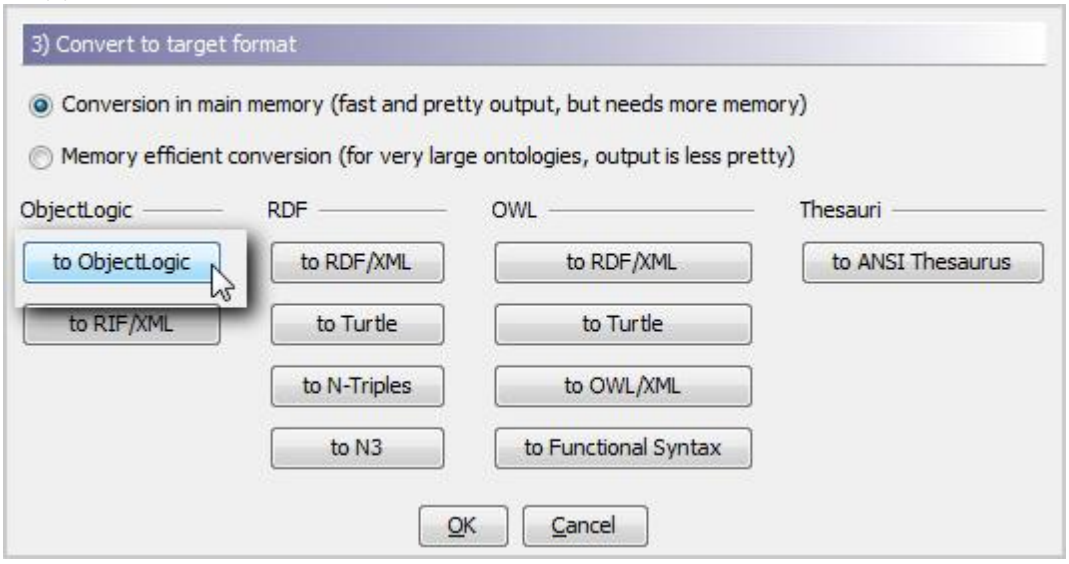
3. Add the ontology to be converted using the dialog box or using drag&drop:



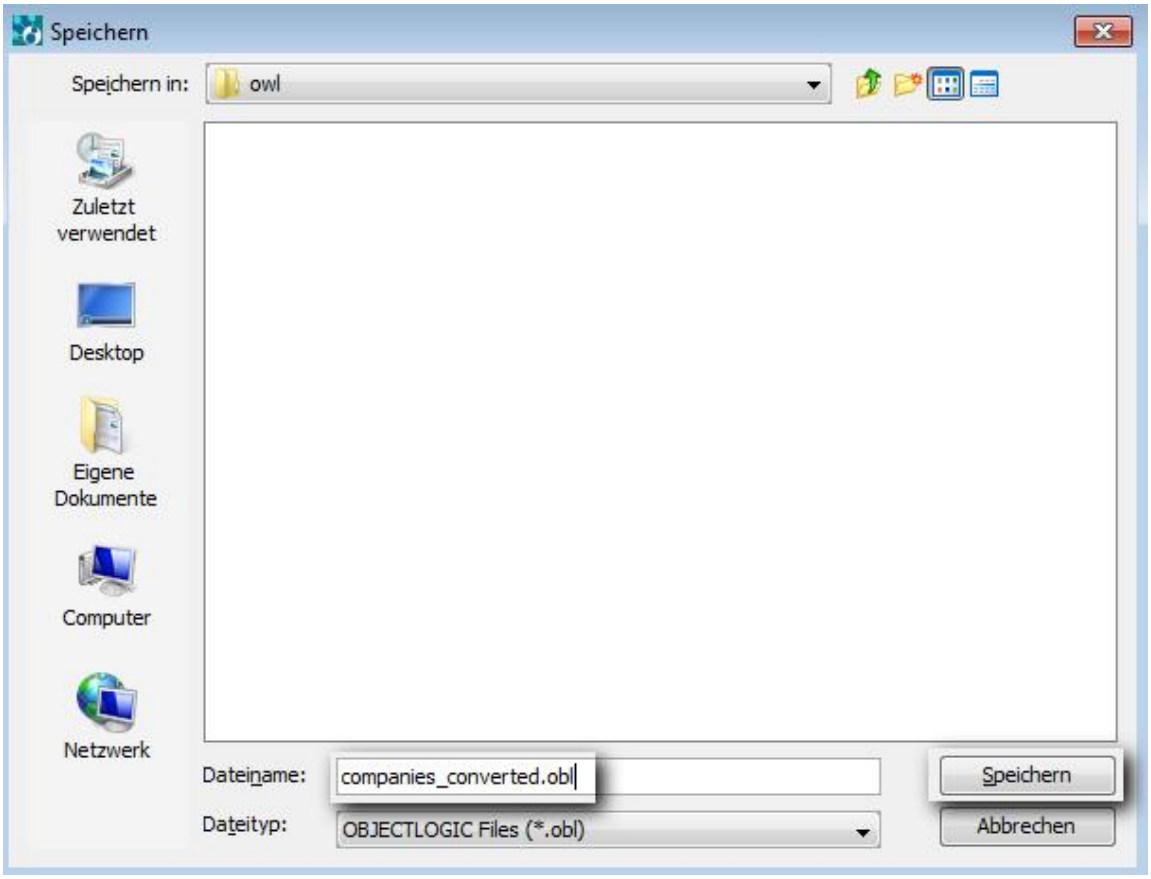
After adding the ontology, basic information on the ontology is displayed:



4. Choose the output format.
   Keep in mind that cardinalities differing from (0,1) and (0,*) are not supported by F-logic and will get lost during conversion. Rules, queries and mappings are not supported by owl and rdf(s) and will thus not be retained. Furthermore cardinalities and symmetric, inverse and transitive relations are not supported by RDF(S) and will also not be retained.



5. Choose a name for the converted ontology and click on **Save**.



The ontology will be converted and saved in the specified folder with the chosen filename.

Note:
If the conversion causes problems or information gets lost, you will be informed in the Messages window at the bottom end of the conversion tool.

ANSI Thesaurus export

This functionality can be triggered using the Ontology Conversion Tool.

- The target format should be an ASCII file (or UTF-8 for special characters) with extension .ansi
- The first line should contain a supplier ID such as:

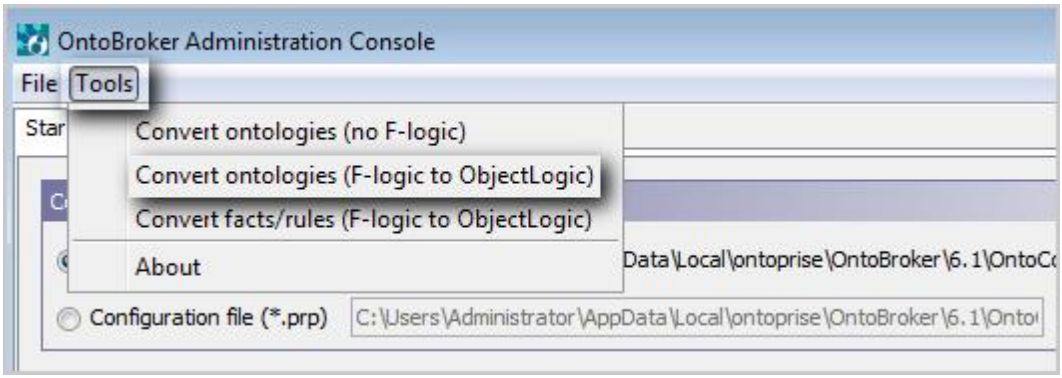
```
   SID ontoprise
 where only alphanumeric characters, _ and - are allowed in the single word supplier
ID (SID).
```

- As blanks are not allowed neither in the SID, nor in URLs they will be replaced with _
- Double quotes are also not allowed and will also replaced with _

F-logic to ObjectLogic conversion

To convert old F-logic ontologies to ObjectLogic proceed in the following way:

1. Start the OntoBroker GUI by clicking on: Start -> Programs -> OntoBroker -> OntoBroker Administration Console.
2. From the top level menu select: **Tool** -> **Convert ontologies/fact/rules (F-logic to ObjectLogic)**.



The "Convert Tool" of OntoBroker opens.

Proceed in the same way as for the other conversion process described at the beginning of this chapter.

4 Storage Systems

A storage system is the data structure where the facts and rules of an ontology are stored. OntoBroker has three different storage systems:

- RAM.AVL.Packed 169
- H2 273
- RAM.TS

The storage parameter can also be set to the value RAM.choose. In this case the OntoBroker selects the best RAM datamodel itself. Each system has its own strengths and unique features. The following section gives a short overview of these storage systems.

RAM.AVL.Packed

This is the new standard RAM datamodel which is tightly packed in order to reduce the memory. It uses hash indices for primary indices, tries for secondary indices and packed AVL trees for secondary sorted indices. This storage system is especially optimized for the evaluation methods BottomUp2 and DynamicFiltering2.

Note:
This is the fastest and most memory-efficient data model for the inference server.

In order to start OntoBroker with this storage system set

```
Storage=RAM.AVL.Packed
```

in OntoConfig.prp.

H2

OntoBroker ships with a persistent storage system (we currently use the H2 SQL engine). This persistent storage is embedded in OntoBroker, so it is very easy to setup OntoBroker with it. Facts and rules are stored persistently, i.e. OntoBroker can be restarted without the need to reload the ontologies.

In order to start OntoBroker with this storage system set

```
Storage=H2
```

in OntoConfig.prp.

RAM.TS

A new memory-efficient RAM model.

In order to start OntoBroker with this storage system set

```
Storage=RAM.TS
```

in OntoConfig.prp. This storage option will store tuples in a compact way. The new high-performance bottom-up evaluator (BottomUp3) will only work with RAM.TS.´

Note:
It is recommended to set `EvaluationMethod=choose` and `BottomUpEvaluator=choose` when using this storage option. Note that e.g. `EvaluationMethod=DynamicFiltering2` will work much slower with `Storage=RAM.TS` than with `Storage=RAM.AVL.Packed`.

5 Data Integration

Besides serving as a common communication language and representing expert knowledge, ontologies serve as an integration means of different legacy systems. The ontology is used to reinterpret given information sources in a common language and thus provide a common and single view of different data sources. An ontology can collect different sources and integrate them in a common logical model. This goes far beyond just building connectors between applications. The goal of integration is to consolidate distributed information intelligently, free of redundancy and providing users and applications with simple access to information without considering the underlying data structure or system.

Supported external Databases:

- Microsoft SQL Server 2000, 2005 and 2008
- Oracle 10g and 11g
- Oracle TimesTen in-memory database 11g
- Oracle TimesTen in-memory database cache 11g
- DB2 9.0 and 9.5
- other JDBC drivers can be embedded by configuration

5.1 Creation of database mapping rules

Database Schema Import in OntoStudio

Database mapping rules will be created automatically when using the database schema import.

SQLExecute

You can create your own db mapping rules, using the sqlexecute predicate:

```
?X:"Person"@_defaultModule :- _sqlexecute("SELECT colname FROM tblname",[?
X],"mydbconnection").
```

NOTE: There must be a fact stored in an loaded ontology defining the connection details of "mydbconnection", e.g:

```
_dbaccessuserdata("mydbconnection",
"oracle","databasename","localhost:1521","user","pwd").
```

For more information see the built-in 227 "_dbaccessuser".

Database User Management and Database Configuration

For external database access you will need read access to the tables and read & write access to the relevant database for creating temporary tables.
The conversion of data types in a database can be different from the data type conversion of OntoBroker. Of course it can happen that when you execute built-ins like string2number on the database, the result differs from the OntoBroker result. When non-trivial queries involving rules with _dbaccessuser built-ins are executed, it can be necessary to create temporary tables for storing intermediate results. Therefore your connection needs permissions to create and delete temporary tables and insert facts in these tables.

MS SQL Server

The following user privileges are required:

- Select
- Insert
- Create table
- Drop table

The tempdb database is used. It is necessary that the database and the tempdb have the same collation.

Oracle

The following user privileges are required:

- Select
- Insert
- Create table
- Drop table

If you don´t have write access to the schema, proceed in the following way:

1. Create a new user.
2. Grant writing access to this new user on his own schema.
3. Grant just read access to this user on the schema with data.
4. Login with this new user when doing a DB schema import in OntoStudio.
5. Choose the schema with data and import the tables.

DB2

The following user privileges are required:

- Select
- Insert
- Create table
- Drop table
- Use on user temporary tablespace (for Version DB2 8)

You need a user temporary tablespace for Version DB2 8.

5.2 Optimization

ShrinkDBAccess[171], SQLRewriter

Access to the database will be optimized whilst executing a query: _dbaccessuser/3 literals are rewritten in a way so that only the required columns are selected. Furthermore, the dbaccessuser literals are combined with some built-ins in a SQL query so that there are some restrictions.

Example:

```
?- _dbaccessuser("projects",F(id,VAR1,project,VAR1),"mydbconnection") and equal
(VAR1,"myproject").
```

will be rewritten

```
?- _sqlexecute("select id, project from projects where project ='myproject'",[?VAR1,?
VAR2],"mydbconnection").
```

It is also possible to combine multiple _dbaccessuser/3 literals. _sqlexecute/3 literals are not optimized.

You may encounter performance problems when dbaccessuser literals are not executed correctly (e.g. needless columns are selected, dbaccessuser literals are executed without restrictions). For a detailed analysis you can turn on the SQL tracing.

You may also try out if changing some of the following switches significantly improves the performance:

- Other evaluation methods
- Use namesground switches: CNG, ANG, MNG on/off.
- Using StrongUnfolder (can also be used if you only use a couple of rules)

Evaluation method DynamicFiltering2 with ANG=on is often a good choice.

5.3 Null Values

The access of external databases with the dbaccessuser built-in may result in null values from the database. A built-in with a null value as an argument is assumed to be false. As a consequence the built-ins are not called as soon as a null value occurs in its arguments.

SuppressNull

Note:
There was a bugfix on SuppressNull! This bugfix can change the behavior of old ontologies with db mapping!

OntoConfig Option:

```
SuppressNull = on
```

Default value is on.

Example*:*

Table "test" has following rows:

id	name
1	a
2	null

Rule:

```
p(?X) :- _dbaccessuser("test", F(ID,?X,NAME,?Y), "connectiondatasource").
```

Result before the bugfix:

?X
1
2

Result after the bugfix:

?X
1

The second row (2, null) is not selected now because the NAME is null. For getting the old result you should change the rule as follows:

```
p(?X) :- _dbaccessuser("tblname", F(ID,?X), "connectiondatasource").
```

Query Option Fillnull

Example:

```
@{q,options[fillNull]} ?- ?X:Person AND ?X[hasForename->?Y].
```

Result:

SuppressNull = off	SuppressNull = on
1,B 2,null	1,B 2,null

For more information on fillnull, see the accordant chapter 64.

5.4 Changing the Connection Data

You can change the connectionsdata by editing dbaccessuserdata_ fact:

```
_dbaccessuserdata("key","oracle","databasename","host:port","user","password").
```

For more information on data integration, see the OntoStudio Manual -> Import -> Import.

Connector for tuple cache

Connector Cache Options for OntoConfig

- `Connector.Database.Cache.Lifetime = query | unlimited`
- `Connector.SPARQL.Cache.Lifetime = query | unlimited`
- `Connector.OntoBroker.Cache.Lifetime = query | unlimited`
- `Connector.Excel.Cache.Lifetime = query | unlimited`
- `Connector.WebService.Cache.Lifetime = query | unlimited`

`query` - tuple cache is removed after the query execution. This is the default value for this option for all connectors in OntoStudio and for the WebService connector in Ontobroker.

`unlimited` - tuple cache is not removed after the query execution and can be used for the next query. This option is better for performance and should always be used if a database won't be updated a lot. The cache update should be done manually (see "Clear connector cache command"). This is the default value for this option in OntoBroker for all of the connectors except for the WebService connector.

5.5 Connector cache clear command

```
clear_connector_cache all | cache key
```

The command cleans a connector cache. This command can be executed only if Ontobroker is started with the option `Connector.*.Cache.Lifetime = unlimited`.

`all` - all tuples will be deleted from the cache.

`cache key` - only tuples for the cache key will be deleted. The cache key is a map term and can contain different values for different connectors.

Examples:

Database connector:

```
clear_connector_cache [_host->"data",_port->1521,_database->"orcl11",_user-
>"testuser",_table->"tblname"]
```

The table is optional; if no host is defined, the localhost will be used.

5.6 MergeImports

If the MergeImports parameter is set to "on" in the OntoConfig.prp, imported ontologies are directly merged into the importing root ontology. This means that the imported ontologies are not available as separate ontologies as for OntoStudio.

Example

To activate the MergeImports switch, set in the OntoConfig.prp

```
MergeImports = on
```

Assume that you have three ontologies A,B, and C. A imports B and B imports C. Then, after loading you have only one ontology A' (consisting of rules and facts from A,B, and C).

Note 1:
The flag influences only the loading process on OntoBroker start (or reload command).

Note 2:
The MergeImports flag cannot be used together with project files.

5.7 Integration Scenario

The scenario for this process was given by the business processes around the testing of cars. Our client has a fleet of test cars. These test cars are continuously reconfigured and then tested with this new configuration. Reconfiguration means changing the engine, changing the gear, changing the electric, i.e. changing all kinds of parts. For changing parts a lot of dependencies between these parts have to be taken into account. In many cases these dependencies are only known by certain human experts and thus require a lot of communication effort between different departments of the manufacturer, between the manufacturer and suppliers and between suppliers. Very often test cars have been configured which did not work or which hampered the measurement of the desired parameters. So, making such dependencies exploitable by computers allows for reducing the error rate in configuring test cars with a lower communication effort. The information on the car parts is stored in a considerable amount of databases filled by CAD-systems, measurement values, and so on, which have to be taken into account. These databases are fed by different tools in different departments. These databases have to be accessed during query-time to ensure the topicality of the data. During this project an ontology has been developed which serves as a communication medium between engineers, serves as a knowledge model representing these complex dependencies and serves as an integration model for different data sources. The following describes the last aspect in detail.

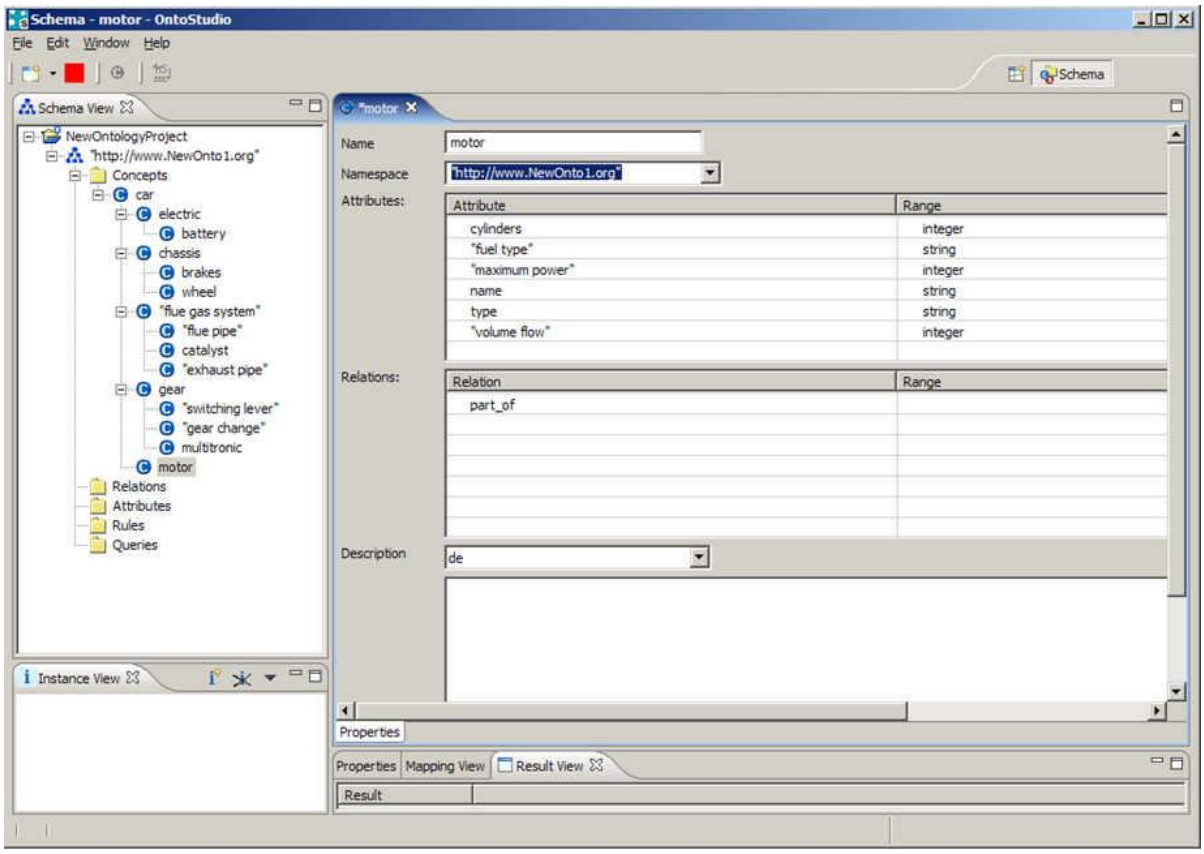
The Ontology

The base ontology very strongly relies on parts which are arranged in a part-of hierarchy and their properties. The instances, i.e. concrete values are most often gained from the parts list in the legacy systems.Our ontologies are represented in ObjectLogic. Specific basic concepts are represented as concepts in ObjectLogic and are arranged in an isa-hierarchy. Concepts may be described by attributes and relationships to other concepts.

```
component[
      has_part {0:*} *=> component,
      is_part {0:*} *=> component,
      horsepower {0:1} *=> _integer].
// instances and relations
tdi_engine:component.
valve2:component.
pump3:component.
tdi_engine[
      has_part->valve2,
      has_part->pump3,horsepower->340].

//rules
@{isPart} ?Y[is_part->?X] :- ?X[has_part->?Y].

//queries
@{allComponents} ?- ?X:component.
```

In this example we have defined a component as a basic concept. A component has a relationship has_part to another component and an attribute horsepower. Then we create an instantiation of a component tdi_engine being a specific engine. Concrete instances valve2, pump3 are given for concept component. A rule is used to describe the inversity of has_part and is_part. With a query we ask for all of the components in the model. OntoBroker, our reasoning system, provides the means for efficient reasoning in ObjectLogic. OntoBroker performs a mixture of forward and backward chaining based on the dynamic filtering algorithm to compute (the smallest possible) subset of the model for answering the query. During forward chaining not

only single tuples of variable instantiations but sets of such tuples are processed. It is well-known that set-oriented evaluation strategies are much more efficient than tuple oriented ones. The semantics for a set of ObjectLogic statements is then defined by a transformation process of ObjectLogic into normal logic (Horn logic with negation) and the well-founded semantics for the resulting set of facts and rules and axioms in normal logic.



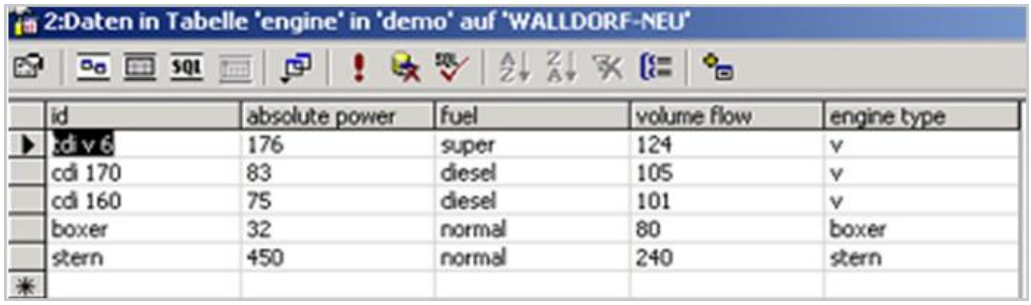
An excerpt of that ontology is shown in a part-of view in our tool OntoStudio. For example, it shows that a gear is part of a car and the switching lever is a part of the gear. For the motor some attributes such as maximum power, type, and so on are shown. An ontology without rules describes only simple relationships between concepts like a part is a part of another part, a part is connected to another part, and so on. More complex relationships have to be described by rules and constraints.

Data Source Integration

Besides serving as a common communication language and representing expert knowledge in our scenario, ontologies serve as an integration means of different legacy systems. The ontology is used to reinterpret given information sources in a common language and thus provide a common and single view of different data sources. In our scenario, the components data and the configuration data is already handled comprehensively in different departments and in different information sources like CAD-, CAE- or CAT-systems or ERP/PPS-applications, databases, and so on. All of these IT systems accompany the whole PLM-process, beginning with the product design and ending with the product release. Our test configuration system, and hence our ontology system must access this live information to be up-to-date, avoid inconsistent data and avoid additional effort. An ontology could now collect these different sources and integrate them in a common logical model. This goes far beyond building just connectors between applications. The goal of integration is to consolidate distributed information intelligently, free of redundancy and provide users and applications with simple access to information without considering the underlying data structure or system. In our case we already have such a commonly accepted logical model: the automotive ontology. This ontology describes schema information and is not yet populated with instances. This means e.g. that there is a concept motor with attributes name, cylinders, type, and so on. But there is no information on concrete motors like TDI V6, with 6 cylinders, fuel type super, and so on, available. This is achieved by attaching the ontology to one or more of the existing information sources. In the following we exemplify the mapping to a relational database.

Database Schema Import

The first step in connecting an ontology to a database is to import the database schema and visualize it in our ontology management environment OntoStudio. Beneath relational database schemas, OntoStudio also has import filters for other schemas like RDF or OWL. In our example we will show the attachment of a database table motor to our ontology.



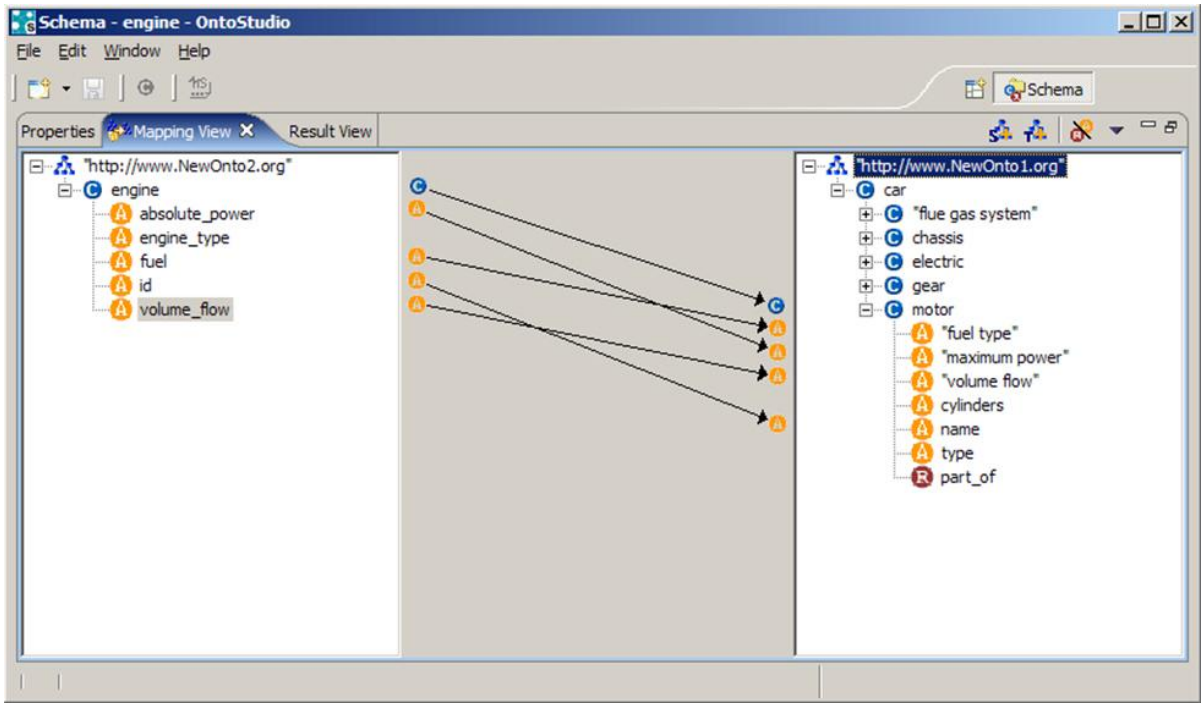
id	absolute power	fuel	volume flow	engine type
a6 v 6	176	super	124	v
cdi 170	83	diesel	105	v
cdi 160	75	diesel	101	v
boxer	32	normal	80	boxer
stern	450	normal	240	stern

The database mapping imports the table engine as a concept, columns id, absolute_power, fuel, volume_flow and engine_type as attributes. This means that every row in the database corresponds to one object in the ontology. OntoStudio automatically creates a connection to the database by the dbaccessuser-connector (there are various connectors to information sources available). This built-in automatically creates a unique object ID. It is used in a rule which defines the access and the mapping to the ontology:

```
@{mappingRule}
?V_ID:Engine[ID->?V_ID, absolute_power->?V_ABSOLUTE_POWER, fuel->?V_FUEL,
volume_flow->?V_VOLUME_FLOW, engine_type->?V_FUEL_TYPE]
:-
_dbaccessuser("engine",
F( "id", ?V_ID, "absolute power", ?V_ABSOLUTE_POWER, "fuel", ?V_FUEL
"volume_flow", ?V_VOLUME_FLOW, "engine_type", ?V_ENGINE_TYPE),
"mydbconnection").
```

Ontology Mappings

After having imported the database schema, the result ontology can be mapped with another ontology. Below you can see how a concept-to-concept mapping connects the concept engine to the concept motor and, additionally, an attribute-to-attribute mapping from id in the database ontology to name in the new ontology.



Another important mapping type is the mapping of attributes to concepts. This results in the attribute value being used as a unique ID for an ontology instance. For example, mapping the ID of engine to the concept motor creates an object for every different ID.

Querying the Integration Ontology

Mappings can be defined to different RDBMS and also to web services at the same time. A query to the integration ontology is thus translated in real-time (via the mapping rules) into calls for appropriate access built-ins which in turn access the data sources (in case of an RDBMS via SQL queries) and translate the answers back into ObjectLogic. Hence a user or an application on top of the ontology requires only this single ontology view and with it single vocabulary to retrieve all of the necessary information. In our scenario different information sources contribute to the same ontology. For example, information on electronic parts is stored in other databases than information on mechanical parts. Information on the 3-D geometry of objects is separated from their mechanical properties, and so on.

How to Handle Inconsistencies

It is clear that in practice the different information sources contain redundant or even inconsistent information. For instance, in our scenario, car types have not been represented in a unique way. The assignment of properties to car types has been described with different keys for one and the same car type. For example, keys like A3/A4 have been used to describe common properties of two car types while unique properties have been assigned to the car type by a key A3. We again use rules and thus inferencing to solve such problems.

```
?T:Car[carType->?Type, has_part->?Part]
:- _dbaccessuser("car", F( "id", ?T, "part", ?Part),
"mydbconnection") and ?Type is ?T._split("/")._memberAt(0).
```

Conclusion

In a real-life industrial project, viz. in the automotive industry at a car manufacturer, we have shown that ontologies may very well be used to enhance business processes and to integrate different information sources. In our case the ontology represents knowledge about relationships between different parts which may automatically be exploited in configuring test cars. This reduces the communication effort between the mechanical engineers, and reduces the error rate in configuring test cars. For this task the ontology is attached to the legacy systems of the manufacturer and thus accesses up-to-date information about parts and configurations. We have shown that our ontology engineering environment OntoStudio supports not only the comfortable development of ontologies but, with the integrated mapping tool OntoMap, also an easy to learn tool for attaching ontologies to different information sources. Our SemanticGuide is based on our ontology run-time environment and inference engine OntoBroker which is based on ObjectLogic. This semantic guide accelerates the configuration of test cars at our customer and hence also accelerates the development of new cars. In the end, this reduces time-to-market.

5.8 Supported Common Datatypes

If there is no ObjectLogic Constant (-), this means that this JDBC datatype is not supported by our tools.

JDBC data type	ObjectLogic Constants
ARRAY	-
BIGINT	INTEGER
BINARY	-
BIT	BOOLEAN
BLOB	-
BOOLEAN	BOOLEAN
CHAR	STRING
CLOB	-
DATALINK	-

DATE	DATE
DECIMAL	DECIMAL
DISTINCT	-
DOUBLE	DOUBLE
FLOAT	DOUBLE
INTEGER	INT
JAVA_OBJECT	-
LONGNVARCHAR	STRING
LONGVARBINARY	-
NCHAR	STRING
NCLOB	-
NULL	-
NUMERIC	DECIMAL
NVARCHAR	STRING
OTHER	-
REAL	DECIMAL
REF	-
ROWID	-
SMALLINT	INT
SQLXML	-
STRUCT	-
TIME	TIME
TIMESTAMP	DATETIME Web Service Connector 54
TINYINT	INT
VARBINARY	-
VARCHAR	-

Additional Oracle-Specific Data Type: NVARCHAR2, STRING

5.9 Web Service Connector

Builtin: _webserviceInfo/4 retrieves information for web service

Area	Description
Usage	`_webserviceInfo(webservice input map, service, method, method info map);`
Keys in Webservice input map	`[_wsdl ->"url of wsdl",_service ->”service name”,_method->"method name",_port->"port"]` The _service, _method and _port key/value pairs in this map are optional and can be left out, if the WSDL contains only one item of them. The service, method and method info map arguments must not be bound.
Example	`?- _webserviceInfo([_wsdl ->"http://weatherservicename/WeatherForecastSoap?wsdl"], ?X, ?Y, ?Z)` The web service connector supports access on web services.

Builtin: _webserviceAccess/3 calls a web service

Area	Description
Usage	`_webserviceAccess(webservice input map, input parameter map, output map)`
Webservice input map	`[_wsdl ->"wsdl string",_service ->_service,_method->"method name",_port->"port", _user->”username”, _password->”password”]` The _service, _method and _port key/value pairs in this map are optional and can be left out, if the WSDL contains only one item of them. If they are needed, you should use on of the values returned from the _webserviceInfo built-in. If the web service is protected with Basic authentication, you need to provide the _user, and _password key/value pairs. The values for the _user and _password parameters must be filled with user name and the password. Please note that the WSDL must be accessible freely. If this is not the case, download the WSDL manually and provide a file URL to it, e.g. "file:///C:/temp/service-to-access.wsdl"
Input parameter map	The concrete input parameter values are depending on the webservice. `["Parameter name"->"value"]`
Output map	This map contains the results of the web service call. The concrete structure depends on the web service. Use built-ins like _memberByPath/3 or _map2Table/4 to extract values. For further information about map data type click here .

Example	`Youmust download the secured wsdl file and provide the local path to the downloaded file in the query!`
	
	`?- _webserviceAccess([_wsdl ->"http://weatherservicename/WeatherForecastSoap?wsdl", _method->"GetWeatherByPlaceName"], ["PlaceName"->"Karlsruhe"],?Z).`

Example for BASIC authentication	`?- _webserviceAccess([_wsdl->”file:///c:/protectedService.xml”, _authorizationType->"BASIC", _user->"scott", _password->"tiger"], [...], [...], ?Result).`

5.10 REST Webservice Connector

This web-service connector is available since OntoBroker 6.3 and connects to REST-like web-services which deliver JSON as result. A OL-mapped equivalent structure is returned by the connector.

Syntax

Arguments for `_restWebserviceAccess/4`:

```
_restWebserviceAccess(<service>, [invocation args], [service args], ?result)
```

- Service: The IRI to access the service
- Invocation arguments: Map setting technical details for the service invocation

  Key `_method`: Selects between `_get` (default) and `_post` method
  Key `_mime`: Allows to explicitly specify the MIME type used, instead of default `application/json`
- Service arguments: Map with the arguments send to the service
- Result: The return value as map/list-structured output

Caching

The caching policy is set via standard (SOAP) Web-service connector configuration. To explicitly clear the cache the "clear connector cache" command with a cache key like.

```
[_connector->_restWebserviceConnector, _service-><...>]
```

has to be called. Please replace the ellipsis with the actual service IRI identifying the REST web-service.

Examples

A few examples to demonstrate the usage of the REST Web-service Connector.

Google Web Search

Searching for term "House"

```
@{options[outorder(?title, ?url)]}

?- _restWebserviceAccess(<http://ajax.googleapis.com/ajax/services/search/web>, [->],
["q"->"House", "v"->"1.0", "hl"->"en"], ?jr) and ?jr[_memberByPath(_path
("responseData", "results"))->?results[_member->?result]] and ?result[_memberAt
("unescapedUrl")->?url, _memberAt("titleNoFormatting")->?title].
```

will return the result

title	url
"FOX Broadcasting Company - House TV Show - House TV Series - House ..."	"http://www.fox.com/house/"
"FOX Broadcasting Company - House TV Show - House TV Series - House ..."	"http://en.wikipedia.org/wiki/House_(TV_series)"
"House M.D. (TV Series 2004–) - IMDb"	"http://www.imdb.com/title/tt0412142/"
"United States House of Representatives, 112th Congress, 1st Session"	"http://www.house.gov/"

Another example might be checking the popularity of words, e.g. comparing "absorbtion" with "absorption"

```
p("absorption").
p("absorbtion").
@{options[outorder(?word, ?count), sort(desc(?count))]}
?- p(?word) and _restWebserviceAccess(<http://ajax.googleapis.com/ajax/services/
search/web>, [->], ["q"->?word, "v"->"1.0", "hl"->"en"], ?jr) and ?jr[_memberByPath
(_path("responseData", "cursor", "estimatedResultCount"))->?cs] and _int[_toType(?
cs)->?count].
```

The result shows that "absorption" is far more popular, which might indicate it is orthographically the right variant

word	count
"absorption"	5900000
"absorbtion"	168000

Google Book Search

Searching for all books with using the word "Thyroid":

```
@{options[outorder(?authors, ?title, ?isbn)]}

?- _restWebserviceAccess(<http://ajax.googleapis.com/ajax/services/search/books>, [-
>], ["v"->"1.0", "rsz"->"small", "q"->"Thyroid"], ?jr) and ?jr[_memberByPath(_path
("responseData", "results"))->?books[_member->?book]] and ?book[_memberAt("title")->?
title, _memberAt("authors")->?authors, _memberAt("bookId")->?isbn].
```

Hint: To get more than the standard maximal four search results change the argument `rsz` from `small` to `large`. This will return up to eight results. Have also a look at the official API documentation for additional arguments of the book search service.

The table below shows the results:

authors	titl	isbn
"Mary J. Shomon"	"The Thyroid Diet: Manage Your Metabolism for Lasting Weight Loss"	"ISBN0060524448"
"M. Sara Rosenthal"	"The thyroid sourcebook for women"	"ISBN0071441611"
"Alan L. Rubin"	"Thyroid for Dummies"	"ISBN0471787558"
"Mario Skugor, Jesse Bryant Wilder"	"Thyroid Disorders"	"ISBN1596240210"

DBpedia

Searching DBpedia for the major cities in Baden-Württemberg, using the SPARQL query

```
PREFIX : <http://dbpedia.org/ontology/> SELECT ?name, ?population {?town a :Town; :
federalState <http://dbpedia.org/resource/Baden-W%C3%BCrttemberg>; :populationTotal ?
population; <http://www.w3.org/2000/01/rdf-
schema#label> ?name . FILTER (lang(?name)='en') } ORDER BY DESC(?population) LIMIT 5
```

will result in the OL query

```
@{options[outorder(?city,?population)]}
?- _restWebserviceAccess(<http://dbpedia.org/sparql>, [_mime->"application/sparql-
results+json"], ["query"->"PREFIX : <http://dbpedia.org/ontology/> SELECT ?name, ?
population {?town a :Town; :federalState <http://dbpedia.org/resource/Baden-W%C3%
BCrttemberg>; :populationTotal ?population; <http://www.w3.org/2000/01/rdf-
schema#label> ?name . FILTER (lang(?name)='en') } ORDER BY DESC(?population) LIMIT
5"], ?jr) and ?jr[_memberByPath(_path("results", "bindings"))->?results[_member->?
row]] and ?row[_memberByPath(_path("name", "value"))->?city, _memberByPath(_path
("population", "value"))->?popStr] and _int[_toType(?popStr)->?population].
```

and returns this result:

city	population
"Stuttgart"	600038
"Mannheim"	311142
"Karlsruhe"	288917
"Freiburg im Breisgau"	217547
"Heidelberg"	145642

Freebase

Querying Freebase for all planets of the solar system:

```
@{options[outorder(?planetName)]}
```

```
?- _restWebserviceAccess(<http://api.freebase.com/api/service/mqlread>, [->],
["query"->"""{"query": [{"id": null, "name": null, "type": "/astronomy/
planet"}]}"""], ?jr) and ?jr[_memberAt("result")->?results[_member->?result]] and ?
result[_memberAt("name")->?planetName].
```

Result:

planetName
"Earth"
"Venus"
"Mars"
"Mercury"
"Jupiter"
"Neptune"
"Saturn"
"Uranus"

Facebook

Search on Facebook for public posts for "fukushima":

```
@{options[outorder(?user,?message,?link)]}
?- _restWebserviceAccess(<http://graph.facebook.com/search>, [->], ["q"->"fukushima",
"type"->"post"], ?jr) and ?jr[_memberAt("data")->?data[_member->?post]] and ?post
[_memberByPath(_path("from", "name"))->?user, _memberAt("message")->?message,
_memberAt("link")->?linkStr] and _iri[_toType(?linkStr)->?link].
```

Result:

user	message	link
"Günther Gruchala"	"Einfach alles Flachreden"	<http://www.stern.de/1676195.html>
"Raphael Schwarzhans"	"made by fukushima"	<http://www.youtube.com/watch?v=Jp_XM4IBWn0&feature=share>
"Matthäus Urch"	"jedes monat am 12ten dementsprechnd outfitten, nicht vergessen...."	<http://www.facebook.com/event.php?eid=202283566473179>
"Volker Sessler"	"""U-Bahn -Alltag in Japan ! Jetzt nach Fukushima die neuste Errungenschaft ! Schau's dir an ! Echtes video ! Kein Witz ! """	<http://www.youtube.com/watch?v=kRuTRp4k_MQ&feature=related>
"Ray Mcphoney"	"""hey friends my latest rmx is out this thursday on beatport! here a lil preview: http://soundcloud.com/ce-records/miguelstyle-limited-melody-ray"""	<http://soundcloud.com/ce-records/miguelstyle-limited-melody-ray>
"Nicole Ludwig"	"Beeindruckende Rechnung!"	<http://www.grist.org/list/2011-04-18-what-if-the-152-billion-to-clean-up-fukushima-were-spent-on-geot>
"Neues Deutschland Online"	"Wenig Hoffnung rund um Fukushima / Ein düsteres Szenario beschreibt Thomas Breuer: Innerhalb der Evakuierungszone um den havarierten Atomreaktor und in den angrenzenden Gebieten liegen verlassene Spielplätze und die Bauern können ihre Felder nicht bestellen. Er trifft überall rat- und hilflose Menschen. "	<http://www.neues-deutschland.de/artikel/195821.wenig-hoffnung-rund-um-fukushima.html>
"Barbara Bruhn"	"Herr Sinn: Fukushima IST die Apokalypse!"	<http://www.youtube.com/watch?v=G548Holomuo>
"Eric Ryan Weisz"	"kennt ihr schon das Fukushima-Ei ?"	<http://www.youtube.com/watch?v=oEpj6m_hooI&feature=player_embedded#at=36>

Twitter

Search on Twitter for latest English tweets related to "fukushima":

```
@{options[outorder(?user,?text,?date)]}
?- _restWebserviceAccess(<http://search.twitter.com/search.json>, [->], ["q"-
>"fukushima", "lang"->"en"], ?jr) and ?jr[_memberAt("results")->?results[_member->?
tweet]] and ?tweet[_memberAt("from_user")->?user, _memberAt("text")->?text, _memberAt
("created_at")->?date].
```

user	text	date
"Lavonae818"	"Tepco Plans to End Fukushima Nuclear Crisis by October http://t.co/ZsVkceE"	"Tue, 19 Apr 2011 15:28:44 +0000"
"king_pata"	"やっぱり不思議がられている東電・保安院の対応? Robots Enter Fukushima Reactors, Detect High Radiation http://t.co/FysnFir via @AutomatonBlog"	"Tue, 19 Apr 2011 15:28:38 +0000"
"nichibei"	"Hearts to Ishinomaki: Follow one woman's journey to #Fukushima and #Sendai. Through 4/22. http://on.fb.me/e0PT9O"	"Tue, 19 Apr 2011 15:28:19 +0000"
"aki_kay1987"	"【DVD】《通常配送無料》高山智恵美 / 未成年 [DVD] http://amzn.to/fQAmEf 福島原発事故で活躍されたハイパーレスキュー高山隊長の娘さんのDVDです #DVD #sougofollow #sogofollow #fukushima"	"Tue, 19 Apr 2011 15:28:12 +0000"
"Masakazu_Sakata"	"RT @thoton: URGENT: #Japan may raise legal radiation limit for #Fukushima workers to 1Sv from 250mSv to hasten end of crisis. (jp) http://amba.to/dNIjEI"	"Tue, 19 Apr 2011 15:27:56 +0000"
"htomfields"	"#Video of Idaho National Laboratory Director discussing lessons learned from Three Mile Island and #Fukushima http://youtu.be/vTKdVM53ULU"	"Tue, 19 Apr 2011 15:27:36 +0000"
"kirishathegreat"	"RT @silverstar22b: Fukushima Residents Seek Answers Amid Mixed Signals From Media, TEPCO and Government. Report from the Radiation Exclusio http://tiny.ly/n27p"	"Tue, 19 Apr 2011 15:27:33 +0000"
"chrisorr1"	"RT @fuyuno_sakura: In evacuation area within 20 km radius of FNPP1, about 3,000 cattle, 30,000 pigs and 600,000 chickens were left behind, Fukushima Pref said."	"Tue, 19 Apr 2011 15:27:22 +0000"
"TakMatsunaga"	"It says most important to explain to the international community the current state of the Fukushima Daiichi "with maximum transparency""	"Tue, 19 Apr 2011 15:27:09 +0000"

Yahoo! Weather Forecast

A simple service mashup: First get the WOEID for Karlsruhe using Yahoo's YQL, then use the ID of the highest rated result (Karlsruhe in Baden-Württemberg) to query the Weather Forecast Service.

```
@{options[outorder(?name, ?woeid, ?condition, ?temperature)]}
?- _restWebserviceAccess(<http://query.yahooapis.com/v1/public/yql>, [->], ["q"-
>"select * from geo.places where text=\"Karlsruhe\"", "format"->"json"], ?jr0) AND
?jr0[_memberByPath(_path("query", "results", "place"))->?places] AND
?places[_memberAt(0)->?karlsruhe] AND
?karlsruhe[_memberAt("woeid")->?woeid, _memberAt("name")->?name] AND
_restWebserviceAccess(<http://weather.yahooapis.com/forecastjson>, [->], ["w"->?
woeid, "u"->"c"], ?jr1) AND
?jr1[_memberByPath(_path("condition", "text"))->?condition, _memberByPath(_path
("condition", "temperature"))->?temperature].
```

name	woeid	condition	temperature
"Karlsruhe"	"664942"	"Partly Cloudy"	18.0

To get the weather for all locations matching "Karlsruhe":

```
@{options[outorder(?name,?country,?woeid,?condition,?temperature)]}
?- _restWebserviceAccess(<http://query.yahooapis.com/v1/public/yql>, [->], ["q"-
>"select * from geo.places where text=\"Karlsruhe\"", "format"->"json"], ?jr0) AND
?jr0[_memberByPath(_path("query", "results", "place"))->?places[_member->?place
[_memberAt("name")->?name, _memberByPath(_path("admin1", "code"))->?country,
_memberAt("woeid")->?woeid]]] AND
_restWebserviceAccess(<http://weather.yahooapis.com/forecastjson>, [->], ["w"->?
woeid, "u"->"c"], ?jr1) AND
```

```
?jr1[_memberByPath(_path("condition", "text"))->?condition, _memberByPath(_path
("condition", "temperature"))->?temperature].
```

name	country	woeid	condition	temperature
"Karlsruhe"	"DE-BW"	"664942"	"Partly Cloudy"	18.0
"Karlsruhe"	"DE-SN"	"20173635"	"Partly Cloudy"	20.0
"Karlsruhe"	"DE-BB"	"664941"	"Mostly Cloudy"	21.0
"Karlsruhe"	"US-ND"	"2430747"	"Cloudy"	13.0
"Karlsruhe"	"DE-MV"	"664943"	"Light Rain"	18.0

5.11 SPARQL Connectors

_sparqlaccess(<endpoint>, [default graphs], [named graphs], [map with output])

The lists of default and named graphs can be empty. The output map is: `[context->?C, subject->?S, predicate->?P, object->?O]`.

When working with _sparqlaccess, the SPAQRL query is generated automatically. If you set concrete values instead of variables in the output map, they are used as restrictions for the query. The default graph is imported as default module. Only SPARQL Protocol services with HTTP bindings (REST-like) are supported, no SOAP bindings.

Examples:

```
_sparqlaccess(<http://localhost:8267/services/sparql/query>, [<my:defaultGraph>],
[<my:namedGraph>], [<context->?C,subject->?S,predicate->?P,object->?O>])
```

```
_sparqlaccess(<http://localhost:8267/services/sparql/query>, [], [], [<context->?C,
subject->?S,predicate->?P,object->?O>])
```

```
_sparqlaccess(<http://dbpedia.org/sparql>, [<http://dbpedia.org>], [<http://dbpedia.
org>], [context->?C, subject->?S, predicate-><http://xmlns.com/foaf/0.1/name>,
object->?O])
```

_sparqlquery(<endpoint>, [default graphs], [named graphs], sparql query, [output map])

The lists of default and named graphs can be empty. The output map must reference all variables occurring in the sparql query. Only SELECT queries are supported. Only SPARQL Protocol services with HTTP bindings (REST-like) are supported, no SOAP bindings.

Example:

```
_sparqlquery(<http://localhost:8267/services/sparql/query>, [<my:defaultGraph>],
[<my:namedGraph>], "SELECT * {?s ?p ?o}", [<"?S"->?S, "?P"->?P, "?O"->?O>])
```

5.12 LOD Integration

Linking Open Data (LOD) follows the linked data approach to make datasets available on the web, and define links between them. One of the examples is DBpedia, providing structured access to structured content extracted from Wikipedia. While linked data is not specific about the way the datasets are represented, in Linking Open Data there is a strong focus on established W3C standards like RDF and SPARQL. By providing integration of LOD sources OntoBroker has access to several datasets, containing about 30 billion RDF triples, interlinked by around 500 million RDF links.

How-to Integrate a LOD Source

Similar to the data schema import, first a data source ontology with schema and mapping rules has to be created. The schema describes the structure of the LOD source, while the rules are accessing the source and mapping the data to the ontology schema.

First step in this process is to identify the SPARQL endpoint URL of the LOD source. For example, DBpedia provides this endpoint via http://dbpedia.org/sparql. Most SPARQL endpoints provide an additional web-based user interface to test queries; therefore it's worth trying to open this URL with your browser.

Within the next step the schema and mapping rules are created. Use the script lodimport in the tools directory of OntoBroker for this task. Without any arguments the tool shows an overview of the accepted syntax:

```
usage: lodimport [-d <GRAPH>] -e <ENDPOINT> [-g] [-h] [-i] -l <LABEL> [-m] [-o
<PATH>]
   [-s <GRAPH> | -x <ONTOLOGY>]
 -d,--data-graph <GRAPH>           Data graphs of the LOD source
 -e,--endpoint <ENDPOINT>          LOD endpoint IRI
 -g,--no-global-properties         Do not extract global properties
 -h,--help                         Print this message
 -i,--no-implicit-root-classes     Do not extract implicit root classes
 -l,--label <LABEL>                Label of the LOD source
 -m,--extra-metadata               Extract extra metadata
 -o,--output <PATH>                Path to save the lifting ontologies too; default
is
                                   the current working directory
 -s,--schema-graph <GRAPH>         Schema graphs of the LOD source; not to be used
                                   with –x
 -x,--external-schema <ONTOLOGY>   External schema ontologies; not to be used with -s
```

For the DBpedia example the following configuration provides good results:

```
lodimport -e http://dbpedia.org/sparql -l DBpedia -d http://dbpedia.org
       -s http://dbpedia.org -g -i
```

There are two ontology files created in the output directory: One containing the schema, the other one containing the mapping rules. To test the integration load both ontologies and define an import relation between the rule and the schema ontology, i.e. let the rule ontology import the schema ontology.

Finally, like in any other integration scenario, the data source ontology should be mapped to an appropriate domain ontology. Due to internal performance optimizations it is highly recommend to remove the import relation between the data source schema and rule ontology for the final productive setting.

Known Limitations

- Cycles in class hierarchy might hide parts of the schema in OntoStudio's ontology navigator
- If sampling is used, the resulting schema might be incomplete
- Datatypes used in LOD source have to be mapped for schema definitions to the restricted set used by ObjectLogic; this restriction however is only relevant for the schema definitions
- The system cannot extract a schema from endpoints that use a large implicitly declared class schema and restrictions on query evaluation time or number of returned results
- Extraction of generic properties might add a large amount of properties to the schema; the resulting data source ontology might require manual cleanup

6 Supported Import Formats

ObjectLogic

This is the native knowledge representation language of OntoBroker. ObjectLogic is the successor to F-logic in OntoBroker and inherits many of its features. But the syntax has been reworked to resolve many of the F-logic usability issues. It has an extended data type system, a more natural support for aggregations and many more improvements on F-logic. In the same way as F-logic, it is an object-oriented and rule-based language and has the same well-defined semantics. We describe the syntax for this language in the ObjectLogic reference guide in the OntoBroker installation directory. Ontobroker provides an easy to use F-logic to ObjectLogic converter. For more information click here 39.

RDF(S)

RDF (Resource Description Framework) is a framework for representing metadata. It is a W3C recommendation and provides many representations (see below). RDFS (RDF schema) build upon RDF to provide a simple ontology-like vocabulary for denoting classes, properties, instances and the relations that hold between them. As we use the Sesame RDF parsers (http://www.openrdf.org/) we support different RDF serializations:

- RDF/XML
- N3 (Notation 3)
- NT (N-Triples)
- Turtle

We support all of these formats for the import and export of ontologies. For further details, see the OntoBroker RDF manual.

OWL

The Web Ontology Language (OWL) is also recommended by the W3C and is a full fledged ontology language derived from ontology languages from the field of Description Logics. For further details, see the OntoBroker Professional manual.

7 Query Languages for ObjectLogic

OntoBroker supports multiple query languages. The primary query language (and the native format of OntoBroker) is ObjectLogic. ObjectLogic explicitly supports data types and has an feature-rich and clean syntax. Other query languages are restricted to the features of the query language (for example, disjunctive queries or queries that contain Built-ins, temporary facts, and so on are only supported by ObjectLogic).

7.1 ObjectLogic

ObjectLogic is the primary query language for OntoBroker. It allows all kinds of complex queries. The following query asks for all instances of the concept "Person":

```
?- ?X:Person.
```

It is also possible to start more complex queries which contain disjunctions, negations and built-ins:

```
?- (?X:Person OR ?X:Human) AND NOT ?X[livesIn->Karlsruhe] AND ?X[hasAge->?AGE] AND ?
AGE < 40.
```

For more details on ObjectLogic, please consult the ObjectLogic reference guide in the OntoBroker installation folder.

7.1.1 Query Options

Query options allow you to influence different aspects of a query.

Generic syntax:

```
@{<query name>, options[<option1>,...<optionN>]} <query>
```

7.1.1.1 Handling of NULL Values: fillNULL

Purpose

A ObjectLogic query is only valid if all of its variables are included. If any variable cannot be bound, the bindings of the other variables do not form a result.

Example

Facts:

```
Jerry:Person.
Tom:Person.
Jerry[hasName->"Jerry Mouse"].
```

Query:

```
?- ?X:Person AND ?X[hasName->?Y].
```

Result:

```
X             Y
Jerry         "Jerry Mouse"
```

For some applications this is not the behaviour the user would expect. From database systems they are used to see:

```
X             Y
Jerry         "Jerry Mouse"
Tom            null
```

To enable OntoBroker to fill out the missing results with such a placeholder the query option "fillnull" can be used. This feature fills up the missing variable bindings with a constant "null".

Usage

Query options in ObjectLogic are written as an annotation in front of the query

Example

Query:

```
@{q1, options[fillnull]} ?- ?X:Person AND ?X[hasName->?Y].
```

Result:

```
X                   Y
Jerry          "Jerry Mouse"
Tom            null
```

Restrictions

The query must allow the inferencing algorithm to first of all ground all instances and relations before calling the aggregation.
Example(s):

```
@{q1, options[fillnull]} ?- ?X[hasName->?Y].
```

This will result in an error such as the following:

```
[ERROR]: error on server encountered: com.ontoprise.exception.
LiteralOrderingException: Could not order literals so that
the program is evaluable. This could have the following
reasons: a) a input signature for at least one Built-in could
not be satisfied or b) the program contains unsafe rules
which could not be ordered so that they are top-down evaluable.
```

Reason: The variable ?X cannot be bound with values, so there is no ordering available that the aggregation can be called (hasName is a hidden aggregation when using fillnull).

Using instead:

```
@{q1, options[fillnull]} ?- ?X:Person AND ?X[hasName->?Y].
```

Will deliver the results, as X can be bound with the available "instance of" information. Do not use any built-ins for attribute/relation values:

Example:

```
@{options[fillnull]}?- ?X:C[a->?Y] and _unify(?Y,b).
```

Built-ins can´t cope with the null-value. Only the _filter(?Y,b) Built-in will work.

Fillnull ignores the assigned values of a variable:

```
 @{options[fillnull]}?- p(?Y)  and ?X:C[a->?Y] .
```

The assigned value of ?Y is ignored.

7.1.1.2 Ordering Query Results: outorder

If you want to specify the sequence of the output variables then you can use the "outorder" query option. For example, with the facts

```
Man::Person.
Woman::Person.
```

the output of the query

```
@{q1, options[outorder(?X,?Y)]} ?- ?X::?Y.
```

will be

```
Man,Person
Woman,Person
```

With

```
@{q1, options[outorder(?Y,?X)]} ?- ?X::?Y.
```

the output will be

```
Person,Man
Person,Woman
```

The "outorder" option can also be used for skipping some variable bindings in the result: With

```
@{q1, options[outorder(?X)]} ?- ?X::?Y.
```

the output will be

```
Man
Woman
```

7.1.1.3 Sorting Query Results: sort

This query option allows sorting of the result of the query.

Example:

```
@{q1, options[sort(asc(?X),desc(?Y))]} ?- ?X::?Y.
```

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

```
@{q1, options[sort(?X,desc(?Y))]} ?- ?X::?Y.
```

The ?X will be be interpreted as asc(?X), so that you can just leave out the asc(..).

7.1.1.4 Skip Sending Answers: skipSendingAnswers

This option is useful if you want to measure the pure query time without I/O overhead of sending the results or if you just want to check if a specific query is evaluable.

Example

Ontology:

```
p(a).
p(b).
```

If you execute the following query:

```
?- p(?X).
```

You get the result

```
a
b
```

But if you use "skipSendingAnswers":

```
@{q1, options[skipSendingAnswers]} ?- p(?X).
```

the query will return no result.

7.1.1.5 Limit the Number of Results: limit

If you want to limit the number of answers then you can use the "limit" option. The following example shows the combination of "limit" and "outorder":

```
@{q1, options[limit(1),outorder(?Y,?X)]} ?- ?X::?Y.
```

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.

7.1.1.6 Tracing and Debugging: trace/visualdebug

There are two options for tracing and debugging:

- trace
- visualdebug

Tracing the Evaluation: trace

In order to trace the evaluation process, simply add "trace" to a ObjectLogic query:

```
@{q1, options[trace]} ?- ?X:?Y.
```

The tracing feature is described in more detail in Tracing the evaluation process 177.

Debugging the Evaluation with the Graphical Debugger: visualdebug

Note:

This debugging tool is not officially supported. It has a couple of limitations which need to be considered carefully before using the tool:

- If you use "visualdebug" a GUI opens at the server side. This means that if you do not have access to the server then you basically have to restart OntoBroker before continuing.
- The debugger is deeply integrated in the OntoBroker inference kernel. It is possible that the debugger will crash the OntoBroker server while or after debugging.

To keep a long story short: **Never use this tool on a production server!!**

In order to debug the evaluation process, simply add "visualdebug" to the ObjectLogic query:

```
@{q1, options[visualdebug]} ?- ?X:?Y.
```

The visual debugger is an advanced debugging tool which shows

- The rule graph
- The selected rules
- The output of the different program re writers
- A graphical display of the compiled operatornet
- A step-by-step debugger which shows the data flow in the operatornet

We currently do not have a detailed documentation on this debugger as this tool is used primary by the internal OntoBroker research and development team.

Note 1:

If access control is enabled (i.e. the configuration parameter Security.AccessControl is set to on), you need the special permission "ob:reasoner:debug" to use this query option.

Note 2:

The visual debugger option is not available if you are using the collaboration server. It will be silently ignored.

7.1.1.7 Turn Off Inferencing: inferOff

You may tell the OntoBroker not to do inferencing for query answering. This will prohibit the evaluation of all rules and you will get only the given facts matching your query (only asserted facts are returned, no derived facts):

```
@{q1, options[inferOff]} ?- ?X::?Y.
```

Note:

The behaviour of the query option 'inferOff' has been changed when ontology imports are involved. E.g.

```
@{options[inferOff]} ?- ?X:?Y.<br>
```

will now return asserted facts also from imported modules. If you do not want to receive facts from imported modules you can combine 'inferOff' with 'ignoreImports':

```
@{options[inferOff,ignoreImports]} ?- ?X:?Y.
```

7.1.1.8 Profile: profile

This query option helps you to track down performance problems. It logs some of the statistics on the most expensive rules. This query option is useful for tracking down performance problems. It shows :

- The most expensive builtins and operators (e.g. join operations)
- The most expensive rules

Here is a small example. The following ObjectLogic program calculates the transitive closure of a graph:

```
edge(a,b).
edge(b,c).
edge(c,d).
edge(d,e).
edge(e,f).

@{r1} closure(?X,?Y,f(?Y,f(?X,nil))) :- edge(?X,?Y).
@{r2} closure(?X,?Y,f(?Y,?U)) :- edge(?Z,?Y) AND closure(?X,?Z,?U).
```

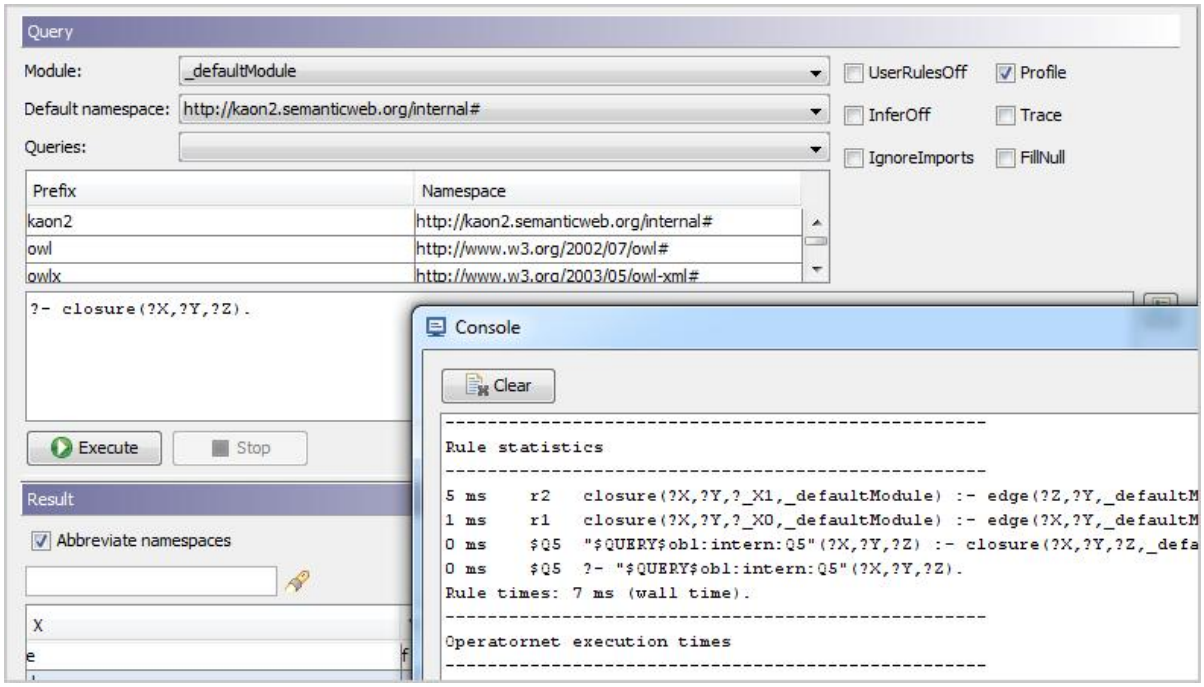
When you execute the query with the "profile" option:

```
@{q1, options[profile]} ?- closure(?X,?Y,?Z).
```

then you get something like:

```
-------------------------------------------------
Preparation(#selected: 3, #final: 4)
-------------------------------------------------
11 ms  PARSER       [OntoBrokerReasoner]           Parse and compile query
9 ms   PREPARATION  [OntoBrokerReasoner]           Compile operatornet
6 ms   PREPARATION  [Evaluator]                    BottomUp preparation
5 ms   REWRITER     [SchemaOptimizer]
4 ms   PREPARATION  [OntoBrokerReasoner]           Selection of required rules
1 ms   REWRITER     [PropagateConstantsRewriter]
... (plus 29 event(s) which took 4 ms)
Preparation time: 42 ms (wall time).
---------------------------------
Most expensive operations
---------------------------------
... (plus 11 operation(s) which took 0 ms)
Operation times: 3 ms (wall time).
---------------------------------
Operator statistics
---------------------------------
bu3joinIDB  5 ms  5 calls    [bu3joinIDB [segment { [closure(?X,?Z,?U,
_defaultModule), "assign"(?_X1,f(?Y,?U))}]]
bu3joinEDB  0 ms  1 call     [bu3joinEDB [edge(?X,?Y,_defaultModule), "assign"(?_X0,f
(?Y,f(?X,nil)))]]
bu3joinEDB  0 ms  1 call     [bu3joinEDB [edge(?Z,?Y,_defaultModule)]]
bu3move     0 ms  1 call     [bu3move [segment { [closure(?X,?Y,?Z,_defaultModule)}]]
bu3move     0 ms  1 call     [bu3move [segment { ["$QUERY$obl:default:q1"(?X,?Y,?
Z)}]]
Operator times: 6 ms (wall time).
-------------------------------------------------
Rule statistics
-------------------------------------------------
5 ms    r2  closure(?X,?Y,?_X1,_defaultModule) :- edge(?Z,?Y,_defaultModule) AND
closure(?X,?Z,?U,_defaultModule) AND "assign"(?_X1,f(?Y,?U)).
0 ms    r1  closure(?X,?Y,?_X0,_defaultModule) :- edge(?X,?Y,_defaultModule) AND
"assign"(?_X0,f(?Y,f(?X,nil))).
0 ms    q1  "$QUERY$obl:default:q1"(?X,?Y,?Z) :- closure(?X,?Y,?Z,_defaultModule).
0 ms    q1  ?- "$QUERY$obl:default:q1"(?X,?Y,?Z).
Rule times: 6 ms (wall time).
-------------------------------------------------
Operatornet execution times
-------------------------------------------------
Wall time: 6 ms.
CPU time: N/A
```

The results above are for "Storage=RAM.TS" and the bottom-up evaluator "BottomUp3". Note that the output for other storage settings and other evaluation methods may be different. The administration console will show the result of "profile" in a separate window:



How to Interpret Profiling Results

The whole point of the "profile" option is to give some ideas on where the evaluation time is spent and to identify the problematic rules. The first section will show the query compilation time. This includes parsing, rewriting and compiling the query.

```
-------------------------------------------------
Preparation(#selected: 3, #final: 4)
-------------------------------------------------
11 ms  PARSER       [OntoBrokerReasoner]           Parse and compile query
9 ms   PREPARATION  [OntoBrokerReasoner]           Compile operatornet
6 ms   PREPARATION  [Evaluator]                    BottomUp preparation
5 ms   REWRITER     [SchemaOptimizer]
4 ms   PREPARATION  [OntoBrokerReasoner]           Selection of required rules
1 ms   REWRITER     [PropagateConstantsRewriter]
... (plus 29 event(s) which took 4 ms)
Preparation time: 42 ms (wall time).
```

The most expensive operation is to parse and to compile the query (11 ms). The total preparation time is 42 ms, which means that there is no special performance issue in the preparation phase. The next section shows the most expensive low-level operations:

```
---------------------------------
Most expensive operations
---------------------------------
... (plus 11 operation(s) which took 0 ms)
Operation times: 3 ms (wall time).
```

Well, at least it would show something if there were some expensive low-level operations (e.g. database accesses or very large joins). But our example ontology is too small and so not much time was spent on low-level operations.
The next section shows the most expensive operators.

```
---------------------------------
Operator statistics
---------------------------------
bu3joinIDB  5 ms  5 calls    [bu3joinIDB [segment { [closure(?X,?Z,?U,
_defaultModule), "assign"(?_X1,f(?Y,?U))}]]
bu3joinEDB  0 ms  1 call     [bu3joinEDB [edge(?X,?Y,_defaultModule), "assign"(?_X0,f
(?Y,f(?X,nil)))]]
bu3joinEDB  0 ms  1 call     [bu3joinEDB [edge(?Z,?Y,_defaultModule)]]
```

```
bu3move     0 ms  1 call     [bu3move [segment { [closure(?X,?Y,?Z,_defaultModule)}]]
bu3move     0 ms  1 call     [bu3move [segment { ["$QUERY$obl:default:q1"(?X,?Y,?
Z)}]]
Operator times: 6 ms (wall time).
```

Operators are typically

- Join operations
- Match operations
- Built-ins 227
- Connectors

This means when you have an expensive connector to an external database it will show up in this list. The next section is probably the one on that you should look at first. It shows the most expensive rules of the evaluation:

```
-----------------------------------------------------
Rule statistics
-----------------------------------------------------
5 ms    r2  closure(?X,?Y,?_X1,_defaultModule) :- edge(?Z,?Y,_defaultModule) AND
closure(?X,?Z,?U,_defaultModule) AND "assign"(?_X1,f(?Y,?U)).
0 ms    r1  closure(?X,?Y,?_X0,_defaultModule) :- edge(?X,?Y,_defaultModule) AND
"assign"(?_X0,f(?Y,f(?X,nil))).
0 ms    q1  "$QUERY$obl:default:q1"(?X,?Y,?Z) :- closure(?X,?Y,?Z,_defaultModule).
0 ms    q1  ?- "$QUERY$obl:default:q1"(?X,?Y,?Z).
Rule times: 6 ms (wall time).
```

In this example the rule "r2" is the most expensive rule because it will be executed multiple times (it is a recursive rule).

7.1.1.9 Ignore Imports: ignoreImports

The query option ignoreImports allows you to run a query without the import rules. Import rules normally map facts and rules from the imported module into the importing module.

Example

Ontology m1:

```
:- module = m1.
A[name=>string].
```

Second Ontology:

```
:- module = _defaultModule.
:- import m1.
B::A.
```

The query

```
?- B[?P *=> ?R]@_defaultModule.
```

returns one result (P=name, R=string).

But the query

```
@{q1, options[ignoreImports]} ?- B[?P *=> ?R].
```

returns **no** result.

7.1.1.10 Distinct: distinct

The elimination of duplicates from queries in ObjectLogic can be set globally by the boolean configuration option 'EliminateDuplicates' (default: on). If set to 'off', the query option 'distinct' can be used for individual queries to activate the elimination of duplicates.
This query option allows you to eliminate duplicates for each query.

```
@{q1, options[distinct]} ?- ?X::?Y.
```

7.1.1.11 Offset: offset

If you want to skip the first n results then you can use the "offset" option:

```
@{q1, options[offset(10)]} ?- ?X::?Y.
```

In this example, only the answers starting from the 10th result are returned.

7.1.1.12 Maximal Query Evaluation Time: timeout

Sometimes you will want to stop a query after a specific amount of time. The "timeout" query option allows you to do this:

```
@{q1, options[timeout(2000)]} ?- ?X::?Y.
```

This query will be stopped after 2 seconds.

7.1.1.13 Tracing and Profiling: profile/trace

OntoBroker will log some information about the query and the performance characteristics if you use the query options "profile" and "trace".

```
@{q1, options[profile,trace]} ?- ?X::?Y.
@{q1, options[profile]} ?- ?X::?Y.
@{q1, options[trace]} ?- ?X::?Y
```

7.1.1.14 Set Evaluation Method Single Query

Usually you specify

- `EvaluationMethod = choose`

in your OntoConfig file, so that OntoBroker can decide which evaluation strategy to use for a query. But some specific queries might be faster if you select another evaluation strategy:

```
@{q1, options[EvaluationMethod(BottomUp)} ?- p(?X).
```

7.1.1.15 Provenance Option for Queries: connectorprovenance

This option for queries returns provenance information on how the answers depend on the facts in connected information sources. A list of such provenance information is given for each answer. This list provides the access literals and the variable substitutions for these access literals.

Let's have a look at a simple example.The following query requests the employee id and the last name of employees in the "Employees" table of the database "data:1433_Northwind_test"

```
?- _dbaccessuser("dbo"."Employees","F"("EmployeeID",?VAR1,"LastName",?
VAR2),"data:1433_Northwind_test").
```

We get answers like

```
?VAR1= 1, ?VAR2 = "Davolio ",
?VAR1= 9, ?VAR2 = "Dodsworth",
?VAR1= 8, ?VAR2 = "Callahan"
```

If we add the connector provenance option to the query:

```
@{options[connectorprovenance(?Z)]} ?- _dbaccessuser
("dbo"."Employees","F"("EmployeeID",?VAR1,"LastName",?
VAR2),"data:1433_Northwind_test").
```

We get the variable ?Z bound to a list of instantiations of the _dbaccessuser predicate:

```
?VAR1= 1, ?VAR2 = "Callahan", ?Z = [_dbaccessuser
("dbo"."Employees","F"("EmployeeID",1,"LastName","Davolio"),"data:1433_Northwind_test
")] ?VAR1= 9, ?VAR2 = "Dodsworth", ?Z = [_dbaccessuser
("dbo"."Employees","F"("EmployeeID",9,"LastName","Dodsworth"),"data:1433_Northwind_te
st")]
```

```
?VAR1= 8, ?VAR2 = "Callahan", ?Z = [_dbaccessuser
("dbo"."Employees","F"("EmployeeID",8,"LastName","Callahan"),"data:1433_Northwind_tes
t")]
```

Indicating that id = 1, last name = "Callahan", id = 9, last name = "Dodsworth", and id = 8, last name = "Callahan" has been found in the database which lead to our answers.

7.1.1.16 Query option explain

For further details click here[179].

7.1.1.17 Query option userRulesOff

The query is executed, but user rules are not considered. This basically means that all rules which are created directly (they are typically shown in the Rules folder in OntoStudio) are not considered.

```
@{q1, options[userRolesOff]} ?- ?X:?Y.
```

7.1.1.18 Query option reduced

This allows the partial duplicate elimination from the results of a SPARQL query. For more details click here.

7.1.1.19 Query option profileAll

`profileAll` will log profiling-related output during query execution. The output will contain executed joins etc. Note that the output format and detail granularity is evaluation-method specific.

```
@{q1, options[profileAll]} ?- ?X::?Y.
```

7.2 SPARQL

SPARQL is a W3C standard. It is a query language for RDF (SPARQL: SPARQL Protocol And RDF Query Language). Please note that OntoBroker only implements a subset of SPARQL (and SPARQL only supports a subset of the ObjectLogic query features). The following table shows how we interpret SPARQL queries in OntoBroker for ObjectLogic ontologies:

Subject	Predicate	Object	ObjectLogic	Comment
?S	rdf:type	rdfs:Class	_concepts(?S)	All classes
?S	rdf:type	rdf:Property	_properties(?S)	All properties (i.e. attributes and relations)
?S	rdf:type	?O	?S:?O AND ...	Instances (including instances of meta-classes rdfs:Class and rdf:Property)
?S	rdfs:subClassOf	?O	?S::?O	Sub-classes
?S	rdfs:subPropertyOf	?O	?S<<?O	Sub-properties
?S	rdfs:range	?O	_PropertyRange(?S,?O)	RDFS domain (not part of OL)
?S	rdfs:domain	?O	_PropertyDomain (?S,?O)	RDFS range (not part of OL)

Some examples are given below. The first example is a simple query with two triples in the WHERE part of the SPARQL query.

```
SELECT *
WHERE {
        ?CAR a <http://example.com/#car> ;
             <http://example.com/#price> ?PRICE
}
```

This SPARQL query could also be expressed using the ObjectLogic query

```
?- ?CAR:<http://example.com/#car> AND ?CAR[<http://example.com/#price>->?PRICE].
```

This example shows a query for subclass relationships. When we have the ObjectLogic ontology:

```
Developer::Employee.
ProjectManager::Employee.
```

Then the following query

```
SELECT ?X
WHERE {
        ?X rdfs:subClassOf ?Y
}
```

will return:

```
X
--------------
Developer
ProjectManager
```

This SPARQL query could also be expressed with the ObjectLogic query

```
@{options[outorder(?X)]} ?- ?X::?Y.
```

8 Run Queries by Query Id

It is possible to run a query by query id.

The query

```
?- _query_(q1)@m2.
```

executes the query q1 of module m2.

If you provide query options, they overwrite the original query options, i.e. it is possible to query

```
@{options[profile]} ?- _query_(q1)@m2.
```

Moreover this option subsitutes parts of queries. You can write

```
_query_(q1) and ?X is a.
```

Expressions like

```
?- _query_(q2)@m2 , ?X < 5.
```

are also possible.

9 RIF

Definition:

„The RIF Framework for Logic Dialects (RIF-FLD) is a formalism for specifying all logic dialects of RIF, including the RIF Basic Logic Dialect [RIF-BLD] and [RIF-Core] (albeit not [RIF-PRD], as the latter is not a logic-based RIF dialect). RIF-FLD is a formalism in which both syntax and semantics are described through a number of mechanisms that are commonly used for various logic languages, but are rarely brought all together. The amalgamation of several different mechanisms is required because the framework must be broad enough to accommodate several different types of logic languages and because various advanced mechanisms are needed to facilitate translation into a common framework. RIF-FLD gives precise definitions to these mechanisms, but allows well-defined aspects to vary. The design of RIF envisions that future standard logic dialects will be based on RIF-FLD. Therefore, for any RIF dialect to become a standard, its development should start as a specialization of FLD and extensions to (or, deviations from) FLD should be justified."

Source: http://www.w3.org/TR/rif-fld/

9.1 Overview

OntoBroker supports RIF/XML as import and export format for ObjectLogic. The provided format handler parses a given RIF/XML-based ontology and imports its content into an ObjectLogic ontology and, vice versa, an ObjectLogic ontology can be saved as RIF/XML. Please note that not all RIF constructs introduced by FLD can be mapped to ObjectLogic.

9.2 ObjectLogic -> RIF

9.2.1 Mapping

The following table shows a (partial) mapping T from ObjectLogic constructs to their RIF/XML representations. The XML entity "rif" refers to the RIF namespace "http://www.w3.org/2007/rif#" and the XML entity and namespace prefix "obld" refer to the namespace "http://www.ontoprise.com/2009/03/rif/obld#".

ObjectLogic	RIF/XML
instance:class	<Member> <instance>T(instance)</instance> <class>T(class)</class> </Member>
object[]	<Frame> <object>T(object)</object> </Frame>
object[property->value]	<Frame> <object>T(object)</object> <slot ordered="yes"> T(property) T(value) </slot> </Frame>
object[property]	<Frame> <object>T(object)</object> <slot ordered="yes">

	T(property) <Const type="&rif;iri">&obld;boolean</Const> </slot> </Frame>
subclass::superclass	<Subclass> _{T(subclass)} <super>T(superclass)</super> </Subclass>
subclass<<superclass	<obld:Subproperty> <obld:sub>T(subclass)</obld:sub> <obld:super>T(superclass)</obld:super> </obld:Subproperty>
class[property{min:max} *=>()]	<Atom> <op><Const type="&rif;iri">&obld;propertyCardinality</Const></op> <args ordered="yes"> T(class) T(property) T(min) T(max) </args> </Atom>
class[property*=>range]	<Atom> <op><Const type="&rif;iri">&obld;propertyRange</Const></op> <args ordered="yes"> T(class) T(property) T(range) </args> </Atom>
class[property=>range]	<Atom> <op><Const type="&rif;iri">&obld;nonInheritablePropertyRange</Const></op> <args ordered="yes"> T(class) T(property) T(range) </args> </Atom>
class[property{inverseOf (inverseProperty)}	<Atom> <op><Const type="&rif;iri">&obld;propertyInverseOf</

*=>inverseDomain]	Const></op> <args ordered="yes"> T(class) T(property) T(inverseProperty) T(inverseDomain) </args> </Atom>
class[property {characteristic}*=>()]	<obld:PropertyCharacteristic> <obld:object>T(class)</obld:object> <obld:property>T(property)</obld:property> <obld:characteristic><Const type="&rif;iri">obl:reserved: characteristic</Const></obld:characteristic> </obld:PropertyCharacteristic>
F AND G	<And> <formula>T(F)</formula> <formula>T(G)</formula> <And>
F OR G	<Or> <formula>T(F)</formula> <formula>T(G)</formula> <Or>
NOT F	<Naf> <formula>T(F)</formula> <Naf>
EXIST v1, ..., vN F	<Exists> <declare>T(v1)</declare> ... <declare>T(vN)</declare> <formula>T(F)</formula> </Exists>
FORALL v1, ..., vN F	<Forall> <declare>T(v1)</declare> ... <declare>T(vN)</declare> <formula>T(F)</formula> </Forall>
F :- G	<Implies> <if>T(G)</if> <then>T(F)</then>

	</Implies>
F <-- G	<Implies> <if>T(G)</if> <then>T(F)</then> </Implies>
F --> G	<Implies> <if>T(F)</if> <then>T(G)</then> </Implies>
F <--> G	<obld:Equivalent> <obld:formula1>T(F)</obld:formula1> <obld:formula2>T(G)</obld:formula2> </obld:Equivalent>

?- F	<Implies>
	  <if>T(F)</if>
	  <then>
	  <Atom>
	  <op><Const type="&rif;iri">&obld;query/arity</Const></op>
	  <args ordered="yes">
	  T(id)
	  T(v1)
	  ...
	  T(vN)
	  </args>
	  </Atom>
	  </then>
	</Implies>
	where id is the id of "?- F" (or some placeholder constant if the formula has no id), v1,..., vN are the free variables in F and arity is N + 1. Query options are serialized as annotations (not presented here).

!- F	<Implies>
	  <if>T(F)</if>
	  <then>
	  <Atom>
	  <op><Const type="&rif;iri">&obld;constraint/arity</Const></op>
	  <args ordered="yes">
	  T(id)
	  T(v1)

	... T(vN) </args> </Atom> </then> </Implies> where id is the id of "!- F" (or some placeholder constant if the formula has no id), v1,..., vN are the free variables in F and arity is N + 1.
?X	<Var>X</Var>
"lexicalValue"^^<datatypeURI>	<Const type="datatypeURI">lexicalValue</Const>
[t1, t2, ..., tN]	<Expr> <op><Const type="&rif;iri">&obld;list/2</Const></op> <args ordered="yes"> T(t1) <Expr> <op><Const type="&rif;iri">&obld;list/2</Const></op> <args ordered="yes"> T(t2) ... <Expr> <op><Const type="&rif;iri">&obld;list/2</Const></op> <args ordered="yes"> T(tN) <Const type="&rif;iri">&obld;list/0</Const> </args> </Expr> ... </args> </Expr> </args> </Expr>
[t1->v1, t2->v2, ..., tN->vN]	<Expr> <op><Const type="&rif;iri">&obld;map/1</Const></op> <args ordered="yes"> T([t1,v1_1,...,t1,v1_M1,t2,v2_1,...,t2,v2_M2,...,tN,vN_1,...,vN_MN])

	</args> </Expr> where {vi_j | j=1,...,Mi}, i = 1,...,N, is the set of values for key ti

9.2.2 Annotations and Formula IDs

ObjectLogic formula IDs are mapped to RIF annotation IDs. While ObjectLogic allows functional terms as formula ID, RIF annotation IDs must be constants of type rif:iri. Any constant used as ObjectLogic formula ID not of type rif:iri will be encoded with a special URI while functional terms will be formatted as a string and handled as if they would be an xsd:string constant with the corresponding value.

9.2.3 Forall-Closing

If a top level ObjectLogic formula F has free variables, the transformation forall-closes the transformed formula. If F has an id or annotations, they apply to the forall-closed formula. On the other hand, the re-transformation removes unnecessary forall-quantifiers on the top level of formulas and maps the id and annotations to the resulting formula.

9.2.4 Modules

ObjectLogic separates the knowledge base of ontologies explicitly using modules. If an ObjectLogic formula contains a module definition, it will be transformed to a special annotation.

ObjectLogic	RIF/XML

…F@module…	<tag>
	<meta>
	<Frame>
	<object><Const type="&rif;iri">&obld;this</Const></object>
	<slot ordered="yes">
	<Const type="&rif;#iri">&obld;module</Const>
	T(module)
	</slot>
	</Frame>
	</meta>
	...
	</tag>
	
	where tag is the main tag of T(F).

Indirectly Supported ObjectLogic Features

The following constructs are indirectly supported only by using special positional term encodings. Currently no axiomatizing formulas are included in the exported ontologies, in fact not all of them can be axiomatized.

- Boolean properties (boolean FPropertyMember), e.g. a[p].
- Property cardinalities (FPropertyCardinality), e.g. A[p{min:max}*=>()].
- Property range (FPropertyRange), e.g. A[p*=>B], A[p=>B], A[*=>p] and A[=>p].
- Inverse properties specification (FPropertyInverseOf), e.g. A[p{inverseOf(q)}*=>B].
- Symmetric property specification (FPropertySymmetric), e.g. A[p{symmetric}*=>()].
- Transitive property specification (FPropertyTransitive), e.g. A[p{transitive}*=>()].
- Queries (Query), e.g. ?-P(?X).
- Constraints (Constraint), e.g. !-P(?X).
- Lists (special terms), e.g. [1, 2, 3].

9.2.5 Unsupported ObjectLogic Features

Built-ins are currently not supported. This includes aggregates as a special form of built-ins.

9.3 RIF -> ObjectLogic

9.3.1 Allowed RIF Formulas

Only the kind of RIF formulas are allowed which can be obtained by applying the ObjectLogic to RIF/XML transformation on a valid ObjectLogic formula. For example, disjunctions are allowed only in implication conditions since a disjunction in a rule head or a fact is disallowed in ObjectLogic. As "syntactic sugar" groups are allowed but are mapped to conjunctions. rif:local constants may be used but are handled in a special way, see below.

9.3.2 Partially Supported RIF Features

Some features are supported only partially.

- Groups are mapped to conjunctions.
- Constants of datatype `rif:local` which are used as predicate symbols are mapped to names including the document ID information, i.e. `"lexicalValue"@<documentId>`, where `lexicalValue` is the lexical value of the rif:local constant and `documentId` is the lexical value of the document id. rif:local Constants used in (non formula) term positions are mapped to ObjectLogic blank nodes, note that the lexicalValue is not preserved.
- Frames with more than one property/value pair are split into a conjunction of frames containing only one property/value pair each.
- The datatypes xsd:dayTimeDuration and xsd:yearMonthDuration are mapped to datatype xsd:duration.
- Positional terms used as functional terms with arity zero are mapped to the constant used as the leading symbol. E.g. "f"^^xsd:string() is mapped to "f"^^xsd:string.
- The leading term of an atomic positional term must be a constant. The import maps atomic positional terms to literals. Since the OntoBroker API only allows strings for the predicate symbol in literals, some special encodings of the leading constant are used to map them to strings.

9.3.3 Unsupported RIF Features

The following RIF constructs are not supported.

- Imports with a profile.
- DTB built-ins and aggregations, only ObjectLogic ones are supported.
- RIF lists terms.
- RIF remote term references and formulas.
- Terms with named arguments.
- Annotations on non top level formulas.
  An exception is the special annotation for ObjectLogic modules.
- Annotations which are not a frame or a group or conjunction of frames.
- Non constant terms as a leading term within a positional term.
- Equality as a fact or within the conclusion of a top level implication.

These RIF terms must be used in formula positions only.

- Equality.
- Membership.
- Subclass.
- Frame.

These RIF terms must not be used backwards in formula positions:

- Constant.

- Variable.

9.4 API

9.4.1 Import

Just open a given RIF/XML ontology as if it would be a native ObjectLogic ontology:

```
OntologyManager oblOntologyManager = ...;
oblOntologyManager.openOntology(ontologyURI, parameters);
```

The ontology URI in a RIF ontology is identified by the annotation ID of the document formula. If it has none, the physical URI is used as the ontology URI. As usual; if necessary you will have to register an ontology to a physical URI mapping in the ontology resolver of the ontology manager.

Alternatively use

```
oblOntologyManager.importOntology(new File(physicalURI), ...);
```

to import a RIF/XML ontology without the need to register an ontology to a physical URI mapping.

9.4.2 Export

Save an ObjectLogic ontology as RIF/XML using the corresponding file format handler identified by OntoBrokerOntologyFileFormat.RIF_OBLD_XML_FOR_FLOGIC2, e.g.:

```
oblOntology.saveOntology(OntoBrokerOntologyFileFormat.RIF_OBLD_XML, file);
```

10 Project Files

OntoBroker supports project files for easier handling of multiple ontologies. A project file specifies:

- What ontology files are loaded
- In which module ontology files are loaded

The project files have a very simple syntax with the two tasks "load" and "resolve" (for more information see "Load Save 199") which we explain below using some examples. If you want to use project files, start OntoBroker with -project <projectfile>.

Additional Information on the "resolve" Task:

When using the resolve task for a data path you should use a slash (/) and not a backslash (\).

Use

```
resolve <http://www.ontoprise.com> = "myproject/testontology.obl"
```

instead of

```
resolve <http://www.ontoprise.com> = "myproject\\testontology.obl"
```

This has the advantage that you can use project files on every operating system (Linux, Windows).

If you use literal backslashes they have to be escaped.

For example, you can write:

```
resolve <http://www.vector.com> = "project\\vector_bus_system.obl"
```

or (better as the project files are then also usable under Linux)

```
resolve <http://www.vector.com> = "project/vector_bus_system.obl"
```

Note that the escaping also allows to specify file names such as

```
resolve <http://www.vector.com> = "project/vector\"_bus_system.obl"
```

10.1 Basic Examples

Load multiple ontology files:

```
load "file1.obl" // module m1
load "file2.obl" // module m2
```

Note:
All characters in a line after // are comments.

Load and materialize multiple ontology files:

```
/*
multiline-comment
*/
load -m "file1.obl"
```

or

```
load -materialize "file1.obl"
Load a ontology file (with specification of the file format):
load -obl "file1.obl"
```

or

```
load -objectlogic "file1.obl"
```

Load and materialize an ontology file:

```
load -m -obl "file1.obl"
```

10.2 Loading an Ontology File into a Specific Module

This feature is especially useful for RDF. In the RDF case we will have one ontology per ontology file. But it is quite common to split large ontology files into multiple smaller files. We could improve performance significantly if we would put the content of all of these files into a single ontology:

```
load "rdf1.ttl" into <m1>
load "rdf2.ttl" into <m1>
```

Then both files are put into the same module.

10.3 Performance Optimization with Project Files

Typically each ontology file will be placed into its own module. For performance reasons it is often better to put multiple ontologies into the same module. Example: We have two files:

File 1:

```
:- module=m1.
:- import=m2.
B::A.
```

File 2:

```
:- module=m2.
C::A.
```

If we write

```
load "file1.obl" // module m1
load "file2.obl" // module m2
```

in the project file then we have two ontology objects and the extensional database contains the facts

```
B::A@m1.
C::A@m2.
```

If it is not required to access "m2" directly, then it is better to write

```
load "file1.obl" // m1
resolve <m2> = "file2.obl" // m2
```

In this case we have

```
B::A@m1.
C::A@m1.
```

in the extensional database (which will provide better performance).

11 Materialization

What is "materialization"? Materialization always means that OntoBroker evaluates rules in advance (on startup) and stores the results in the extensional database (EDB). This has the following effects:

- Queries can be executed very fast (as OntoBroker has fewer rules to evaluate).
- OntoBroker needs more memory (as all facts defined implicitly by rules are stored in the database).
- The startup time increases (sometimes significantly because all of the reasoning is done in the startup phase).

So to keep it short: Materialization is a very good way to optimize performance and to provide guaranteed answering times. Another important use case for materialization is to materialize database mapping rules. This way the users of the ontology can execute offline queries (without access to the mapped database).

Note:
Materialization is only supported for the ontology langugages ObjectLogic and RDF. Currently OWL ontologies cannot be materialized.

11.1 Basic Usage

OntoBroker will materialize all rules in a file if it is loaded via

```
OntoBroker32.exe -m "someRulesToBeMaterialized.obl"
```

If you have several rules but you only want to materialize some of them, you could put the rules to be materialized in a separate file.

11.2 Standard Materialization

The standard materialization will materialize rules and store the derived facts in the datamodel. It is not possible to update the materialization at a later point of time. The materialized rules are never again used for reasoning. This means that if you added facts to the knowledge base after materialization, OntoBroker was not able to detect that the materialized rules should be applied again. This means that when your knowledge base changes, you basically have to restart OntoBroker and materialize the rules again (this time with the new facts).

Tip:
If you want to store the materialized facts in a file, just start OntoBroker with the -m option and execute the "exportAllModules" command. OntoBroker will dump all facts and rules of all ontologies of the knowledge base to multiple files.

Tip:
You can use the new "-runscript" commandline option when starting the server to materialize a set of rules and store the resulting ontologies in a file. This allows you to materialize rules in a batch job.

11.3 Updatable Materialization

Since version 5.3 OntoBroker supports updatable materialization. This means that when you add facts or rules after materialization, the materialized rules are re-evaluated together with the changed facts and rules. Technically this works by separating the base facts from the derived facts.

An example: If you have

```
Tom:Cat.
RULE r1: ?X:"Predator" :- ?X:"Cat".
```

and you want to materialize rule "r1" then the materialized fact

```
Tom:Predator.
```

will not be stored in the table "instanceOf", but instead it will be stored in the table "MAT_instanceOf".

This way OntoBroker can differentiate between base facts and materialized facts. Hence it is possible to revert and update the materialization. If you want the materialization to be updatable then you have to materialize via

```
OntoBroker32.exe -m:update someRulesToBeMaterialized.obl
```

or via

```
start-ontobroker.sh -m:update someRulesToBeMaterialized.obl
```

on Unix-based systems.

If you don't want to update the materialization you materialize the rules this way:

```
OntoBroker32.exe -m:noupdate someRulesToBeMaterialized.obl
```

or

```
OntoBroker32.exe -m someRulesToBeMaterialized.obl
```

In this case the materialized facts will be stored in the same tables as the base facts. So it is no longer possible to differentiate between the two types of facts.

Materialization strategies

OntoBroker supports different materialization strategies. The "DRed" strategy works incrementally, the "RFS" strategy will update the knowledge base from scratch. Which strategy will deliver better performance depends on your rules and the change rate.

DRed (Delete and Rederive)

This materialization strategy works incrementally. It creates a maintenance program which aims to re-evaluate only the rules which are affected by the changes. Note that there are cases where this approach does not very well. A typical example is when you have many connected rules and simple changes lead to a re-evaluation of most rules. We also experienced problems when the number of rules is too large.

Note:
This materialization strategy will delay changes to the ontology until the materialization is triggered again. This means if you add a fact then queries and requests for this fact will only return the expected result after the materialization was executed again.

The DRed algorithm is a well-known materiallization algorithm (original paper: "Maintaining Views Incrementally" by Gupta, Mumick, Subrahamanian). We use a pure declarative version of the algorithm. Basically we rewrite rules like

```
p(?X,?Z) :- q(?X,?Y) AND r(?Y,?Z).
```

to the maintenance rules

```
p_plus(?X,?Z) :- q_insert(?X,?Y) AND r(?Y,?Z).
p_plus(?X,?Z) :- q(?X,?Y) AND r_insert(?Y,?Z).
```

Then we execute queries on the "p_plus" predicates and add the results to the "p" table of the EDB.

RFS (Rebuild From Scratch)

This strategy works very simple: If a change occurs all materialized facts are thrown away. Then all rules are materialized again. This simple strategy works especially well if

- the initial materialization is very fast
- changes do not occur very often
- changes often affect many rules

If you decided you want updatable materialization you have to find out which update strategy matches your needs. Currently we support two update strategies: triggered updates and immediate updates.

Note:
Currently bulk change operations are not supported.

Update strategies

Immediately

This update strategy will execute the materialization as soon immediately after a fact or rule was added or removed.

Triggered

This update strategy will execute the materialization only when the "mat_executeMaterialization 87" command was executed. A typical scenario is to update the materialization at some time where no one accesses the server (e.g. once a night).

11.4 Materialization Commands

This section describes how to manage the materialization using server commands. The commands can be executed remotely (e.g. via the webconsole 36) or via the API 92 (OntologyManager.execute()).

Function	Syntax	Description / Example

Add Rule as Materialized Rule (For Updatable Materialization Only)	`mat_addRule <ruleID>, <ontologyID>`	To materialize the rule
		`@{r1} ?X[enemyOf->Jerry] :- ?X:Cat AND ?X[hasName->Tom].`
		of ontology "comics" you have to execute the command
		`mat_addRule r1, comics`
		If you use the API you can write
		`OntologyManager manager; manager.execute ("mat_addRule r1 comics");`

Remove Rule as Materialized Rule (For Updatable Materialization Only)	`mat_removeRule <ruleID>, <ontologyID>`	This command will remove the rule with the given ID as a materialized rule. Note that the rule still remains in the ontology. The materialized facts for this rule will be removed.
Execute the Materialization (For Updatable Materialization Only)	`mat_executeMaterialization`	This command will trigger an immediate update of the materialization.

Materialize rules (for not revertable/not updatable materialization)	`mat_materializeRules <ruleIDs>, <ontologyID>`	This command works similar to the "mat_addRule" command, but the materialization of these rules can not be updated again.
		This example shows how to materialize a single rule:
		`mat_materializeRules r1, comics`
		You have to specify at least one rule ID. To materialize multiple rules you can execute
		`mat_materializeRules r1, r2, r3, r4, comics`

Materialization Status	`mat_status`	This command will return a short status about the materialized rules, and so on. The output will look like this:
		`# of facts in the EDB: 132`
		`# of rules in the IDB: 22`
		`Materialization strategy: RFS`
		`Materialized rules:`
		`@{r1} ?X[enemyOf->Jerry] :- ?X:Cat AND ?X[hasName->Tom].`

11.5 Tracing the Materialization

During materialization some information about the materialization progress is shown. For debugging purposes it might be useful to turn on additional tracing:

```
Trace.Materialization = on
```

The materialization monitor logs out information about the materialization process and about the updates.

11.6 All Configuration Options

Configuration Option	Possible values / results	Code
Materialization.Updatable	Possible values: "on" and "off". This option will turn on or off the updatable materialization.	`Materialization.Updatable = on` `Materialization.Updatable = off`
Materialization.Strategy	Possible values: "DRed" and "RFS"	`Materialization. Strategy=DRed` `Materialization. Strategy=RFS`
Materialization.UpdateStrategy	Possible values: "Triggered" and "Immediately"	`Materialization. UpdateStrategy=Triggered` `Materialization. UpdateStrategy=Immediately`
Trace.Materialization	Possible values: "on" and "off". This option will show more information about the materialization progress.	`Trace.Materialization = on` `Trace.Materialization = off`

11.7 Materialization with the API

The following code shows how to set up a new ontology manager with some materialization options:

```
Properties properties = new Properties();
properties.setProperty(IConfig.MAT_UPDATABLE_KEY, IConfig.ON);
properties.setProperty(IConfig.MAT_STRATEGY_KEY, IConfig.MAT_STRATEGY_RFS);
properties.setProperty(IConfig.MAT_UPDATE_STRATEGY _KEY,
                       IConfig.MAT_UPDATE_STRATEGY_TRIGGERED);
OntologyManager manager = KAON2Manager.newOntologyManager(properties);
Then you can access the materialization manager by calling
MaterializationManager matManager = manager.getMaterializationManager();
```

This manager allows you to add rules to the materialization, to update the materialization etc:

```
matManager.materializeRule(myOntology, myRule, true); // materialize rule
matManager.materializeRule(myOntology, myRule, false); // remove from materialization
matManager.isRuleMaterialized(myOntology, myRule); // check materialization of rule
```

It is also possible to materialize standard axioms:

```
matManager.materializeStandardAxiom(StandardAxiom.flogicSubclassTransitivity);
```

11.8 Materialization of Standard Axioms

As well as user-defined rules it is also possible to materialize the built-in standard axioms. Standard axioms are special rules which are used to, for example, derive the subclass transitivity of ObjectLogic or RDF. Often, materializing the standard axioms will significantly improve performance.

Materializing ObjectLogic Standard Axioms

If you want to materialize the ObjectLogic standard axioms you have to turn on the corresponding switch in your OntoConfig.prp. The following standard axioms can be materialized:

- Attribute inheritance (Materialize.ObjectLogic.AttributeInheritance)
- Attributes (Materialize.ObjectLogic.Attributes)
- Concepts (Materialize.ObjectLogic.Concepts)

- Relations (Materialize.ObjectLogic.Relations)
- Root concepts (Materialize.ObjectLogic.RootConcepts)
- Subclass transitivity (Materialize.ObjectLogic.SubclassTransitivity)
- Subproperty transitivity (Materialize.ObjectLogic.SubpropertyTransitivity)
- Subset relationship for concepts (Materialize.ObjectLogic.SubsetRelationship)
- Subset relationship for properties (Materialize.ObjectLogic.PropertySubsetRelationship)
- Structural inheritance for instances (Materialize.ObjectLogic.StructuralInheritanceForInstances)

For example, if you want to materialize the subclass transitivity standard axiom you have to put in your OntoConfig

```
Materialize.ObjectLogic.SubclassTransitivity = on
```

As a default, no ObjectLogic standard axiom is materialized on startup.

Materializing RDF/S Standard Axioms

When you use the ontology language RDF, you can materialize the RDF standard axioms which are described in the chapter which describes RDF inferending (Inferencing) [19]. For example, if you want to materialize the class transitivity then you add the following line to your OntoConfig.prp:

```
Materialize.RDF.Entailment.ClassTransitivity = on
```

As a default, no RDF standard axiom is materialized on startup.

12 Interfaces to OntoBroker

The OntoBroker server can be accessed from multiple programming languages and platforms:

- Java 92
- Microsoft .NET 109
- Web service 109

You can execute OntoBroker queries and commands via the standard webservice interface from all programming platforms that have some kind of webservices library. The following chapters give more details and some examples on how to do this. It is also possible to embed OntoBroker into your own Java applications. Just add the OntoBroker libraries to your application and develop programs using the OntoBroker Java API 92.

12.1 Socket Client API

OntoBroker provides a API for accessing a OntoBroker inference server. The new API replaces the classes:

- EmbeddedQueryClient
- EmbeddedCommandClient
- QueryClient
- CommandClient

The new API is more user-friendly and easier to understand. The new API is based on a new (binary) client/server socket protocol which allows faster streaming of result data.

Getting Started

The socket client API was designed so that you only need one OntoBroker library to add to your CLASSPATH. Just take the:

```
com.ontoprise.ontobroker.socketclient_6.0.0.jar
```

from the "plugins" directory of your installation and add it to your program.

Short Tutorial

The basic idea behind the API is that you need a connection object for all actions (queries and commands). Connection objects can be created via

```
OntoBrokerConnection conn = OntoBrokerConnectionManager.getConnection();
```

The connection manager automatically opens the connection and tries to connect to an inference server. If the user name and password are provided, the manager will authenticate with the server. Note that the various "getConnection()" methods allow arguments for specifying host, port, user name, password and socket timeout.

The most important API interfaces are

- OntoBrokerConnectionManager (creates connections to an inference server)
- OntoBrokerConnection (represents a connection to an inference server and allows you to execute queries and commands)
- ServerException (is thrown when an error occurs on the server)
- ClientQueryConsumer (allows the query results to be received as a stream)

When you have a connection you can execute:

- queries and
- commands

on the connection. Besides the simple forms of retrieving results (get results as String arrays) it is also possible to get the results via a callback handler:

```
OntoBrokerConnection conn = OntoBrokerConnectionManager.getConnection();
conn.executeQuery("?- ?X::?Y.", new PrintingConsumer());
```

The "PrintingConsumer" is a simple implementation of the callback interface:

```
private class PrintingConsumer implements ClientQueryConsumer {

      private int _resultArity;

      @Override
      public void receivedQueryFinished() {
          System.out.println("Done.");
      }

        @Override
        public void receivedQueryStarted(long queryKey, String[] variableNames) {
            System.out.println("Query started: " + queryKey);
            _resultArity = variableNames.length;
            for (int i=0; i<_resultArity; i++) {
                if (i != 0) {
                    System.out.print(", ");
                }
                System.out.print(variableNames[i]);
            }
            System.out.println();
        }

        @Override
        public void receivedTraceMessage(String msg) {
            System.out.println(msg);
        }

        @Override
        public void receivedTuple(ClientTerm[] tupleBuffer) {
            for (int i=0; i<_resultArity; i++) {
                if (i != 0) {
                    System.out.print(", ");
                }
                System.out.print(tupleBuffer[i]);
            }
            System.out.println();
        }

    }
```

The "ClientTerm" looks similar to the "Term" interface, but behaves differently:

- It is not internalized. This means that two equal terms may not be identical.
- The "toString()" method only supports ObjectLogic

Besides this, handling is similar to the KAON2 API. For example, you can call a "getType()" method which will return a "ClientDatatype" which allows you to check if the term is a IRI constant, a string constant, a list, ...

12.2 OntoBroker Java API

This document covers fundamental aspects of using the Java API of OntoBroker 6. Its goal is to acquaint developers with the concepts of the API via useful examples and is to be seen as a supplement to the OntoBroker API javadoc, which remains the main source of reference.

OntoBroker 6 introduced numerous innovations compared to the last major release, amongst others, ObjectLogic replaced F-Logic as the main ontology language and will therefore be the language used in our examples. As the OntoBroker Java API has also seen a number of changes and improvements, even developers with previous OntoBroker experience will benefit from reviewing the document.

The code snippets presented throughout the document are kept concise, however, complete source code for working demo applications is attached to the respective chapters.

Accessing OntoBroker

The preferred way of accessing the OntoBroker Java API is via the 'Remote API' (for our purposes equivalent to the 'Collaboration Server' mode and further referred to as 'remote API'); only the remote API is guaranteed to support the complete API feature set.

Using the remote API means running OntoBroker independently from the application being developed, potentially on a different computer.

OntoBroker Setup

In order to start using the remote API, OntoBroker has to be configured to allow connections via the 'Remote API' or 'Collaboration Server'. If you plan on running OntoBroker on a different computer for development purposes, enabling the Web Console will let you control the server easily from your development workstation. To try out the examples and demo applications presented, remember to configure OntoBroker to use ObjectLogic. Refer to the OntoBroker documentation for further information on configuration.

If you run OntoBroker on a different machine, adjust the hostname or IP address in the example code. In case of connection problems, make sure that OntoBroker is really running and check the firewall settings.

Development Environment Setup

Using the OntoBroker Java API requires a number of semafora and third party libraries, which come packaged within the OntoBroker installation. The subdirectory interfaces\development\CodeExamples of your OntoBroker installation contains an Eclipse project, that will 'compile' the required libraries into a subdirectory. Follow the instructions in ReadMe.txt (in the project directory). Upon completing the steps, you will also have configured a 'User library collection' that can be added to your own projects with a few clicks making sure you don't miss any remote API library dependency.

OntologyManager

Having touched on connecting to an OntoBroker instance, let us now see how this can be programmed. The central point for establishing a connection to OntoBroker and managing ontologies is the so-called Ontology Manager. OntologyManager is parameterized by a Properties object, that holds any relevant connection settings and specifies the ontology language of choice.

This code snippet below instantiates an OntologyManager for the ObjectLogic language. The variable url in the fourth line must hold the hostname or IP of the OntoBroker instance, in our examples, this will be "localhost". Also note the two commented out lines for setting the username and password, unless you change the default security OntoBroker settings, these will not be needed.

```
// create a new ontology manager for ObjectLogic
Properties properties = new Properties();
properties.put(IConfig.ONTOLOGY_LANGUAGE, OntologyLanguage.OBJECTLOGIC.toString());
properties.setProperty("CollaborationServer.Url", url);             // IP or hostname
of OntoBroker
// properties.setProperty("CollaborationServer.User", "username");  // if applicable
// properties.setProperty("CollaborationServer.Password", "psswd"); // if applicable
OntologyManager manager = KAON2Manager.newOntologyManager(properties)
```

Note:

Ignoring security settings might be acceptable for development, however, make sure to secure OntoBroker properly for production use. For more information, see the OntoBroker documentation.

Note:

The examples mentioned in the API documentation can be downloaded at http://otc.ontoprise.com -> Downloads (registration is free).

12.2.1 Ontology Management

Prior to working with a knowledge base, the necessary ontologies have to be loaded or created. After modifying a file-based ontology, it has to be saved in order to preserve the changes. This topic deals with ontology management.

Opening an Existing Ontology from a File Location

If we intend to access an existing ontology stored in a file, we first of all need to import it. OntologyManager contains multiple overloads of the importOntology() method for doing this.

Using a File parameter is convenient as this lets us easily specify both relative and absolute paths on the local host. Please note that local host means the computer where your Java application is running. This is not necessarily the same as the host on which OntoBroker is running.

```
// open an ontology from a file
File ontologyFile = new File("resources/ontology.obl");
Ontology ontology = OntologyManager.importOntology(ontoFile, null);
```

Another alternative is to use one of the physical URI overloads; this is especially suitable for importing ontologies located on a Web server, although the 'file://' pseudoprotocol can be used to specify paths on the local file system.

```
// open an ontology from a web url
String ontologyPhysicalUri = "http://example.org/ontology.obl";
Ontology ontology = OntologyManager.importOntology(ontologyPhysicalUri, null);
```

For importing multiple ontology files at once, overloads exist for both parameter types. There is also one additional parameter in all the overloads, the ProgressListener; this allows for progress notifications of the client code, see Progress Listener 102 for more information.

Creating a New Ontology

Having initialized an OntologyManager instance, it can be used to create a new ontology. Only the ontologyURI must be passed to the method, null is allowed in place of the additional options parameter:

```
// create a new ontology
Ontology ontology = manager.createOntology("http://www.example.org/MyOnto", null);
```

Note that the ontology URI is a purely logical concept and although it may resemble a web URL, it does not in any way determine the file name or physical location upon saving. Also recall how the (logical) URI is different from the concept of a namespace. While a namespace can span multiple ontology files, there must be a one-to-one correspondence between ontology files and URIs. A (logical) URI corresponds to what is referred to as a module in F-Logic.

Saving an Ontology to a File

Whether you created a new ontology, or modified an existing one, it has to be saved to a file to preserve the changes. Saving an ontology to a file with the method saveOntology is straightforward; besides the file format, only the target must be provided. However, it is advisable to explicitly specify the text encoding to prevent problems with accented or national characters in a multi-platform setting.

Overloaded versions of the method also exist allowing writing an ontology to a an OutputStream or a Writer.

```
// save ontology to a file
File targetFile = new File("MyOntoFile.obl");
ontology.saveOntology(OntoBrokerOntologyFileFormat.OBJECTLOGIC,
                          targetFile, SerializationConstants.ENCODING_UTF);
```

Deleting an Ontology

When an ontology is no longer needed, it can be deleted. Deleting removes the ontology from the manager making it unavailable until it is re-imported again. This does not, however, modify the file system, should the ontology be originally imported from a file or not.

```
// delete an ontology
Ontology ontology = manager.getOntology("http://www.example.org/MyOnto");
manager.deleteOntology(ontology);
```

12.2.2 Data Manipulation

12.2.2.1 Retrieval

Queries vs. Requests

There are two fundamental ways of retrieving information from OntoBroker. Answering queries involves

reasoning, so that, for example, attribute values derived by rules are considered. This kind of information retrieval is identical to answering queries posed in the OntoBroker text query window.

On the other hand, answering requests only involves explicitly stated (or asserted) facts. Answering requests resembles syntactic matching.

Choice of the Retrieval Method

The choice of retrieval method also determines the form results are returned in. When answering a query, OntoBroker looks for variable assignments applicable to the conditions you state in the query; therefore, what is returned are n-tuples of qualifying variable values derived from the known facts. The form of these tuples may or may not directly correspond to that of the facts in the knowledge base.

As an example of the latter, consider querying a department of motor vehicles knowledge base for the average number of speed limit violations depending on the vehicle category. Assuming this information is not stored as a fact, an attempt to modify a tuple from such a result set and submit it back to OntoBroker would not be meaningful as it does not directly correspond to an actual object in the knowledge base. This is how queries are to be seen.

However, answering requests fetches objects from the knowledge base the way they were explicitly entered. This means that a record in the request result set directly corresponds to a record in the knowledge base and as such, can be easily deleted or modified. Hence, this retrieval method is especially of interest when programmatically modifying a knowledge base.

Queries Without Reasoning

When reasoning is not desired but requests are not suitable, there is an additional retrieval method. Using the inferoff option disables reasoning for a query whilst maintaining the greater degree of flexibility of queries.As an example, consider the following query; it will only retrieve direct instances of the concept area, ignoring any instances of its possible subconcepts.

```
@{options[inferoff]} ?- ?A:Area.
```

12.2.2.11 Queries

Querying through the API occurs in three steps:

1. Obtaining a Reasoner objects and Creating a Query.

Queries can be created from ObjectLogic code, or retrieved from the knowledge base. In the example below, we define a query directly using ObjectLogic.

Note the use of the outorder option, it lets you leave out any unnecessary helper variables from the result set and provides confidence concerning index range and element location when later accessing values in a result array.

```
Reasoner reasoner = ontology.createReasoner();
String queryText = "@{options[outorder(?D, ?T)]} " +
                   "?- ?M:Measurement[atDate->?D[_month->1], hasTemperature->?T].";
Query query = reasoner.createQuery(ns, queryText);
```

Namespaces

To keep the code readable, we avoid using fully qualified names for concepts and properties. This is made possible by a Namespaces object. A Namespaces object can be assigned one default and multiple prefixed namespaces allowing for more concise ObjectLogic code.

```
Namespaces ns = new Namespaces();
ns.setDefaultNamespace("http://www.ontoprise.de/ontology1#";);
```

2. Opening the Query and Processing the Results.

Upon calling the open method of a query, it is submitted to OntoBroker. As explained above, a result set is formed by a number of n-tuples, where n is the number of variables contained in the query. The number of tuples depends on the query and the knowledge base, it can always be a single tuple for an aggregation query, or a large number. However, you should always handle the case of an empty result set. (This may be surprising at first but, for example, the count aggregate function results in an empty result set rather than a single tuple holding a zero when no records are found).

Often, the number of tuples in a result set is not previously known and, in this case, the afterLast() method comes in handy. This method indicates if we have gone through all the tuples in the result set. If not, tupleBuffer() returns the current n-tuple as an array of Terms (See below for more information about query return types). The method next() moves a query's internal iterator to the next tuple.

```
query.open()
Term[] tuple = query.tupleBuffer();
while (!query.afterLast()) {
       System.out.println("Temp on " + tuple[0] + " was " + tuple[1]);
       query.next();
}
query.close();
```

3. Closing the Query.

To prevent a resource leak, remember to close your query. As exceptions can occur when querying, this is typically done in a final block.

```
query.close();
```

Working with Data Types

Query results are returned as an array of the generic type Term, the concrete type of the variables depends on what you queried for. For example, querying for simple attribute values yields a Constant. Having successfully cast a Term to Constant, the method Constant.getTypeURI() should provide sufficient guidance as to what Java datatype, the actual variable value returned by Constant.getValue() can be cast to. However, the preferred way to get a concrete value is using the TypeRegistry class:

```
// Extract a double typed temperature
if (TypeRegistry.isDouble(tuple[1])) {
    Double value = TypeRegistry.getDouble(tuple[1]);
}
```

12.2.2.12 Requests

As already explained, requests circumvent any reasoning and fetch axioms as explicitly entered in the knowledge base. The following illustrates requesting axioms with a simple condition.

Obtaining a Request Object

Axioms can be retrieved using a Request object; a Request instance can be obtained by calling the Ontology method createAxiomRequest. An axiom type must be passed to this method, virtually all subclasses of Axiom are admissable. The sample code below creates a request for axioms of type FClassMember, in other words, for axioms carrying the information on what objects are instances of what concepts.

```
Request<FClassMember> request = geographyOnto.createAxiomRequest(FClassMember.class);
```

Setting Conditions

Depending on the type of object being requested, different conditions can be set. Axioms pertaining to ObjectLogic concepts, instances and properties are concentrated in the class hierarchy below FAtom, each of these classes defines string constants for the applicable conditions (see the javadoc). The conditions are self-explanatory and follow from what information a particular FAtom subclass carries.

For example, the FClassMember binds an instance (called 'object') to a concept (called 'class object'). Since we intend to request the instances of a particular concept, we set a condition on the concept ('class object'), that should be equal to the particular concept name.

```
request.setCondition(FClassMember.CLASS_OBJECT_REQUEST_CONDITION_KEY,
    KAON2Manager.factory().constantIRI("http://www.ontoprise.de/ontology1#City"));
```

Multiple values for a condition can also be specified using the method setConditionMultipleValues. Any one of the specified values will then qualify a record for inclusion. Consider the following variation of the previous condition; in addition to instances of City, instances of Village would also be included in the result.

```
request.setConditionMultipleValues(FClassMember.CLASS_OBJECT_REQUEST_CONDITION_KEY,
    KAON2Manager.factory().constantIRI("http://www.ontoprise.de/ontology1#City"));
    KAON2Manager.factory().constantIRI("http://www.ontoprise.de/ontology1#Village"));
```

Accessing Results

The Request class provides several methods for accessing the results, being interested in all matching axioms from the current ontology, we call the plain get() variant. This returns the complete result set at once as a typed Set of FClassMember. In line with our expectations, the type of the result elements corresponds to the type requested, we can therefore directly make use of the three accessor methods.

```
Set<FPropertyMember> result = request.get();
for (FPropertyMember ax : result) {

       String cityLocal = ((IRIHolder) (((Constant) ax.getObject()).getValue())).
getLocal();
       String stateLocal = ((IRIHolder) (((Constant) ax.getValue()).getValue())).
getLocal();
       String propLocal = ((IRIHolder) (((Constant) ax.getProperty()).getValue())).
getLocal();
       System.out.printf("%-20s [ %20s -> %-20s ]%n", cityLocal, propLocal,
stateLocal);
}
```

Knowing the Right Axiom Type

When unsure about how a particular ObjectLogic construct is represented in the Axiom form (which is necessary for correctly setting the request condition), there is an alternative to a lengthy trial-and-error brain exercise. The KAON2Factory class provides a method for creating axioms from an ObjectLogic expression.

Inspecting the result of a code snippet like the one below should quickly get you back on the right track. (A similar check can obviously be done in the debugger.)

```
List<Axiom> axioms = KAON2Manager.factory().axioms(
              "Offenburg:City[hasName->\"Offenburg\", "
              + "hasPopulation->59171, partOfFederalState->BW].",
              namespaces, geographyOnto.getModule());
for (Axiom ax : axioms) {
       String className = "";
       if (ax instanceof FClassMember)
              className = "FClassMember";
       else if (ax instanceof FPropertyMember)
              className = "FPropertyMember";
       else if ...
```

12.2.2.13 Concept Hierarchy

The concepts or properties in an ontology often form a hierarchy. The ontology class provides methods for retrieving these hierarchies that can then be easily traversed.
This listing shows output from a sample program for printing the concept hierarchy:

```
Area
   City
   Country
   Federal State
Weather Station
Measurement
Date
CityWithID
```

Retrieving a Hierarchy

There are two methods available for retrieving hierarchies: getConceptHierarchy and getPropertyHierarchy. For both, the object returned is of the type Hierarchy.

```
Hierarchy hierarchy = ontology.getConceptHierarchy(false);
```

Traversing a Hierarchy

The traversal can be started at the root nodes proceeding the child nodes, or by jumping to a concrete node corresponding to a particular concept or property; the following code snippet demonstrates the first option (see the sample program for a slightly more sophisticated implementation):

```
for (Term t : hier.getRootTerms()) {
   Node n = hier.getNodeFor(t);
   printNodeHierarchy(n);
}

void printNodeHierarchy(Node n) {
   System.out.println(n.getTerm());
   for (Node cn : n.getChildNodes())
      printNodeHierarchy(cn);
}
```

12.2.2.2 Modifications

Ontology contents can be modified in two ways; by adding and removing axioms. These activities are closely related to Requests (please remember that requests retrieve asserted facts, i.e. explicitly stated knowledge base elements). Removing axioms, for example, would typically be preceded by placing a request. There is an alternative approach to modifying ontology contents using so-called bulk operations as demonstrated below.

Simple Add or Remove

Individual or multiple axioms can be added or removed by calling the appropriate method of the Ontology class. In the code snippet below, an axiom is created using the KAON2Factory and subsequently added to an ontology:

```
Axiom axComp = factory.fclassMember(
       factory.constantIRI(namespaces, "Heidelberg"),
       factory.constantIRI(namespaces, "City"));
geographyOnto.addAxiom(axComp);
```

Note:
The use of the namespaces object that frees us from having to write out the fully qualified object names.

An equivalent axiom can be created directly from ObjectLogic code as the next code snippet illustrates:

```
Axiom axText = factory.axiom("Heidelberg:City.", namespaces, geographyOnto.getModule
());
```

Adding or Removing Multiple Axioms

Multiple axioms can be added or removed equally easily:

```
String axiomText = "Offenburg:City[hasName->\"Offenburg\" hasPopulation->59171,
partOfFederalState->BW].";
List<Axiom> axioms = factory.axioms(axiomText, namespaces, geographyOnto.getModule
());
geographyOnto.addAxioms(axioms);
...
geographyOnto.removeAxioms(axioms);
```

Bulk Operations

As mentioned in the introduction, it is also possible to add or remove multiple axioms at once without having to specify each one of them individually. The described functionality is available in the form of the two methods bulkAddFacts and bulkRemoveFacts of an Ontology object. They both work in a similar fashion; one parameter specifies a pattern against which existing axioms are to be matched. The matching works very much like evaluating a request, i.e. no reasoning is performed and available axioms are simply compared with the provided prescription; this prescription (or pattern) also introduces variables that are used in the

second parameter to describe the changes to be carried out in a generic way.

The meaning of the second parameter is different for the two bulk operations. The second parameter of the bulkAdd operation specifies facts that should be asserted true after the execution of the operation. That is, if the facts from the second parameter do not already exist for a matching variable assignment, they will be added to the knowledge base. On the other hand, the bulkRemove operation asserts the facts from the second parameter false, i.e. if for a variable assignment (from the first parameter) facts of the form specified by the second parameter exist, they will be removed from the knowledge base.

To demonstrate this functionality assume that there is an ontology containing German cities along with the federal state they belong to. Cities are expressed as instances of a concept called City and related to the respective states by a property called partOfFederalState. For the sake of the example, assume that we introduce a new concept called BavarianCity as a subconcept of City and intend to make all of the existing cities located in Bavaria instances of this new concept, rather than of the generic city.

The condition or pattern for matching cities in Bavaria is composed of two axioms. The first axiom, formulaPartOfBayern, states that a variable C is assigned to an instance called Bayern by the means of a property called partOfFederalState:

```
Term var = factory.variable("C");
Term partOf = factory.constantIRI(ns, "partOfFederalState");
Term bayern = factory.constantIRI(ns, "Bayern");
FactFormula formulaPartOfBayern = factory.fpropertyMember(var, partOf, bayern);
```

The second axiom, formulaIsCity, states that the same variable is an instance of a concept called City;

```
Term city = factory.constantIRI(ns, "City");
FactFormula formulaIsCity = factory.fclassMember(var, city);
```

These two axiom form the complete condition, or pattern for matching existing facts:

```
List<FactFormula> queryList = new ArrayList<FactFormula>();
queryList.add(formulaPartOfBayern);
queryList.add(formulaIsCity);
```

Now, we need to express the changes that need to be made. Let us first of all focus on assigning the matching instances to the new concept BavarianCity. Upon execution, the matching instances should be assigned to the new concept. This is done by using the same variable to describe the state after execution.

```
Term bavCity = factory.constantIRI(ns, "BavarianCity");
FactFormula formulaIsBavCity = factory.fclassMember(var, bavCity);
List<FactFormula> addList = new ArrayList<FactFormula>();
addList.add(formulaIsBavCity);
```

Having prepared both arguments, we can now perform the bulkAdd operation:

```
geographyOnto.bulkAddFacts(queryList, addList);
```

At this point the Bavarian cities are explicitly assigned to both concepts, City and BavarianCity. To remove this possible source of inconsistencies, we will use the bulkRemove operation to remove the Bavarian cities from the regular City concept. This is easily done as we already have a pattern to match against that we can reuse:

```
List<FactFormula> removeList = new ArrayList<FactFormula>();
removeList.add(formulaIsCity);
geographyOnto.bulkRemoveFacts(queryList, removeList);
```

The bulk operations offer a high-performance alternative, especially in the case of changes involving large numbers of axioms.

12.2.3 Datamodel Listener

The OntoBroker API provides a convenient mechanism following the observer pattern for notifying client code about changes in an ontology. A listener can only be registered for selected ontologies and particular event types if desired.

Demo program output:

```
Initial average temperature: 12,000
New avg temperature: 13,100
New avg temperature: 14,400
```

```
New avg temperature: 15,600
```

Implementing a DatamodelListener

In order to monitor ontology changes, the DatamodelListener interface must be implemented. The interface is intuitive containing a single method handleEvent, which receives one argument of the generic type DatamodelEvent. Concrete events can be differentiated based on the particular dynamic type of this argument (see the javadoc for the possible subtypes). For example, when axioms are added or deleted from an ontology, events of type AxiomsUpdatedEvent are raised.

The following code snippets show a simple DatamodelListener implementation which prints the string representation of an added axiom.

```
class AddListener implements DatamodelListener {
   Formatting form = new Formatting(OntologyLanguage.OBJECTLOGIC);

   public void handleEvent(DatamodelEvent event) {
      if (event instanceof AxiomsUpdatedEvent) {
         for (OntologyChangeEvent chEvent : ((AxiomsUpdatedEvent) event)
               .getChanges()) {
            if(chEvent.getChangeAction() == ChangeType.ADD) {
               Axiom ax = chEvent.getAxiom();
               System.out.println("Axiom added: " + ax.toString(form);
            }
         }
      }
   }
}
```

Registering a Listener

The listener can be registered at an OntologyManager by calling the addDatamodelListener method. Besides a DatamodelListener instance, one additional argument of type DatamodelEventFilter is required. The filter controls which events of which type, occurring in which ontologies, are passed to a listener. The following instantiates a filter for the AXIOMS_UPDATED event in one ontology:

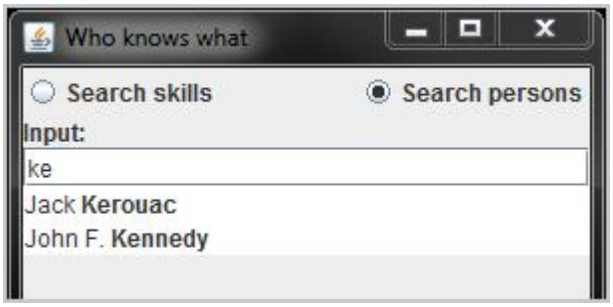
```
ArrayList<Integer> ontoIds = new ArrayList<Integer>(1);
ontoIds.add(ontology.getOntologyID());
DatamodelEventFilter filter = new DatamodelEventFilter(ontoIds, EnumSet
      .of(DatamodelEvent.EventType.AXIOMS_UPDATED), true);

manager.addDatamodelListener(new AddListener(), filter);
```

Controlling a Listener Lifecycle

Upon registering, events start being immediately passed to the listener. Calling the OntologyManager method removeListener will deregister the listener again. The method updateDatamodelListenerFilter allows you to adjust the filter applied.

12.2.4 Autocomplete

The auto-complete feature enables easy user interaction with the OntoBroker-based application by providing a list of partial textual matches to the user whilst entering an existing term name.



Autocomplete support covers concepts, instances, attributes, and relations. The completion results are basically term names from an ontology matching a user-supplied name prefix (the beginning of a local term name). Optional conditions and search options can be specified to further constrain the search.

The autocomplete functionality is accessible through the AutocompleteHelper class in the package com.ontoprise.indexer. As AutocompleteHelper's package location suggests, it makes use of the indexer infrastructure, that the programmer, however, need not be concerned with other than to ensure that the indexer is enabled in the ontology manager.

The following text briefly explains basic usage of the AutocompleteHelper class (complete source code for an example program is available at the end).

Indexer

Since we are working with ontologies, we first need to instantiate an OntologyManager object as usual, supplying the property values relevant for our environment.
Include the indexer enabling option:

```
Properties props = new Properties();
...
props.setProperty("FullTextIndex", "on");    // Enable the indexer
OntologyManager manager = KAON2Manager.newOntologyManager(props);
```

Note:
Enable the indexer in OntologyManager.

AutocompleteHelper Setup

After setting up an OntologyManager and loading the necessary ontology, we can instantiate the AutocompleteHelper. Completion results can first be retrieved when the index has been rebuilt.

```
AutocompleteHelper helper = new AutocompleteHelper(manager);

while (!AutocompleteHelper.isIndexUpdateToDate(myOntology)) {
   Thread.sleep(100);
}
```

Before retrieving any completion results, we need to set up an AutocompleteHelper.Options object to pass to the AutocompleteHelper. These options control, amongst other things, how terms are fetched and presented and allow additional search criteria to be specified. Multiple options objects can be kept handy and applied depending on the context in which autocomplete is needed.

A very useful option is the Concept Filter, which allows autocompletion to be limited to just instances of a particular concept. In the code snippet below, we limit the search to instances of the concept Person.

```
AutocompleteHelper.Options options = new AutocompleteHelper.Options();
Term concept = KAON2Manager.factory().constantIRI("http://www.ontoprise.de/
example#Person");
options.setConceptFilter(concept);    // Only search instances of Person
```

A further condition can be imposed on the completion results, where the variable ?OBJ refers to the potential completion suggestion, that would, with no additional filter, be returned. Our example filter eliminates completion suggestions for persons, who are currently on vacation (see the demo program for another example).

```
options.setAdditionalFilter(
     "AND ?OBJ[<http://www.ontoprise.de/example#isOnVacation>->false]");
```

Retrieving Completion Suggestions

At this point, a first set of completion results can be retrieved. Suppose our ontology contains three instances of concept Person: Jackie, James and Jim. The user first types a "j", that corresponds to the startString parameter in the code below. Consequently, all three instances are fetched and output. The user then continues by typing an "a". Some input handling method calls our code again, this time with the searchString "ja". This time around only the instances Jackie and James are outputted.

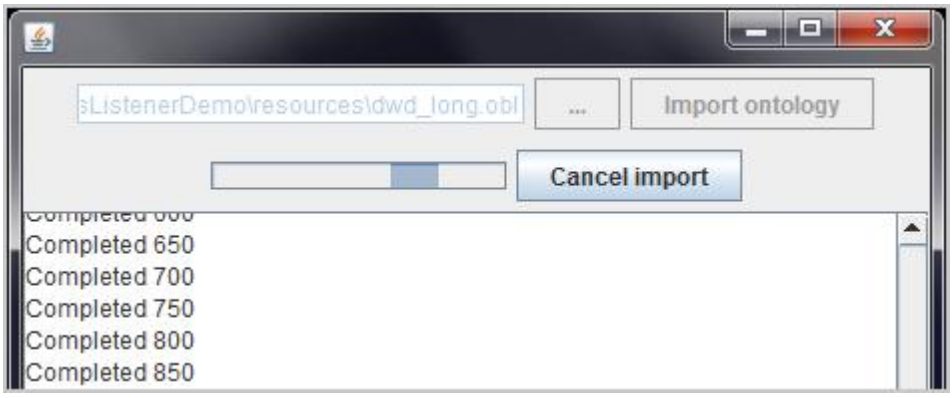
```
CompletionResults results
```

```
      = helper.getCompletion(ontology, options, Type.Instance, startString, 0, 20);

   for (CompletionElement elem : results.getStartingResults()) {
      model.addElement(elem);
      System.out.println(elem.getDisplay());
   }
```

In this section we have shown how easily the AutoCompleteHelper can be used. For a possible implementation, see the attached program.

12.2.5 Progress Listener

Selected methods with potentially lengthy execution take an optional argument of type ProgressListener. ProgressListener specifies an interface, through which these long running operations can provide feedback to the client code.



Operation execution is split into phases, whose beginning and end are reported; besides, operation progress is reported in units of work. This allows the developer to provide feedback to the user in case of interactive applications, or to log or otherwise record progress and results of a long-running operation.

ProgressListenerDemo

The Demo project is a simple interactive application that imports the selected ontology file into OntoBroker. Importing the 2MB ontology file bundled with the project makes the lag already apparent.

Worker Thread

In order to keep the GUI responsive so that we can display operation progress and let the user cancel the running operation, the long running operation must execute in another thread (different from the GUI thread). The demo application uses the convenient SwingWorker class to do this.

```
new SwingWorker<Void, Void>() {
   protected Void doInBackground() throws Exception {
      try {
         loader.importExistingOntology(f, listener);
      } catch (Exception ignore) {}
      return null;
   }
}.execute();
```

Implementing the ProgressListener Interface

For reasons of simplicity, the class representing the main application frame directly implements the ProgressListener interface. The methods simply display text messages about operation progress in a text area. Additionally, method calls indicating operation termination change the form state to indicate the import operation is no longer executing.

```
public void workUnitCompleted(int workUnitsCompleted, int totalWorkUnits) {
   outputText("Completed " + workUnitsCompleted);
}
public void operationCompleted() {
```

```
   outputText("op completed");
   setGuiState(false);
}
```

When implementing the interface methods, consider that they will be called from another thread. Therefore, any calls from within these methods must be thread-safe or guaranteed to execute on the right thread.

This code snippet shows how the demo application makes sure it is only modifying state of GUI components from within the GUI thread.

```
private void setGuiState(final boolean isImporting) {
   if (SwingUtilities.isEventDispatchThread()) {
            ...
      buttonCancel.setVisible(isImporting);
   } else {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            setGuiState(isImporting);
}});}}
```

12.2.6 Namespaces and Modules

In ObjectLogic, there are two ways of organizing ontology elements: modules and namespaces. Both can be used concurrently and independently of each other. The difference between the two may seem subtle in some situations, yet it is important to understand and differentiate them as they are powerful constructs that are encountered frequently. This chapter will briefly clarify the differences and then explain how the two concepts are reflected in the OntoBroker API.

Output from a demo program (read further for explanation):

```
Printing concepts from all modules from namespace: http://example.org/animals#
Concept                                  Module
http://example.org/animals#Cobra         http://example.org/Europe
http://example.org/animals#Fox           http://example.org/America
http://example.org/animals#Jaguar        http://example.org/America
http://example.org/animals#Mustang       http://example.org/America
http://example.org/animals#Viper         http://example.org/Europe

Printing concepts from all modules from namespace: http://example.org/cars#
Concept                                  Module
http://example.org/cars#Cobra            http://example.org/America
http://example.org/cars#Fox              http://example.org/Europe
http://example.org/cars#Jaguar           http://example.org/Europe
http://example.org/cars#Mustang          http://example.org/America
http://example.org/cars#Viper            http://example.org/America
```

The Difference Between namespaces and modules

When working with a single ontology, it corresponds to exactly one module. The module name is normally near the top of each ontology file. Because of this one-to-one correspondence, the module name is sometimes also called the ontology name (e.g. in OntoStudio). If you recall how ontology files are loaded from the chapter on Ontology Management, you will also notice a one-to-one correspondence between ontology files and instances of the Ontology class. The terms module, ontology, and ontology file may thus be used interchangeably, as long as the notion of a physical container for ontology elements described above is meant.

On the other hand, namespaces represent a logical concept which serves the purpose of naming and referring to ontology elements unambiguously. This is independent of modules: one module may contain elements from multiple namespaces and one namespace may span several modules.

For illustration purposes, consider the above listing; the first half are concepts from an '...animals' namespace, looking at the individual concepts, you will notice they come from two different modules: '... Europe' and '...America'. In the second half of the listing, you will see concepts with the same local names as above with a different namespace assigned, which disambiguates them. In a nutshell, modules (Europe, America) represent a 'physical' organization, whereas namespaces (cars, animals) capture a logical organization.

Namespaces and modules in the OntoBroker API

In this part, we will demonstrate how namespaces and modules are handled in the OntoBroker API. Although, the examples only cover adding concepts to existing ontologies, this should result in an intuitive understanding that can also be applied to other tasks.

The examples assume the existence of two ontologies, America and Europe, whose contents are listed above. Both ontologies contain concepts from a cars namespace as well as from an animals namespace; note that the concepts above are listed according to the namespace, not the ontology (module).

NOTE: There are certainly better ways to model cars and animal species than as unrelated concepts.

We intend to add the taxi and police favorite Chevrolet Impala to the 'America' ontology. We already have the necessary ontology object (americaOnto). Thus, we only need to create the necessary axiom, this can be achieved using one of the axiom factory methods. We need to explicitly specify not only the 'cars' namespace to make it clear that we do not mean the antelope but also the module, otherwise an attempt to add the axiom to an ontology would fail.

```
// add Impala to cars in America
// module explicit as text
// namespace explicit as text
Axiom ax = KAON2Manager.factory().axiom(
       "<http://example.org/cars#Impala>[]@<http://example.org/America>.",null);
americaOnto.addAxiom(ax);
```

As a next step, we extend our knowledge base by adding the classic among cars, the Volkswagen Beetle, to the 'Europe' ontology. This time, we make use of a convenience method of the Ontology class, that allows adding textually specified axioms directly to an ontology. By using the convenience method, it is no longer necessary to specify the module explicitly, as it is implied by the ontology (europeOnto).

```
// add Beetle to cars in Europe
// module implicit (one module per ontology!)
// namespace explicit as text
europeOnto.addAxioms("<http://example.org/cars#Beetle>[].");
```

For adding Mercury Cougar to the 'America' ontology, we follow the two-step approach again, first creating the axiom object using a factory method. Hence, the module needs to be specified explicitly. However, using a different overload of the factory method, we now pass the module and namespace as separate arguments. The module can be simply retrieved by calling the appropriate method of the Ontology instance. The namespace can then be fixed using a Namespaces object, which has the ability to hold multiple namespace declarations along with their respective prefixes and one default namespace.

```
// add Cougar to cars in America
// module explicit as a Term
// namespace explicit, default using a Namespaces object
Namespaces ns = new Namespaces();
ns.setDefaultNamespace(carsNS); // "http://example.org/cars#"
ax = KAON2Manager.factory().axiom("Cougar[].", ns, americaOnto.getModule());
americaOnto.addAxiom(ax);
```

Similar to the previous example, we add the rabbit to the 'animals' ontology. Only this time we make use of a non-default namespace. Therefore, the registered namespace prefix must also appear in the textual definition of the new axiom.

```
// add Rabbit to animals in America
// module explicit as a Term
// namespace explicit, non-default using a prefix
ns.registerPrefix("an", animalsNS);
ax = KAON2Manager.factory().axiom("an#Rabbit[].", ns, europeOnto.getModule());
europeOnto.addAxiom(ax);
```

The rabbit concludes our excursion to modules and namespaces in the OntoBroker API.

Querying modules and namespaces in ObjectLogic

Let us finish up the section with a tip for an ObjectLogic query that may find useful when experimenting with modules and namespaces; it lists concepts from all modules and all namespaces currently loaded.

```
?- ?C[]@?M AND ?C[_localName->?LN, _namespace->?NS].
```

12.3 OntoBroker and Web Services (SOAP/WSDL)

OntoBroker has a Web service interface for submitting queries and commands. It also allows you to add or remove multiple facts in a single call. The Web service interface is activated by starting the standalone OntoBroker server with the »-webservice« command line option. This description will give a short overview of the supported features. It is not intended to give an implementation overview.

Usage of Web service in the Standalone Server

In this case, the Web service is just another protocol for submitting queries and commands. For more details, see the "Standalone Deployment" chapter.

12.3.1 Webservice Interface

12.3.1.1 Web Service Operations

The OntoBroker web service provides six operations:

- `query` - Use this operation to run ObjectLogic or SPARQL queries
- `command` - Use this operation to send commands to the OntoBroker server
- `queryBatch` - Special operation to send multiple queries in a single message to OntoBroker
- `openQuery` - Starts evaluation of query and returns the query key
- `getNextQueryResults` - Retrieves the next results
- `cancelOpenQuery` - Call this operation to cancel an open query

12.3.1.2 Details on Query Input and Output

The "query" operation has two input parameters. With the "query" parameter you provide the query text of the ObjectLogic or SPARQL query. With the "fullxml" parameter you specify in which format the results are returned.

If fullxml is set to false, only the ObjectLogic representation of the terms is returned, i.e. only the "repr" field list of "WsTuple" is filled. If fullxml is set to true, the terms are returned with their complete structure. In this case, only the "term" field list of "WsTuple" is filled.

Example:
To see the difference, see the output of the following ObjectLogic query

```
?- _unify(?X,1.0) AND _unify(?Y,"Hi") AND _unify(?Z,["a","b","c"]).
```

On the XML level, the results look like this

```
With fullxml = false

<queryResponse>
<queryKey>2</queryKey>
<variableNames><var>X</var><var>Y</var><var>Z</var></variableNames>
         <result>
            <repr>1.0</repr>
            <repr>"Hi"</repr>
            <repr>[a,b,c]</repr>
</result>
</queryResponse>


With fullxml = true

<queryResponse>
<queryKey>3</queryKey>
<variableNames><var>X</var><var>Y</var><var>Z</var></variableNames>
<result>
            <term><d>1.0</d></term>
            <term><s>"Hi"</s></term>
            <term><list><item><s>a</s></item><item><s>b</s></item>
                                 <item><s>c</s></item></list></term>
</result>
</queryResponse>
```

More details can be found in the WSDL. The Deployment[106] section describes how to access the concrete WSDL.

12.3.2 Deployment

OntoBroker usually runs as a standalone server, but additionally provides the features described above on an embedded HTTP server. After installing the OntoBroker distribution, the only difference is the additional command line argument "-webservice". This tells OntoBroker to start its embedded HTTP server. The socket communication is still available in the same manner as in previous OntoBroker versions. By default, the HTTP server uses the port <8267>, but you can specify it explicitly in the "OntoConfig.prp" with the property "WsHttpPort"

To access the WSDL, use the following URL: `http://<ontobrokerhost>:<wshttpport>/services/ontobroker?wsdl`

If the OntoBroker server is running on your local machine and you are using the default port <8267>, the link is: `http://localhost:8267/services/ontobroker?wsdl`

To open the HTTP console, use the following URL: `http://<ontobrokerhost>:<wshttpport>/`

12.3.3 Load Balancing for Web Services

OntoBroker supports the AJP protocol to enable load balancing with the Apache HTTP Server 2.2.

1. Install Apache HTTP Server 2.2.
2. Edit <APACHE_HOME>/conf/httpd.conf

   a) Enable mod_proxy, mod_proxy_ajp, and mod_proxy_balancer.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

   b) Set up your cluster.

```
ProxyPass / balancer://mycluster/

<Proxy balancer://mycluster>
BalancerMember ajp://localhost:9001 loadfactor=1
BalancerMember ajp://localhost:9002 loadfactor=1
# The below is sample how to add a hot standby
#BalancerMember ajp://1.2.3.6:8009 status=+H
</Proxy>
```

For more details, see the ProxyPass Directive documentation at http://httpd.apache.org/docs/2.2/mod/mod_proxy.html.

3. Start OntoBroker instances:

   a) Set the following parameters in the first instance of OntoConfig.prp:

```
WorkerName          = ob1
AJP.Port            = 9001
WebConsole          = on
AJP                 = on
```

   Then start the OntoBroker.

   b) Set the following parameters in OntoConfig.prp of the second instance:

```
WorkerName          = ob2
AJP.Port            = 9002
WebConsole          = on
AJP                 = on
```

   Then start the OntoBroker.

Note:
For a production server do not forget to secure the Apache server!

After Apache is started as a load balancer and the OntoBroker instances are also running as well, the webservice can be accessed via a WSDL pointing to the Apache port e.g.:

```
http://localhost/services/ontobroker?wsdl  (Apache on port 80)
```

instead of

```
http://localhost:9001/services/ontobroker?wsdl
```

(OntoBroker on port 9001, which will not work)

The webconsole is accessible via:

```
http://localhost/console/Console.html
```

For more information on the web console click here 36.

Restrictions

Only stateless Web services are supported. As one negative example, the Semantic Guide Web service is not supported as it mainly consists of stateful operations. For the same reasons, the collaboration server and remote API are not available.

12.3.4 Paging Support for Web Service

The OntoBroker Web service now has support for paging through the results of a query. This is useful if you want to process large result sets in smaller pieces.

There are three operations in the OntoBroker Web service:

Operation	Annotation
openQuery	Starts evaluation of the query and returns the query key. `long queryKey = _client.openQuery(query, fullXml)`
getNextQueryResults	Retrieves the next results. `PagedQueryResults results = _client.getNextQueryResults(queryKey, maxResults, maxWaitTimeInMs)` The number of the results returned is restricted by two parameters: No more than maxResults are returned. If the call waits longer than maxWaitTimeIsMs milliseconds, the call returns the results it has collected during this time span. If the PagedQueryResults.isFinished() method returns true, the query is finished and there are no more results. The query is automatically closed on the server side.
cancelOpenQuery	Call this operation to cancel an open query. If you do not retrieve all of the query results, it is important to call this method as the query is still open on the server side and will block resources.

12.3.5 Extensions Directory to Deploy Web Services

To add a Web service to OntoBroker, you simply implement the normal JAX-WS webservice class which implements the marker interface OntoBrokerService and publishes it as an OntoBrokerService to OSGI.

```
package com.ontoprise.ontobroker.webservice;

/**
 * Marker interface for web services deployed together with OntoBroker
 */
@ExcludeAll
public interface OntoBrokerService {
}
```

The skeleton of your Web service class will look like this:

```
package com.acme.mywebservice

@javax.jws.WebService(endpointInterface ="com.acme.mywebservice.MyItf")
public class MyService implements MyItf, OntoBrokerService {
    private OntologyManager _ontologyManager;

    protected void bindOntoBrokerServiceManager(OntoBrokerServiceManager manager) {
        _ontologyManager = manager.getOntologyManager();
    }

    //  web service operations are added here
}
```

Normally a Web service deployed in OntoBroker needs access to the ontologies or the reasoner. For this purpose, your webservice class should reference the OntoBrokerServiceManager service and remember the OntologyManager on the bind operation. In addition you need to specify the relative address for publishing the Web service endpoint.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="com.acme.
mywebservice">
   <implementation class="com.acme.mywebservice.MySvc"/>
   <reference cardinality="1..1" interface="com.ontoprise.ontobroker.webservice.
OntoBrokerServiceManager" name="OntoBrokerServiceManager" policy="static" bind="
bindOntoBrokerServiceManager" />
      <provide interface="com.ontoprise.ontobroker.webservice.OntoBrokerService"/>
   </service>
   <property name="address" type="String" value="/myservice"/>
</scr:component>
```

The rest is similar to the above description for the built-in bundle.

Dynamic Web Services

Dynamic Web services are only generated automatically from OntoStudio or with the OntoBroker command generateDynamicSvc.

12.4 How to Access OntoBroker From Other Programming Languages

12.4.1 Web Service Interface

To access OntoBroker from the Web service interface, you first need the concrete URL for the OntoBroker WSDL. If OntoBroker runs as standalone server with the command line option -webservice, the URL is

```
http://<ontobrokerhost>:<wsport>/services/ontobroker?wsdl
```

The default value for <wsport> is 8267. For more details on the correct URL see "Deployment scenario 106".
The samples directory interfaces\webservice contains project archives for Visual Basic, C#, Java, and Python.

Note:
The COM interface is no longer supported. Please use the webservice interface instead (we provide sample VisualStudio projects for C# and VisualBasic).

12.4.2 Accessing Web Service with .NET

If you are using Visual Studio, you have to add a Web reference to your project before you can use the OntoBroker Web service:

1. Click on the menu item Project of the main menu and select **Add Service Reference...**.
2. Enter the URL of the WSDL and press the **GO** button.
   The dialog should report "1 service found".
3. Finally enter a name for the reference and press the **Add Reference** button.

For other languages or Web service frameworks see the documentation in order to understand how to build/ generate a Web service client starting with a WSDL.

The following example shows how to access OntoBroker from C#. The samples directory interfaces\webservice contains complete project archives for Visual Basic and C#. You can use these projects directly with Microsoft Visual Studio 2008 or Microsoft Visual Studio Express Editions.

```
using System;
using OntoBrokerWebserviceSample.OntoBroker;

namespace OntoBrokerWebserviceSample
{
    /// <summary>
    /// This is a sample how to use the OntoBroker 6.0 webservice with C#
    /// It is assumed that this webservice has already been added to the project
    /// using Project/Add Service Reference...
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
            OntobrokerTypeClient client = new OntobrokerTypeClient();
            string module = "<http://sample.ontoprise.de/ws>";
            string suffix = "@" + module + ".";

            // add some facts to the module http://sample.ontoprise.de/ws
            String msg;
            msg = client.command("insert Michael[age->15]" + suffix);
            Console.Out.WriteLine(msg);
            msg = client.command("insert John[age->18]" + suffix);
            Console.Out.WriteLine(msg);
```

```
            msg = client.command("insert Sandra[age->21]" + suffix);
            Console.Out.WriteLine(msg);

            // add a rule to the module http://sample.ontoprise.de/ws
            string rule = "@{adult} ?P:Adult :- ?P[age->?X], ?X >= 18.";
            // remove it first (only done here to allow multiple runs of this sample)
            msg = client.command(":- module " + module + ".\ndelete rule adult.");
            Console.Out.WriteLine(msg);
            // really add the rule
            msg = client.command(":- module " + module + ".\ninsert {" + rule + "}");
            Console.Out.WriteLine(msg);

            // query for all adult persons
            QueryResults results = client.query("?- ?P:Adult[age->?Age]@?M.", false);

            Console.Out.WriteLine("Results returned: " + results.result.Length);
            PrintResults(results);

            Console.Out.WriteLine();
            Console.Out.WriteLine("Same query with paging");
            Console.Out.WriteLine();

            // same query using paging
            int timeout = 900; // query can be removed after 900 seconds = 15 min, if
client dies
            long queryKey = client.openQuery("?- ?P:Adult[age->?Age]@?M.", false,
timeout);
            int maxResults = 1; // maximum number of results to return
            int maxWaitTime = 100; // maximum wait time for results on server side
            bool finished = false;
            while (!finished) {
                PagedQueryResults pagedResults = client.getNextQueryResults(queryKey,
 maxResults, maxWaitTime);
                PrintResults(pagedResults);
                finished = pagedResults.finished;
            }

            Console.Out.WriteLine("Press any key");
            Console.In.Read();
        }

        private static void PrintResults(QueryResults results)
        {
            foreach (WsTuple tuple in results.result)
            {
                for (int i = 0; i < results.variableNames.Length; i++)
                {
                    if (i > 0)
                        Console.Out.Write(", ");
                    Console.Out.Write(results.variableNames[i] + " = " + tuple.repr
[i]);
                }
                Console.Out.WriteLine();
            }
        }
    }
}
```

12.4.3 Accessing Web Service with Java

Below you can see how to access the OntoBroker Web service with JAX-WS. If you are using Java 1.6, nothing needs to be installed, as JAX-WS is part of Java Version 6 (Standard Edition). If you are using Java 1.5, you need to install JAX-WS separately. There are also other web service stacks available for Java which work similarly. Java 1.6 allows easy usage of Web services after generation of the client code. This is done by calling the wsimport tool of Java:

```
wsimport -d <classes> [-s <source>] http://<ontobrokerhost>:<wsport>/services/
ontobroker?wsdl
```

After this call, the generated java classes are written to the <classes> directory, their source code optionally to the directory <source>. If you include these generated classes to your code, you can start working with OntoBroker. The following sample code shows how to perform commands and queries.

The samples directory interfaces\webservice contains a complete project archive which can be used directly with Eclipse 3.5 and later.

```
/*
 * Copyright (c) 2008-2010 ontoprise GmbH.
 */
package sample;

import java.util.List;

import com.ontoprise.schema.ontobroker.ws._2009._01.PagedQueryResults;
import com.ontoprise.schema.ontobroker.ws._2009._01.QueryResults;
import com.ontoprise.schema.ontobroker.ws._2009._01.WsTuple;
import com.ontoprise.services.OntobrokerSvcService;
import com.ontoprise.services.OntobrokerType;

/*
 * This is a sample how to use the OntoBroker 6.0 webservice with Java 1.6 using JAX-
WS
 */
public class WebserviceClient {

    public static void main(String[] args) throws Exception {
        OntobrokerType client = new OntobrokerSvcService().getOntobrokerSvcPort();

        String module = "<http://sample.ontoprise.de/ws>";
        String suffix = "@" + module + ".";

        String msg;
        msg = client.command("insert Michael[age->15]" + suffix);
        System.out.println(msg);
        msg = client.command("insert John[age->18]" + suffix);
        System.out.println(msg);
        msg = client.command("insert Sandra[age->21]" + suffix);
        System.out.println(msg);

        // add a rule to the module http://sample.ontoprise.de/ws
        String rule = "@{adult} ?P:Adult :- ?P[age->?X], ?X >= 18.";
        // remove it first (only done here to allow multiple runs of this sample)
        msg = client.command(":- module " + module + ".\ndelete rule adult.");
        System.out.println(msg);
        // really add the rule
        msg = client.command(":- module " + module + ".\ninsert {" + rule + "}");
        System.out.println(msg);

        // query for all adult persons
        QueryResults results = client.query("?- ?P:Adult[age->?Age]@?M.", false);

        System.out.println("Results returned: " + results.getResult().size());
        printResults(results);

        System.out.println();
        System.out.println("Same query with paging");
        System.out.println();

        // same query using paging
        int timeout = 900; // query can be removed after 900 seconds = 15 min, if
client dies
        long queryKey = client.openQuery("?- ?P:Adult[age->?Age]@?M.", false,
timeout);
```

```
        int maxResults = 1; // maximum number of results to return
        int maxWaitTime = 100; // maximum wait time for results on server side
        boolean finished = false;
        while (!finished) {
            PagedQueryResults pagedResults = client.getNextQueryResults(queryKey,
maxResults, maxWaitTime);
            printResults(pagedResults);
            finished = pagedResults.isFinished();
        }

        System.out.println("Press any key");
        System.in.read();
    }


    private static void printResults(QueryResults results) {
        for (WsTuple tuple : results.getResult()) {
            final List<String> variableNames = results.getVariableNames().getVar();
            for (int i = 0; i < variableNames.size(); i++) {
                if (i > 0)
                    System.out.print(", ");
                System.out.print(variableNames.get(i) + " = " + tuple.getRepr().get
(i));
            }
            System.out.println();
        }
    }
}
```

12.5 Using the Extension API to Develop OntoBroker Extensions

Built-ins and rewriters may be developed to extend the capabilities of OntoBroker. They allow developing access functions to arbitrary systems, to develop any function needed or to introduce additional data into the inferencing process. The basic architecture of the inference engine is a system graph. In this graph, tuples flow from one vertex to the other vertices. A built-in is such a vertex. This means that it receives tuples, completes them and sends away the completed tuples.

CustomerIDE/CodeExamples

`com.ontoprise.ontobroker.examples.api` provides an eclipse-based environment for developing own OntoBroker extensions. It already contains code examples and unit tests for builtins, commands, connectors, rewriters and dynamic webservices.

12.5.1 Developing Your Own Built-ins

12.5.1.1 Making New Built-ins Known to OntoBroker

Make sure that your new built-in does not have the same name as an already existing one. You will find example built-ins in the directory

```
interfaces/development/Code Examples
```

This directory contains a ready-to-use Eclipse project which includes some examples.

12.5.1.2 Extensions Directory for Deploying Builtins

It is possible to package one or multiple builtins in an OSGI bundle. The bundle must register a BuiltinProvider service to announce its builtin contributions to OntoBroker. Most simply, a declarative service in OSGI can be used for this purpose.

The BuiltinProvider interface has one single method to return the builtin classes of this extension bundle.

```
package com.ontoprise.ontobroker.kernel.builtinsystem;

public interface BuiltinProvider {
    Collection<Class<? extends IBuiltin>> getBuiltins();
```

```
}
```

To implement this interface, create a class like the following example

```
package com.acme.mypackage

import java.util.*
import com.ontoprise.ontobroker.api.extension.builtin.IBuiltin;

public class MyBuiltinsProvider implements BuiltinProvider {
    @Override
    public Collection<Class<? extends IBuiltin>> getBuiltins() {
        ArrayList<Class<? extends IBuiltin>> list = new ArrayList<Class<? extends
IBuiltin>>();
        list.add(MyBuiltin.class);
        return list;
    }

}
```

Package this class together with your builtin classes in a JAR file. You need to add an OSGI conform META-INF/MANIFEST.MF file to the JAR.

META-INF/MANIFEST.MF (minimalistic version)

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyBuiltins
Bundle-SymbolicName: com.acme.mypackage
Bundle-Version: 1.0.0
Require-Bundle: com.ontoprise.ontobroker.base,com.ontoprise.ontobroker.datamodel
Import-Package: org.osgi.framework,org.osgi.service.component
Export-Package: com.acme.mypackage
Service-Component: OSGI-INF/component.xml
```

In addition, to register the OSGI service the simplest way is to define it as declarative service in an OSGI-INF/component.xml.

OSGI-INF/component.xml

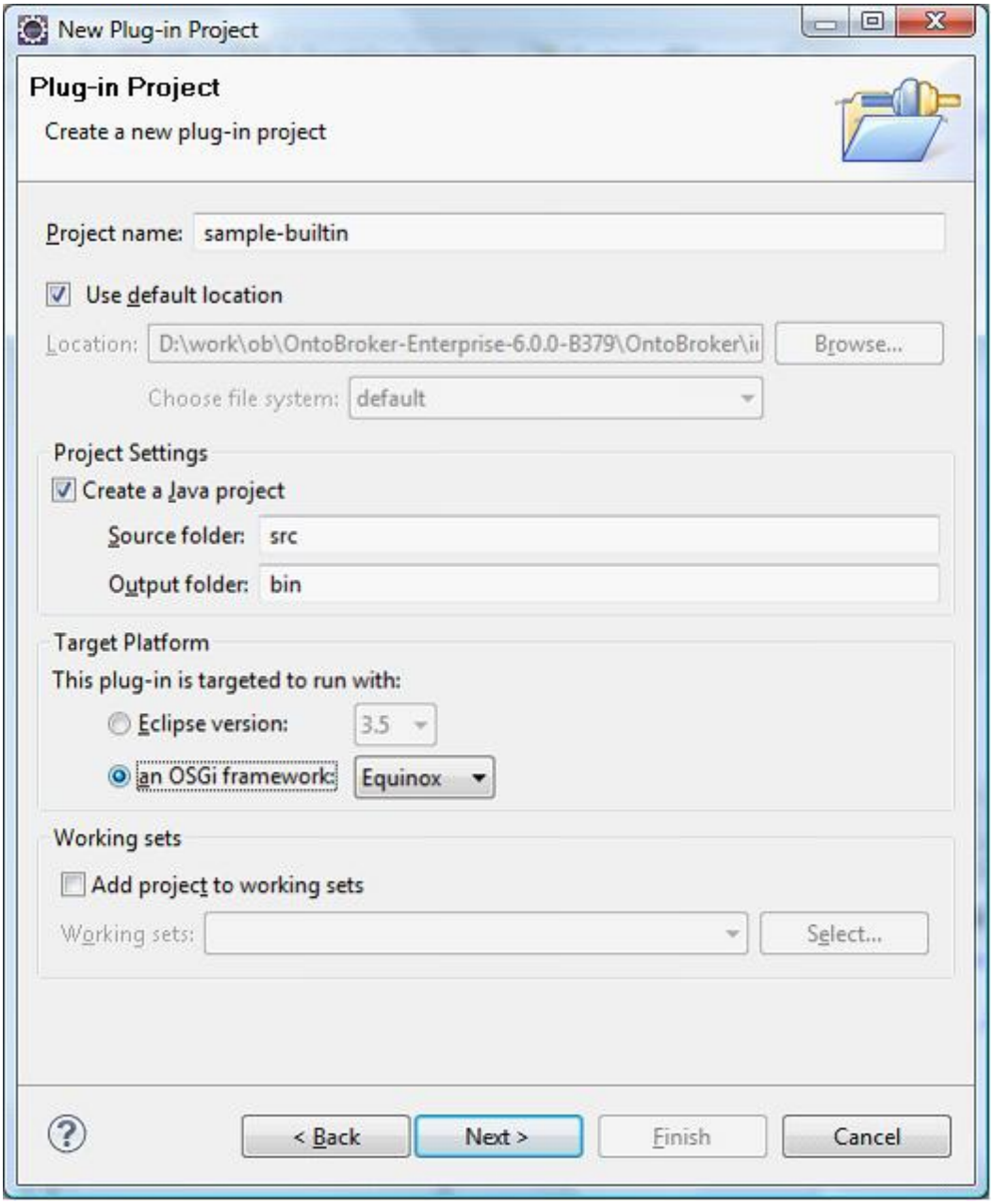
```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="com.acme.
mypackage">
   <implementation class="mypackage.MyBuiltinsProvider"/>
   <service>
      <provide interface="com.ontoprise.ontobroker.kernel.builtinsystem.
BuiltinProvider"/>
   </service>
</scr:component>
```

Also see the example in the Eclipse project located at $ONTOBROKER_HOME/interfaces/development/CodeExamples

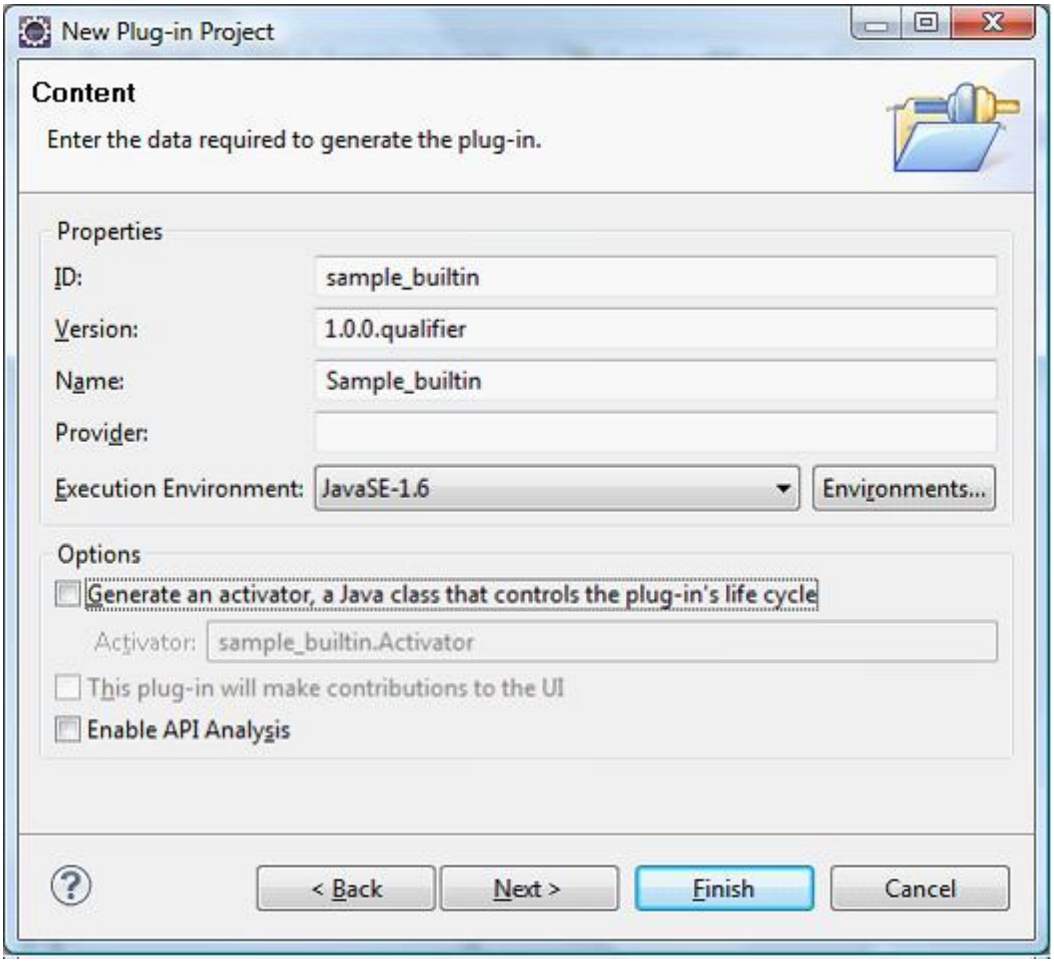
12.5.1.3 Writing a Built-in for OntoBroker 6

External builtins for OntoBroker must be packaged as an OSGi bundle. The bundle has to provide a BuiltinProvider service to announce its built-ins. After setting up the environment in Eclipse, you create your own built-in project in the following way:

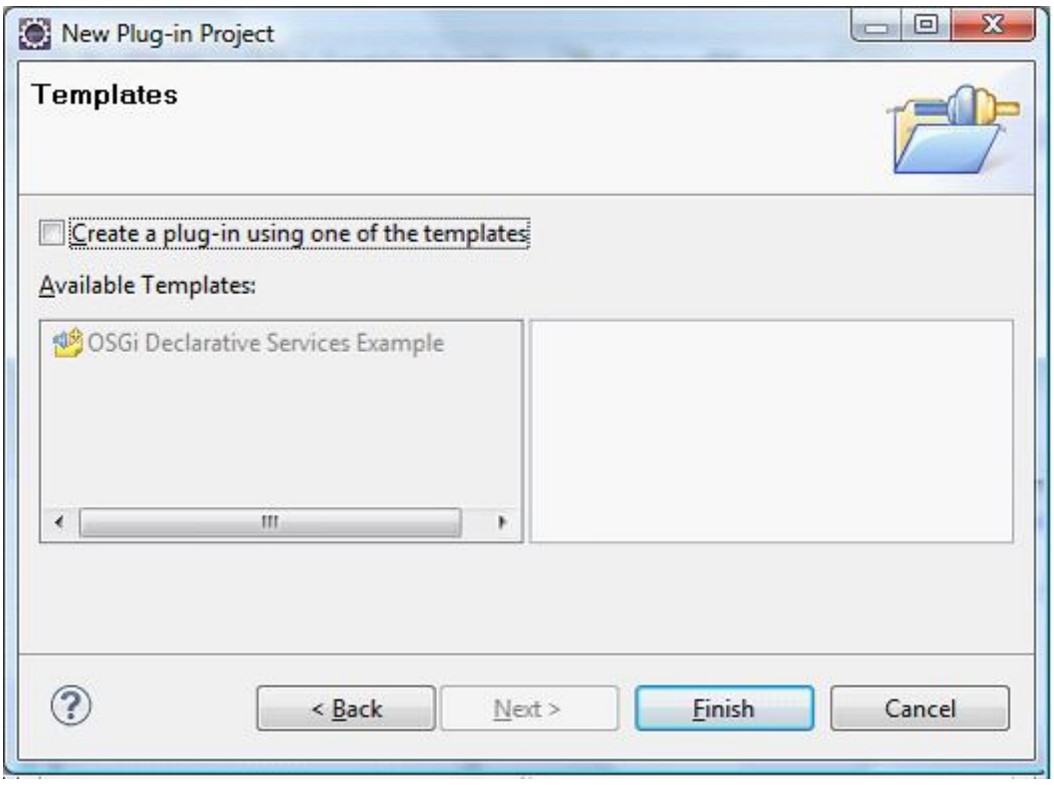
1. Eclipse Main menu **File** -> **New** -> **Other...** -> **Plugin-Project**.



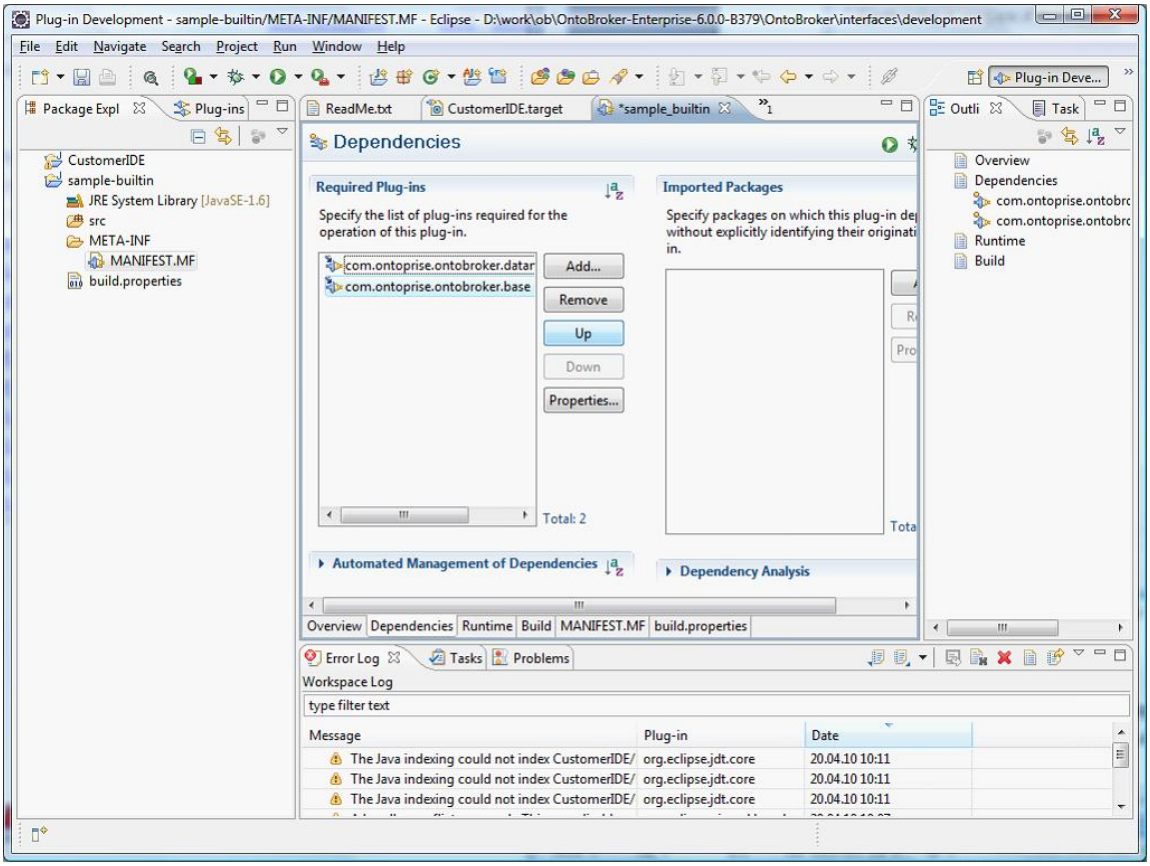
2. Ensure to check **an OSGi framework** in the Target Platform group.
3. Press **Next**.



4. Deactivate **Generate an activator**.
5. Press **Next**.



6. Deactivate the checkbox **Create a plug-in using one of the templates**.
7. Press **Finish**.
   Now add OntoBroker dependencies.
8. Select **META-INF/MANIFEST.MF** and double-click to open the editor.
9. Switch to tab **Dependencies**.



10. Press the **Add...** button of the required plug-ins and add
    - `com.ontoprise.ontobroker.base`
    - `com.ontoprise.ontobroker.datamodel`
11. Press the **Add...** button of Imported Packages and add `org.apache.log4j`.
12. Save the changes.
13. Add the following classes: `com.acme.sample.ConcatBuiltin.java`.

```
package com.acme.sample;


import java.util.BitSet;

import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.OntologyLanguage;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.ontobroker.api.extension.builtin.IBuiltinContext;
import com.ontoprise.ontobroker.api.extension.builtin.IFunctionalBuiltin;
import com.ontoprise.ontobroker.api.extension.builtin.IGrounds;
import com.ontoprise.ontobroker.apibase.extension.builtin.BuiltinSpec;
import com.ontoprise.util.OntopriseConstants;
import com.ontoprise.util.TermUtilities2;

/**
 * Example built-in which concats two strings.
 */
public class ConcatBuiltin implements IFunctionalBuiltin {

    /**
     * This method is called during the evaluation. We must first check
which of our four possible signatures applies and then either calculate
the result or
     * check if a result is valid
     *
     * @param tuple the tuple which must be checked
     * @throws InterruptedException if the process was interrupted
     * @throws KAON2Exception if an error occurs
     */
    @Override
    public boolean evaluate(Term[] input, IGrounds grounds) throws
KAON2Exception, InterruptedException {
        String str1 = TermUtilities2.getConstantValueAsString(input[0]);
        String str2 = TermUtilities2.getConstantValueAsString(input[1]);

        System.out.println("str1=" + str1 + ", str2=" + str2);
        if (str1 == null || str2 == null)
            return false; // input parameters cannot be converted to
strings

        String s = str1 + str2;
        Term result = KAON2Manager.factory().constantString(s);

        switch (grounds.getInt()) {
            case IGrounds.FIRSTSECONDTHIRD:
                  return result == input[2]; // unify calculated and given
result?
            case IGrounds.FIRSTSECOND:
                  input[2] = result;
                  return true;
            default:
                  return false; // should never be reached because of
behaviour of isEvaluable
        }
```

```
    }

    /**
     * This method will be called after evaluation for cleanup purposes.
     *
     * NOTE: It is possible that this operator will be used again when the
same query object is openened again.
     * init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void evaluationFinished() throws KAON2Exception,
InterruptedException {
        // This built-in has just one state, no need to set it back.
        // It is used to set a built-in to the post-init() state.
    }

    /**
     * Method to get informations about this built-in. This method has to
deliver a valid BuiltinSpec object
     * after the constructor finishes, before init() has be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this, "obl:
reserved:myconcat", 3);
        builder.setAllowedOntologyLanguages(OntologyLanguage.OBJECTLOGIC);
        builder.setDescription("Concats two strings");
        builder.setParameters("first string", "second string", "result");
        return builder.build();
    }

    /**
     * Initialisation of this built-in. This is done for the instance
before evaluation.
     *
     * @param context Contains additional Informations needed for some
built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void init(IBuiltinContext context, Term[] args) throws
KAON2Exception, InterruptedException {
        // Nothing to do, as this built-in does not need any
initialization.
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable
.
     * This method has to be usable context free, init() is called in an
other instance as
     * isEvaluable will be called. All needed information should be
provided by the current
```

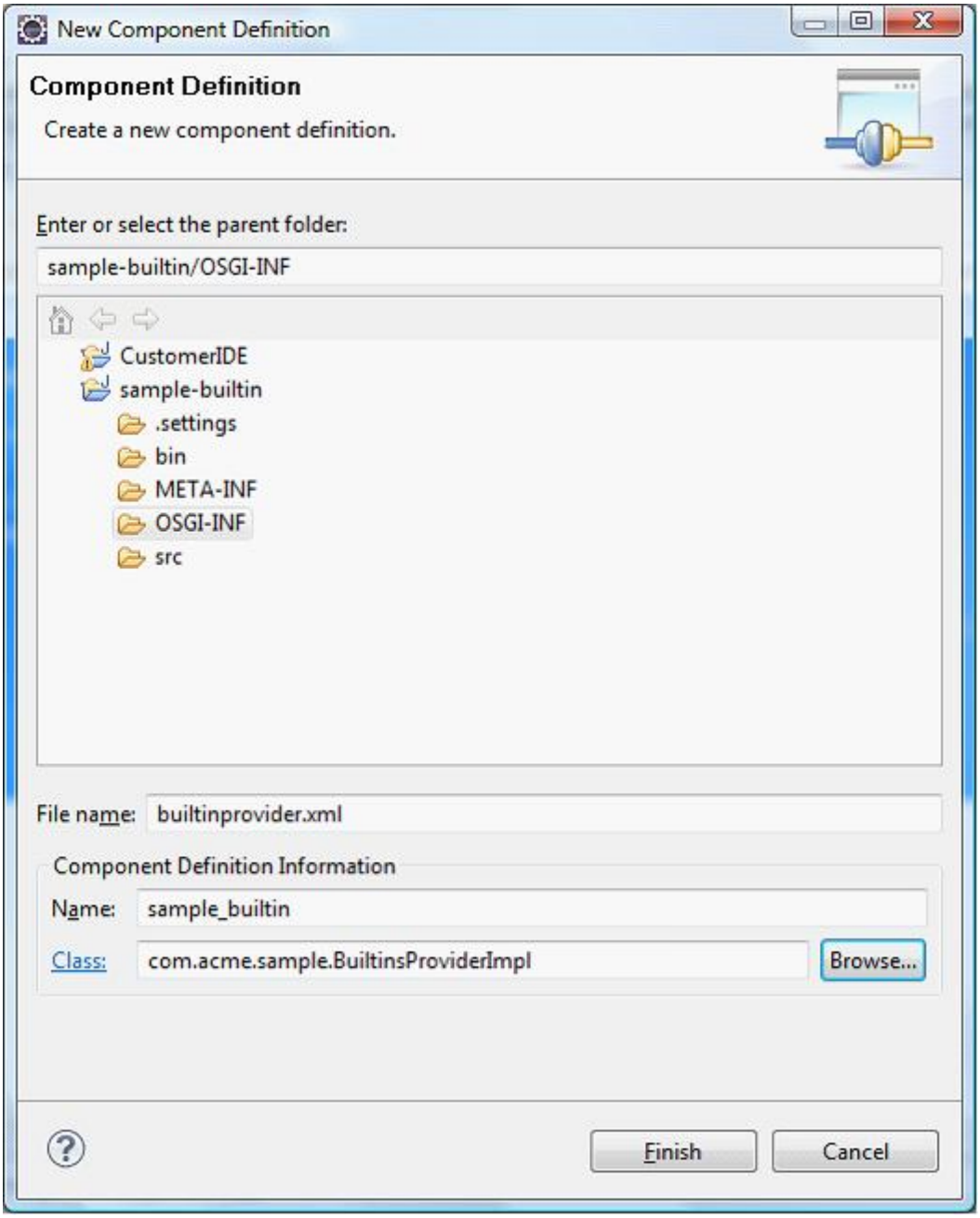
```
     * method arguments.
     *
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of
all variables that occurs. Needed especially if partial-ground functions
are used.
     * @param builtinContext BuiltinContext with additional context
informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    public boolean isEvaluable(IGrounds grounds, BitSet
variableInstantiations, IBuiltinContext builtinContext, Term[] args)
throws KAON2Exception {
      // first and second argument must be ground
        switch (grounds.getInt()) {
            case IGrounds.FIRSTSECONDTHIRD:
            case IGrounds.FIRSTSECOND:
                return true;
            default:
                return false;
        }
    }

}
```

com.acme.sample.BuiltinsProviderImpl.java

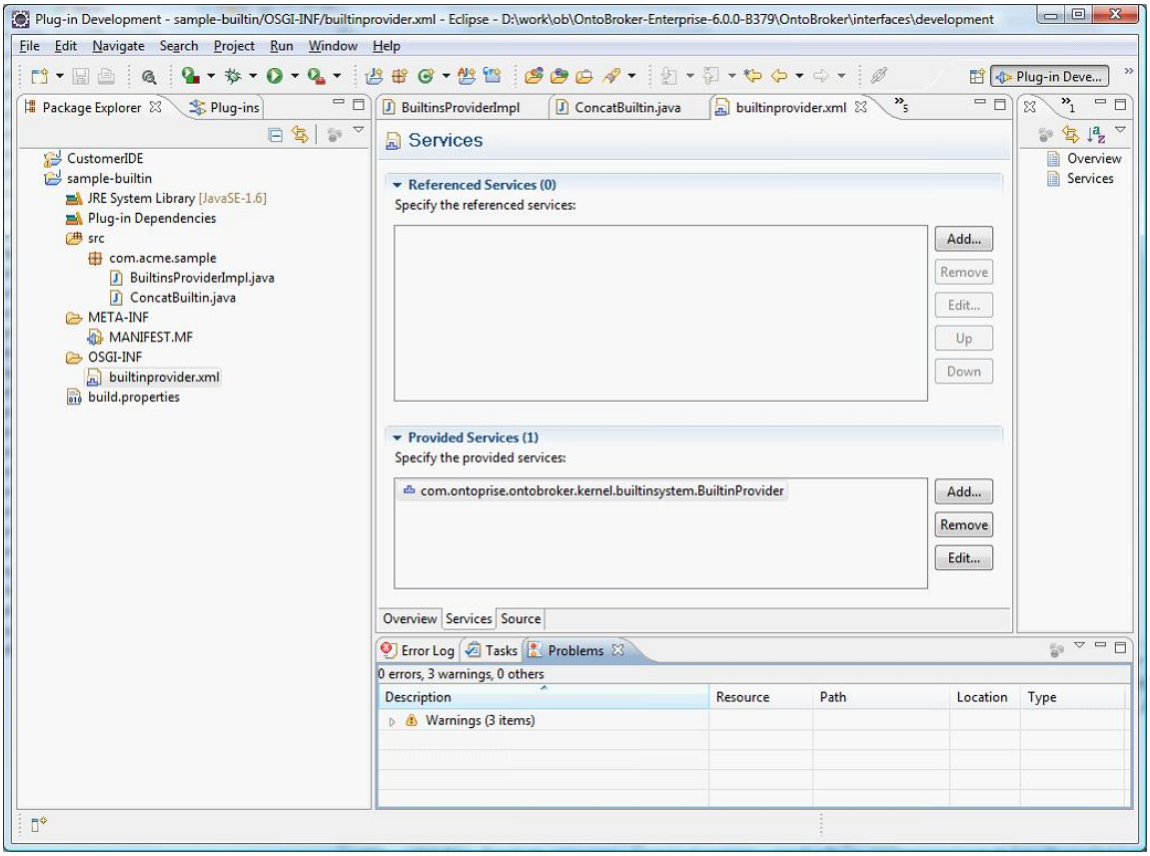
```
package com.acme.sample;

import java.util.ArrayList;
import java.util.Collection;

import com.ontoprise.ontobroker.api.extension.builtin.IBuiltin;
import com.ontoprise.ontobroker.kernel.builtinsystem.BuiltinProvider;

/**
 * Provider for sample builtins
 */
public class BuiltinsProviderImpl implements BuiltinProvider {
    @Override
    public Collection<Class<? extends IBuiltin>> getBuiltins() {
        ArrayList<Class<? extends IBuiltin>> list = new ArrayList<Class<? extends
IBuiltin>>();
        // this bundle has only one built-in
        list.add(ConcatBuiltin.class);
        return list;
    }
}
```

Declare the service

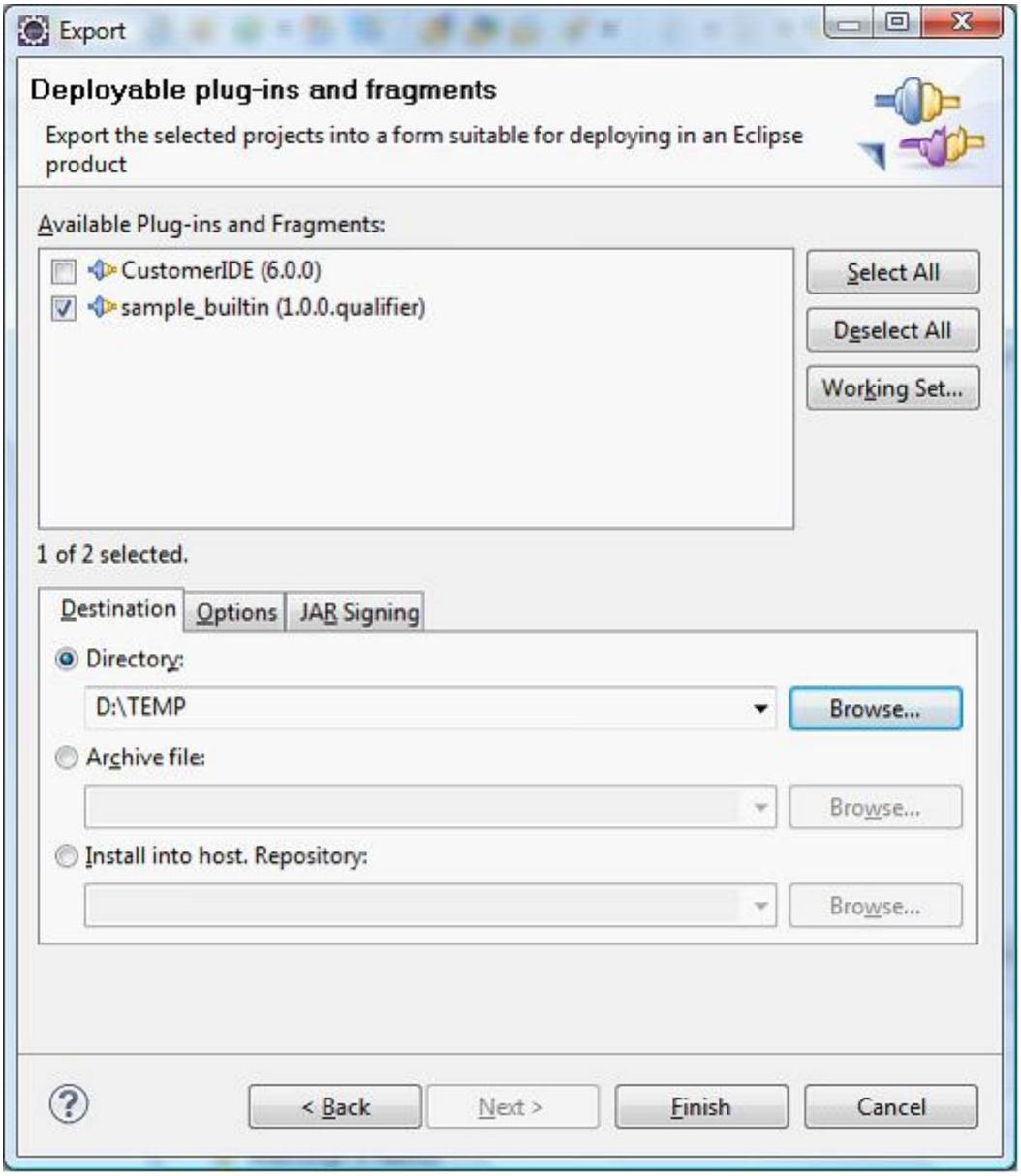
1. Add the folder "OSGI-INF" to the project.
2. Select the folder and choose from the context menu **New** -> **Component Definition**.
3. Fill the wizard.



4. Select the tab **Services** and add to Provided Services: `com.ontoprise.ontobroker.kernel.builtinsystem.BuiltinProvider`



5. Export the Jar file by selecting the sample-builtin project and choose **Export** -> **Deployable plug-ins and fragments**.



Now you have the sample_builtins_1.0.0.xxxx.jar in the directory D:\TEMP\plugins

Use the built-in

Copy this jar file to the OntoBroker extensions directory to use the built-in.

On adding the jar file to the extensions directory you should see following message in the log:

INFO [CONFIG] Builtin added: obl:reserved:myconcat/3

Run following query:

```
?- _myconcat("ab", "cd", ?X).
```

You should get as result:

```
?X = "abcd"
```

12.5.1.4 The IsNumber Example Built-in

All built-ins have to implement one of the following interfaces: IFilterBuiltin, IFunctionalBuiltin, IRelationalBuiltin and IConnector. These are different subclasses of built-ins. The filter built-ins just accept ground input tuples and return either true – if it matches the implemented filter, or false otherwise. The functional built-ins may return none or one single result, the relational built-ins may return none, one or several results. The connector built-ins are used to get data from external sources. A relational built-in

implies a functional built-in, and a functional built-in implies a filter built-in. So a relational (or functional) built-in could be used instead of the other types, but this will decrease the performance.

All built-ins have to implement methods to give information about the built-in, to say when it can be evaluated and to do the evaluation. The advantage of using of interfaces is that you now don't have to subclass a fixed class. The constructor must not take parameters, and the method getInfo() has to deliver the corresponding BuiltinSpec at any time, after an instance of the built-in was constructed.

It is a good style of programming to let all built-ins (perhaps except connectors) accept tuples with all arguments ground. Please consider this example:

```
?- 2.0[_add(2.0)->?X] AND _square(?X,4.0).
```

To use the same functionality without multiply accepting ground tuples you would have to do this:

```
?- 2.0[_add(2.0)->?X] AND _square("Y",4.0) AND _unify(?X,"Y").
```

In this case this is just one additional built-in, but it could decrease the performance, as in this case the results for Y are 2 and -2, two times the work for unify. Consider a built-in that would create more than 1000 results. Let's first have a look at a built-in that is already available. The unary built-in <IsNumber>; decides whether its argument is a number or not. It is based on <IFilterBuiltin>.

```
package com.ontoprise.builtin.filterbuiltin;

import java.util.BitSet;

import org.apache.log4j.Logger;
import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.OntologyLanguage;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.builtin.BuiltinContext;
import com.ontoprise.builtin.BuiltinSpec;
import com.ontoprise.builtin.builtin.BuiltinServices;
import com.ontoprise.builtin.interfaces.IFilterBuiltin;
import com.ontoprise.builtin.interfaces.IGrounds;
import com.ontoprise.util.OntopriseConstants;

/**
 * Example builtin which checks if a given tuple is a number.
 *
 */
public class IsNumberDemoBuiltin implements IFilterBuiltin {

    /**
     * Provide a no-argument constructor which does nothing - in this case.
     */
    public IsNumberDemoBuiltin() { // Empty as nothing is to do for this built-in
    }

    /**
     * This method is called during the evaluation. We must check for each input if
the condition ("isNumber") matches.
     *
     * @param input Term[] of always ground values.
     * @return boolean if this input is true or not.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public boolean isTrue(Term[] input) throws KAON2Exception, InterruptedException {

        try {
            // Here we always get a term array with arity 1 (this is what we
specified in getInfo()).
            if (BuiltinServices.isNumber(input[0])) {
```

```
                // Only return true if the condition matches.
                return true;
            }
        } catch (ClassCastException e) { // This exception cannot be thrown, as
BuiltinServices.isNumber does a check.
            System.out.println("invalid parameters", e); // However, any error should
be logged, no exception should be thrown just of invalid input.
        }
        return false; // Default return value for all non-matching input.
    }

    /**
     * This method will be called after evaluation for cleanup purposes.
     *
     * NOTE: It is possible that this operator will be used again when the same query
object is openened again.
     * init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void evaluationFinished() throws KAON2Exception, InterruptedException {
        // This built-in has just one state, no need to set it back.
        // It is used to set a built-in to the post-init() state.
    }

    /**
     * Method to get informations about this built-in. This method has to deliver a
valid BuiltinSpec object
     * after the constructor finishes, before init() has be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    @Override
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this, "isnumberdemo",
1);
        builder.setAllowedOntologyLanguages(OntologyLanguage.F_LOGIC);
        builder.setDescription("is X a number");
        builder.setParameters("X");
        return builder.build();
    }

    /**
     * Initialisation of this built-in. This is done for the instance before
evaluation.
     *
     * @param context Contains additional Informations needed for some built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void init(BuiltinContext context, Term[] args) throws KAON2Exception,
InterruptedException {
        // Nothing to do, as this built-in does not need any initialization.
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable.
     * This method has to be usable context free, init() is called in an other
instance as
```

```
     * isEvaluable will be called. All needed information should be provided by the
current
     * method arguments.
     *
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of all
variables that occurs. Needed especially if partial-ground functions are used.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    @Override
    public boolean isEvaluable(IGrounds grounds, BitSet variableInstantiations,
BuiltinContext builtinContext, Term[] args) throws KAON2Exception {
        return grounds.getBitSet().get(0); // The one argument this built-in needs
has to be ground
    }

}
```

Now take a look at MultiplyDemoBuiltin. It is a functional built-in, so it can deliver one result at most.

```
package com.ontoprise.builtin.functionalbuiltin;

import java.util.BitSet;

import org.apache.log4j.Logger;
import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.OntologyLanguage;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.builtin.BuiltinContext;
import com.ontoprise.builtin.BuiltinSpec;
import com.ontoprise.builtin.Grounds;
import com.ontoprise.builtin.builtin.BuiltinServices;
import com.ontoprise.builtin.interfaces.IFunctionalBuiltin;
import com.ontoprise.builtin.interfaces.IGrounds;
import com.ontoprise.util.OntopriseConstants;

/**
 * Example built-in which multiplies two numbers.
 *
 */
public class MultiplyDemoBuiltin implements IFunctionalBuiltin {
    protected static Logger _log = Logger.getLogger(OntopriseConstants.CORE_LOG);

    /**
     * Provide a no-argument constructor which needs nothing to do for this built-
in.
     */
    public MultiplyDemoBuiltin() { // Empty as nothing is to do for this built-in
    }

    /**
     * This method is called during the evaluation. We must first check which of our
four possible signatures applies and then either calculate the result or
     * check if a result is valid
     *
     * @param tuple the tuple which must be checked
     * @throws InterruptedException if the process was interrupted
     * @throws KAON2Exception if an error occurs
     */
```

```
    @Override
    public boolean evaluate(Term[] input, IGrounds grounds) throws KAON2Exception,
InterruptedException {
        try {
            switch (grounds.getInt()) {
                case IGrounds.FIRSTSECONDTHIRD:
                    // All three positions are ground. This means we must check if
"position 0" multiplied with "position 1"
                    // equals the "position 2" value. If this is true we send away
the received tuple.
                    double op1 = BuiltinServices.getNumber(input[0]);
                    double op2 = BuiltinServices.getNumber(input[1]);
                    double result = BuiltinServices.getNumber(input[2]);
                    if (op1 * op2 == result) {
                        return true;
                    }
                    break;
                case IGrounds.FIRSTSECOND: {
                    // First and second position is ground. We must multiply the
first with the second position
                    // and send away the result tuple.
                    op1 = BuiltinServices.getNumber(input[0]);
                    op2 = BuiltinServices.getNumber(input[1]);
                    // third argument must be filled with the result
                    result = op1 * op2;
                    // Create and send away the new tuple.
                    input[2] = KAON2Manager.factory().constantDouble(result);
                    return true;
                }
                case IGrounds.FIRSTTHIRD: {
                    // First and third position is ground. First position multiplied
with second position
                    // must be equal to the third position. We need to calculate the
second position.
                    op1 = BuiltinServices.getNumber(input[0]);
                    op2 = BuiltinServices.getNumber(input[1]);
                    // third argument must be filled with the result
                    result = op2 / op1;
                    // Create and send away the new tuple.
                    input[1] = KAON2Manager.factory().constantDouble(result);
                    return true;
                }
                case IGrounds.SECONDTHIRD: {
                    // Second and third position is ground. We must calculate the
first position.
                    op1 = BuiltinServices.getNumber(input[1]);
                    op2 = BuiltinServices.getNumber(input[2]);
                    // third argument must be filled with the result
                    result = op2 / op1;
                    // Create and send away the new tuple.
                    input[0] = KAON2Manager.factory().constantDouble(result);
                    return true;
                }
            }
        } catch (Exception e) {
            _log.error("Invalid input", e);
        }
        return false;
    }

    /**
     * This method will be called after evaluation for cleanup purposes.
     *
```

```
     * NOTE: It is possible that this operator will be used again when the same query
object is openened again.
     * init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void evaluationFinished() throws KAON2Exception, InterruptedException {
        // This built-in has just one state, no need to set it back.
        // It is used to set a built-in to the post-init() state.
    }

    /**
     * Method to get informations about this built-in. This method has to deliver a
valid BuiltinSpec object
     * after the constructor finishes, before init() has be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this, "multiplydemo",
3);
        builder.setAllowedOntologyLanguages(OntologyLanguage.F_LOGIC);
        builder.setDescription("multiplydemo");
        builder.setParameters("first parameter", "second parameter", "result");
        return builder.build();
    }

    /**
     * Initialisation of this built-in. This is done for the instance before
evaluation.
     *
     * @param context Contains additional Informations needed for some built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void init(BuiltinContext context, Term[] args) throws KAON2Exception,
InterruptedException {
        // Nothing to do, as this built-in does not need any initialization.
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable.
     * This method has to be usable context free, init() is called in an other
instance as
     * isEvaluable will be called. All needed information should be provided by the
current
     * method arguments.
     *
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of all
variables that occurs. Needed especially if partial-ground functions are used.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    public boolean isEvaluable(IGrounds grounds, BitSet variableInstantiations,
BuiltinContext builtinContext, Term[] args) throws KAON2Exception {
        // There are four different signatures that can be evaluated by this built-
in.
        // It is just necessary that at least three of the arguments are ground.
```

```
        // This method has not to check the input type, just the grounds.
        switch (grounds.getInt()) {
            case Grounds.FIRSTSECONDTHIRD: {
                return true;
            }
            case Grounds.FIRSTSECOND: {
                return true;
            }
            case Grounds.FIRSTTHIRD: {
                return true;
            }
            case Grounds.SECONDTHIRD: {
                return true;
            }
            default:
                return false;
        }
    }

}
```

Take a look at a relational built-in. Square either computes X^2 for input X, or it returns sqrt(X) and -sqrt(X):

```
package com.ontoprise.builtin.relationalbuiltin.mathematics;

import java.util.BitSet;

import org.apache.log4j.Logger;
import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.OntologyLanguage;
import org.semanticweb.kaon2.api.logic.Constant;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.builtin.BuiltinContext;
import com.ontoprise.builtin.BuiltinSpec;
import com.ontoprise.builtin.interfaces.IGrounds;
import com.ontoprise.builtin.interfaces.IReceiver;
import com.ontoprise.builtin.interfaces.IRelationalBuiltin;
import com.ontoprise.util.OntopriseConstants;

public class Square implements IRelationalBuiltin {
    protected static Logger _log = Logger.getLogger(OntopriseConstants.CORE_LOG);

    /**
     * Provide a no-argument constructor which needs nothing to do for this built-
in.
     */
    public Square() {
        // Empty as nothing is to do for this built-in
    }

    /**
     * Method to start evaluation. After this method has finished, any results have
     * to be sended by the IReceiver, no additional data is allowed.
     *
     * @param input Term[] with input tuples.
     * @param grounds IGrounds object with grounding informations.
     * @param destination Receiver to send every result to.
     *        Use destination.send(Term[] resultbuffer) to send results away.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
```

```
    public void evaluate(Term[] input, IGrounds grounds, IReceiver destination)
throws KAON2Exception, InterruptedException {
        try {
            switch (grounds.getInt()) {
                case 3: {
                    if (!(((Constant) input[0]).getValue() instanceof Double) || !
(((Constant) input[1]).getValue() instanceof Double)) {
                    }
                    double op1 = ((Double) ((Constant) input[0]).getValue()).
doubleValue();
                    double result = ((Double) ((Constant) input[1]).getValue()).
doubleValue();
                    if (op1 * op1 == result) {
                        destination.send(input); // sends the result away. The input
Term[] is used as buffer.
                    }
                    break;
                }
                case 1: {
                    if (!(((Constant) input[0]).getValue() instanceof Double)) {
                    }
                    double op1 = ((Double) ((Constant) input[0]).getValue()).
doubleValue();
                    double result = op1 * op1;
                    input[1] = KAON2Manager.factory().constant(new Double(result));
                    destination.send(input); // sends the result away. The input Term
[] is used as buffer.
                    break;
                }
                case 2: {
                    if (!(((Constant) input[1]).getValue() instanceof Double)) {
                    }
                    double op2 = ((Double) ((Constant) input[1]).getValue()).
doubleValue();
                    double result = Math.sqrt(op2);
                    input[0] = KAON2Manager.factory().constant(new Double(result));
                    destination.send(input); // sends the result away. The input Term
[] is used as buffer.
                    input[0] = KAON2Manager.factory().constant(new Double(result * -
1)); // just write the second result into the buffer, overwriting the first result.
                    destination.send(input); // sends the result away. The input Term
[] is used as buffer.
                    break;
                }
                default:
            }
        } catch (ClassCastException e) { // catch exception for wrong input values.
This is logical handled as false.
            _log.error("invalid input", e);
        }
    }

    /**
     * This method will be called after evaluation for cleanup purposes.
     *
     * NOTE: It is possible that this operator will be used again when the same
     * query object is openened again. init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void evaluationFinished() throws KAON2Exception, InterruptedException {
        // This built-in has just one state, no need to set it back.
```

```
        // It is used to set a built-in to the post-init() state.
    }

    /**
     * Method to get informations about this built-in. This method has to deliver a
     * valid BuiltinSpec object after the constructor finishes, before init() has
     * be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    @Override
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this, "square", 2);
        builder.setAllowedOntologyLanguages(OntologyLanguage.F_LOGIC);
        builder.setDescription("X^2 or sqrt(X) and -sqrt(X), depending of the
grounds");
        builder.setParameters("X", "X^2");
        return builder.build();
    }

    /**
     * Initialisation of this built-in. This is done for the instance before
evaluation.
     *
     * @param context Contains additional Informations needed for some built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void init(BuiltinContext context, Term[] args) throws KAON2Exception,
InterruptedException {
        // Nothing to do, as this built-in does not need any initialization.
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable. This
method has to be
     * usable context free, init() is called in an other instance as
     * isEvaluable will be called. All needed information should be provided by the
current method arguments.
     *
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of all
variables that occurs.
     *        Needed especially if partial-ground functions are used.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    @Override
    public boolean isEvaluable(IGrounds grounds, BitSet variableInstantiations,
BuiltinContext builtinContext, Term[] args) throws KAON2Exception {
        // There are four different signatures that can be evaluated by this built-
in.
        // It is just necessary that at least three of the arguments are ground.
        // This method has not to check the input type, just the grounds.
        switch (grounds.getInt()) {
            case 1: // first argument is ground, binary: 01
                return true;
            case 2: // second argument is ground, binary: 10
                return true;
```

```
            case 3: // both arguments are ground, binary: 11
                return true;
            default:
                return false; // cannot be evaluated
        }
    }
```

12.5.1.5 The Multiply Example Built-in

The ternary built-in multiply takes two numbers (the first two arguments) multiplies them and returns the result in the third argument. In addition to the other examples a new resulting atom must be generated which returns the result. This built-in has three arguments and has the string representation <multiply> in ObjectLogic programs:

```
package com.ontoprise.builtin.functionalbuiltin;

import java.util.BitSet;

import org.apache.log4j.Logger;
import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.OntologyLanguage;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.builtin.BuiltinContext;
import com.ontoprise.builtin.BuiltinSpec;
import com.ontoprise.builtin.Grounds;
import com.ontoprise.builtin.builtin.BuiltinServices;
import com.ontoprise.builtin.interfaces.IFunctionalBuiltin;
import com.ontoprise.builtin.interfaces.IGrounds;
import com.ontoprise.util.OntopriseConstants;

/**
 * Example built-in which multiplies two numbers.
 *
 */
public class MultiplyDemoBuiltin implements IFunctionalBuiltin {
    protected static Logger _log =
Logger.getLogger(OntopriseConstants.CORE_LOG);

    /**
     * Provide a no-argument constructor which needs nothing to do for this built-in.
     */
    public MultiplyDemoBuiltin() { // Empty as nothing is to do for this built-in
    }

    /**
     * This method is called during the evaluation. We must first check which of our
four possible signatures applies and then either calculate the result or
     * check if a result is valid
     *
     * @param tuple the tuple which must be checked
     * @throws InterruptedException if the process was interrupted
     * @throws KAON2Exception if an error occurs
     */
    @Override
    public boolean evaluate(Term[] input, IGrounds grounds) throws KAON2Exception,
InterruptedException {
        try {
            switch (grounds.getInt()) {
                case IGrounds.FIRSTSECONDTHIRD:
                    // All three positions are ground. This means we must check if
"position 0" multiplied with "position 1"
```

```
                    // equals the "position 2" value. If this is true we send away
the received tuple.
                    double op1 = BuiltinServices.getNumber(input[0]);
                    double op2 = BuiltinServices.getNumber(input[1]);
                    double result = BuiltinServices.getNumber(input[2]);
                    if (op1 * op2 == result) {
                        return true; // return true to signal there is a result in
the buffer
                    }
                    break;
                case IGrounds.FIRSTSECOND: {
                    // First and second position is ground. We must multiply the
first with the second position
                    // and send away the result tuple.
                    op1 = BuiltinServices.getNumber(input[0]);
                    op2 = BuiltinServices.getNumber(input[1]);
                    // third argument must be filled with the result
                    result = op1 * op2;
                    // Create and send away the new tuple.
                    input[2] =
KAON2Manager.factory().constantDouble(result); // write the result into the buffer
                    return true; // return true to signal there is a result in the
buffer
                }
                case IGrounds.FIRSTTHIRD: {
                    // First and third position is ground. First position multiplied
with second position
                    // must be equal to the third position. We need to calculate the
second position.
                    op1 = BuiltinServices.getNumber(input[0]);
                    op2 = BuiltinServices.getNumber(input[2]);
                    // third argument must be filled with the result
                    result = op2 / op1;
                    // Create and send away the new tuple.
                    input[1] =
KAON2Manager.factory().constantDouble(result); // write the result into the buffer
                    return true; // return true to signal there is a result in the
buffer
                }
                case IGrounds.SECONDTHIRD: {
                    // Second and third position is ground. We must calculate the
first position.
                    op1 = BuiltinServices.getNumber(input[1]);
                    op2 = BuiltinServices.getNumber(input[2]);
                    // third argument must be filled with the result
                    result = op2 / op1;
                    // Create and send away the new tuple.
                    input[0] =
KAON2Manager.factory().constantDouble(result); // write the result into the buffer
                    return true; // return true to signal there is a result in the
buffer
                }
            }
        } catch (Exception e) {
            _log.error("Invalid input", e);// catch exception for wrong input values.
This is logical handled as false.
        }
        return false; // return false to signal there no valid result in the buffer
    }

    /**
     * This method will be called after evaluation for cleanup purposes.
     *
```

```
     * NOTE: It is possible that this operator will be used again when the same query
object is openened again.
     * init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void evaluationFinished() throws KAON2Exception, InterruptedException {
        // This built-in has just one state, no need to set it back.
        // It is used to set a built-in to the post-init() state.
    }

    /**
     * Method to get informations about this built-in. This method has to deliver a
valid BuiltinSpec object
     * after the constructor finishes, before init() has be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this, "multiplydemo",
3);
        builder.setAllowedOntologyLanguages(OntologyLanguage.F_LOGIC);
        builder.setDescription("multiplydemo");
        builder.setParameters("first parameter", "second parameter", "result");
        return builder.build();
    }

    /**
     * Initialisation of this built-in. This is done for the instance before
evaluation.
     *
     * @param context Contains additional Informations needed for some built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    public void init(BuiltinContext context, Term[] args) throws KAON2Exception,
InterruptedException {
        // Nothing to do, as this built-in does not need any initialization.
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable.
     * This method has to be usable context free, init() is called in an other
instance as
     * isEvaluable will be called. All needed information should be provided by the
current
     * method arguments.
     *
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of all
variables that occurs. Needed especially if partial-ground functions are used.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    public boolean isEvaluable(IGrounds grounds, BitSet variableInstantiations,
BuiltinContext builtinContext, Term[] args) throws KAON2Exception {
        // There are four different signatures that can be evaluated by this built-
in.
        // It is just necessary that at least three of the arguments are ground.
```

```
        // This method has not to check the input type, just the grounds.
        switch (grounds.getInt()) {
            case Grounds.FIRSTSECONDTHIRD: {
                return true; // all arguments are ground
            }
            case Grounds.FIRSTSECOND: {
                return true; // the third argument is not ground
            }
            case Grounds.FIRSTTHIRD: {
                return true; // the second argument is not ground
            }
            case Grounds.SECONDTHIRD: {
                return true; // the first argument is not ground
            }
            default:
                return false; // cannot be evaluated
        }
    }

}
```

12.5.1.6 A Connector Built-in

A connector is a built-in which is used to access external servers (for example, databases, indexes, search engines).
Query of the ontology:

```
?- p(?X) AND democonnector(?X,?Y).
```

How A Connector Works:

At first OntoBroker looks what variables have to be bound, free and which variables are not important for being evaluable. Then, it asks the connector for the goodness of some of the patterns. It is up to the connector to interpret the goodness, it could be the difference of a full table scan and using the primary key of a database access.

Then, the instance of the connector for the evaluation is created, and init is called. Start is called, followed by some calls for input. The connector could now start to work. If it needs to have all input tuples, it has to start the evaluation when stop is called. The connector has to send all data before leaving the stop method. (So don't use a separate thread for it.) After stop was called, other start/input*/stop cycles could occur. evaluationFinished may possible be called and could be followed from a repetition of the evaluation, but now started with start/input*/stop cycles.

```
package com.ontoprise.builtin.iconnector;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.BitSet;

import org.apache.log4j.Logger;
import org.semanticweb.kaon2.api.KAON2Exception;
import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.builtin.BuiltinContext;
import com.ontoprise.builtin.BuiltinSpec;
import com.ontoprise.builtin.interfaces.IConnector;
import com.ontoprise.builtin.interfaces.IGrounds;
import com.ontoprise.builtin.interfaces.IReceiver;
import com.ontoprise.util.OntopriseConstants;
import com.ontoprise.util.TermUtilities;
```

```
import com.ontoprise.utils.StringUtil;

/**
 * Example connector which reads information from a URL. A connector receives a list
of tuples
 * and sends away a list of tuples. This demo connector reads the content of each URL
and determines
 * the number of characters for each URL.
 *
 */
public class CountCharactersOfURLsConnector implements IConnector {
    private IReceiver _destination;
    private ArrayList _input;
    protected static Logger _log = Logger.getLogger(OntopriseConstants.CORE_LOG);
//    {{STRING, VARIABLE}};

    /**
     * Provide a no-argument constructur which sets the properties of the connector.
     */
    public CountCharactersOfURLsConnector() {
    }


    /**
     * Method to get informations about this built-in. This method has to deliver a
valid BuiltinSpec object
     * after the constructor finishes, before init() has be called.
     *
     * @return BuiltinSpec object with informations about this built-in.
     */
    @Override
    public BuiltinSpec getInfo() {
        BuiltinSpec.Builder builder = new BuiltinSpec.Builder(this,
"countCharactersOfURLs", 2);
        builder.setDescription("gets number of characters of different URI-
documents");
        builder.setParameters("URI","number of characters");
        return  builder.build();
    }

    /**
     * Method to signal that several input tuples will arrive. Between start() and
stop() the grounds will not change.
     *
     * @param grounds IGrounds object with grounding informations of all following
input tuples.
     * @param destination Receiver for each result created.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void start(IGrounds grounds, IReceiver destination) throws KAON2Exception,
 InterruptedException {
        _destination = destination;
        _input = new ArrayList(200);
    }

    /**
     * This method will provide the input tuples. May just called between start() and
stop().
     *
     * @param input Term[] with input.
     * @throws KAON2Exception Is thrown on Error
```

```
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void input(Term[] input) throws KAON2Exception, InterruptedException {
        Term[] terms = new Term[input.length];
        System.arraycopy(input, 0, terms, 0, input.length);
        _input.add(terms);
    }

    /**
     * Method to signal that no further input will follow within this block.
     * (Although there could be another start() / stop() cycle.)
     * All what has to be done to deliver results has to be done here if it was not
done before.
     * After this method has finished, any results have to be sended by the
IReceiver, no additional
     * data is allowed.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void stop() throws KAON2Exception, InterruptedException {
        Term[] newArguments = new Term[2];
        for (Term[] arguments: _input) {
            String url = StringUtil.unQuote((String) TermUtilities.getValue(arguments
[0]));
            int numOfCharacters = openAndReadURL(url);
            newArguments[0] = arguments[0];
            newArguments[1] = KAON2Manager.factory().constant(new Double
(numOfCharacters));
            _destination.send(newArguments);
        }
    }


    /**
     * Opens a URL, reads the content and counts the characters.
     *
     * @param urlString                the URL
     * @return                         the number of characters of the document at
the URL location
     */
    private int openAndReadURL(String urlString) {
        int numOfCharacters = 0;
        try {
            URL url = new URL(urlString);
            InputStreamReader isr = new InputStreamReader(url.openStream());
            BufferedReader in = new BufferedReader(isr);
            String line;
            while ((line = in.readLine()) != null) {
                numOfCharacters += line.length();
            }
            in.close();
        } catch (MalformedURLException ex) {
            _log.error("DemoConnector received invalid URL: " + urlString);
        } catch (IOException ex) {
            _log.error("DemoConnector got an IOException while reading the URL '" +
urlString + "'.", ex);
        }
        return numOfCharacters;
    }

    /**
```

```
     * Returns how effective these grounds may be evaluated. Should deliver 0 if it
does not matter.
     * Like IBuiltin.isEvaluable() this method may be called before init() has been
called.
     *
     * @param grounds IGrounds object with grounding informations of all following
input tuples.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return int with goodness value.
     */
    @Override
    public int getGoodness(IGrounds grounds, BuiltinContext builtinContext, Term[]
args) throws KAON2Exception, InterruptedException {
        // this connector has just one possible binding, so just use a constant
number between 0 and 1000.
        // 1000 should mean this connector is much faster than a connector with
goodness 0.
        return 500;
    }


    /**
     * This method will be called after evaluation for cleanup purposes.
     *
     * NOTE: It is possible that this operator will be used again when the same query
object is openened again.
     * init() won't be called a second time.
     *
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void evaluationFinished() throws KAON2Exception, InterruptedException {
        // This built-in has just one state, no need to set it back.
        // It is used to set a built-in to the post-init() state.
        // btw.: This is not a dispose method, don't use it to clean up. Maybe it
won't be called.
    }

    /**
     * Initialisation of this built-in. This is done for the instance before
evaluation.
     *
     * @param context Contains additional Informations needed for some built-ins
     * @param args Term[] that contains the literal arguments.
     * @throws KAON2Exception Is thrown on Error
     * @throws InterruptedException Is thrown on Interruption
     */
    @Override
    public void init(BuiltinContext context, Term[] args) throws KAON2Exception,
InterruptedException {
    }

    /**
     * Method to determine if a certain grounds configuration is evaluable.
     * This method has to be usable context free, init() is called in an other
instance as
     * isEvaluable will be called. All needed information should be provided by the
current
     * method arguments.
     *
```

```
     * @param grounds Object with grounding informations.
     * @param variableInstantiations This BitSets has the instantiation of all
variables that occurs. Needed especially if partial-ground functions are used.
     * @param builtinContext BuiltinContext with additional context informations
     * @param args Literal arguments
     * @return if this grounds are evaluable or not.
     * @throws KAON2Exception Is thrown on Error
     */
    @Override
    public boolean isEvaluable(IGrounds grounds, BitSet variableInstantiations,
BuiltinContext builtinContext, Term[] args) throws KAON2Exception {
        return grounds.getBitSet().get(0) && !grounds.getBitSet().get(1); // the
first argument must be ground, the second argument must not be ground.
    }

}
```

12.5.1.7 Important Note for Datatype Conversions

Working with type conversions can be tricky. There are a couple of problems that might lead to unexpected query results.

Examples:

```
<- string2number("35",35.0).
```

will result in "true". The query

```
<- string2number("35.0",35.0).
```

will return "true", too.

But this query

```
<- string2number(X,35.0) AND equal(X,"35").
```

will return "false". The reason is that X will be bound with "35.0" by the string2number builtin and the equal builtin correctly evaluates that the string "35.0" is not equal to the string "35". But if the optimizer "EqualUnifyRewriter" is turned on it will eliminate the equal builtin and will rewrite this rule to

```
<- string2number("35",35.0).
```

and this rule will again return "true".

12.5.1.8 Important Note for Floating Point Operation

The <add> built-in delivers non-whole-numbered results.

Description:

```
"?- 1.1[_add(?X)->16.1]."  delivers X = 15.000000000000002, but
"?- 1.1[_add(?X)->15.1]."  delivers X = 14.0.
```

The problem with this is that it is not OntoBroker-specific but related to floating point operations in general. The error can simply be reproduced by this java code:

```
        public static void main(String[] args) {
                double op1 = 1.1;
                double op2 = 15.1;
                double result = op2 - op1;
                System.out.println(op2+" - "+op1+" = "+result);
                op2 = 16.1;
                result = op2 - op1;
                System.out.println(op2+" - "+op1+" = "+result);
```

The output will be:

```
15.1 - 1.1 = 14.0
16.1 - 1.1 = 15.000000000000002
```

The point is that you are forced to round/format your result if you use floating point numbers. You should use the round/rounddigit built-ins to round the result of the add-operation to the desired precision.

12.5.1.9 Important Note for Number Usage

We support integer numbers that are 8 bytes signed (equivalent to Java's long data type) and ranges from:

```
-9,223,372,036,854,775,808   to    +9,223,372,036,854,775,807
```

We support floating point numbers according to IEEE 754 (equivalent to Java's double data type). Floating point numbers cover a range from:

```
4.94065645841246544e-324    to     1.79769313486231570e+308
```

both positive and negative.

12.5.1.10 List of Possible Signature Types

None. Every built-in has to check this for itself. If invalid input is delivered, no exception should be thrown (but log it). The built-in is just one method of realizing a predicate, and it should be of no difference if you use built-ins or rules or facts in the edb. If the arguments do not match, the result is false.

12.5.1.11 List of Predefined Grounds

Found within interface IGround.

```
FIFTH  FOURTHFIFTH
FIRSTFIFTH  FIRSTFOURTHFIFTH
SECONDFIFTH  SECONDFOURTHFIFTH
FIRSTSECONDFIFTH  FIRSTSECONDFOURTHFIFTH
THIRDFIFTH  THIRDFOURTHFIFTH
FIRSTTHIRDFIFTH  FIRSTTHIRDFOURTHFIFTH
SECONDTHIRDFIFTH  SECONDTHIRDFOURTHFIFTH
FIRSTSECONDTHIRDFIFTH   FIRSTSECONDTHIRDFOURTHFIFTH
```

All of these grounds define a bit vector which describes a ground argument by 1 and a variable by 0. Instead of these names an appropriate bit vector can be used. The vector is read from right to left; its last bit describes the first argument.

12.5.2 Developing Your Own Rewriters

When OntoBroker evaluates a query, it checks which rules are relevant for the query and then rewrites the relevant rules (e.g. for performance reasons). It is also possible to develop your own rewriters which then are applied during the reasoning process. The following example rewriter exchanges "less" built-ins with "greater" built-ins (of course it doesn't make any sense, but it helps illustrating the power of rewriters):

```
package com.ontoprise.rewriter.example;

import org.semanticweb.kaon2.api.KAON2Manager;
import org.semanticweb.kaon2.api.logic.Literal;
import org.semanticweb.kaon2.api.logic.PredicateSymbol;
import org.semanticweb.kaon2.api.logic.Rule;
import org.semanticweb.kaon2.api.logic.Term;

import com.ontoprise.exception.DataModelException;
import com.ontoprise.obfuscation.ExcludePublicMembers;
import com.ontoprise.rewriter.AbstractRuleRewriter;



public class ExampleRuleRewriter extends AbstractRuleRewriter {

    /**
     * This is the method which is called during the rewriting process. This method
is called for each
     * of the rules which are relevant for the current query.
     * <p>
     * If you do not want to touch a rule just return the original rule.
     * </p>
     */
```

```
    public Rule rewrite(Rule rule) throws DataModelException {
        Rule result = rule;
        if (containsLessLiterals(rule)) {
            result = exchangeBodyByGreater(rule);
        }
        return result;
    }

    /**
     * Checks if the given rule contains a "less" literal.
     *
     * @param rule          the rule to be checked
     * @return              <tt>true</tt> if it contains a "less" literal,
<tt>false</tt> otherwise.
     */
    private boolean containsLessLiterals(Rule rule) {
        for (Literal bodyLiteral: rule.getBodyLiterals()) {
            if (isLessLiteral(bodyLiteral)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Is the given literal a "less" literal?
     *
     * @param literal       the literal to be checked
     * @return              <tt>true</tt> if the given literal is a "less" literal,
<tt>false</tt> otherwise.
     */
    private boolean isLessLiteral(Literal literal) {
        PredicateSymbol predicate = literal.getPredicateSymbol();
        if (predicate.getName().equals("less") && predicate.getArity() == 2) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Returns a new rule where all occurences of "less" are replaced with "greater".
     *
     * @param rule          the rule with the "less" literals
     * @return              the new rule (with the "greater" literals)
     */
    private Rule exchangeBodyByGreater(Rule rule) {
        Literal[] newBodyLiterals = new Literal[rule.getBodyLength()];
        int index = 0;
        for (Literal bodyLiteral: rule.getBodyLiterals()) {
            if (isLessLiteral(bodyLiteral)) {
                // Create a new "greater" literal.
                PredicateSymbol greaterPredicate = KAON2Manager.factory().
predicateSymbol("greater", 2);
                Term[] newArguments = bodyLiteral.getArguments().clone();
                Literal greaterLiteral = KAON2Manager.factory().literal(bodyLiteral.
isPositive(), greaterPredicate, newArguments);
                newBodyLiterals[index] = greaterLiteral;
            } else {
                // Just reuse the old literal.
                newBodyLiterals[index] = bodyLiteral;
            }
            index++;
        }
```

```
        Rule newRule = KAON2Manager.factory().rule(rule.getAxiomID(), rule.
getHeadLiterals(), rule.isHeadConjunctive(), newBodyLiterals);
        return newRule;
    }
}
```

After you have developed your own rewriter you can try it out by compiling it, putting it into a JAR library and placing that library to the "rewriters" folder of your OntoBroker installation. When OntoBroker starts you should see a message that the rewriter was loaded and initialized.

12.5.2.1 Extensions Directory for Deploying Rewriters

To add one or multiple rewriters to OntoBroker, you need to publish a IRewriterProvider service.

```
package com.ontoprise.api.rewriter;

public interface RewriterProvider {
    List<RewriterSpec> getRewriterSpecs();
}
```

The procedure is analog to the above description for the builtin JAR file. Only the declarative service definition file is shown here.

OSGI-INF/component.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="com.acme.
myrewriters">
   <implementation class="mypackage.MyRewritersProvider"/>
   <service>
      <provide interface="com.ontoprise.api.rewriter.RewriterProvider"/>
   </service>
</scr:component>
```

12.5.3 Extensions Directory for Deploying a Web Service

To add a Web service to OntoBroker, you simply implement a normal JAX-WS webservice class, which implements the marker interface OntoBrokerService, and publish it as an OntoBrokerService to OSGI.

```
package com.ontoprise.ontobroker.webservice;

/**
 * Marker interface for web services deployed together with OntoBroker
 */
@ExcludeAll
public interface OntoBrokerService {
}
```

The skeleton of your Web service class will look like this:

```
package com.acme.mywebservice

@javax.jws.WebService(endpointInterface ="com.acme.mywebservice.MyItf")
public class MyService implements MyItf, OntoBrokerService {
    private OntologyManager _ontologyManager;

    protected void bindOntoBrokerServiceManager(OntoBrokerServiceManager manager) {
        _ontologyManager = manager.getOntologyManager();
    }

    //  web service operations are added here
}
```

Normally a Web service deployed in OntoBroker needs access to the ontologies or the reasoner. For this purpose, your webservice class should reference the OntoBrokerServiceManager service and remember the OntologyManager on the bind operation. Additionally you need to specify the relative address for publishing the Web service endpoint.

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="com.acme.
mywebservice">
   <implementation class="com.acme.mywebservice.MySvc"/>
   <reference cardinality="1..1" interface="com.ontoprise.ontobroker.webservice.
OntoBrokerServiceManager" name="OntoBrokerServiceManager" policy="static" bind="
bindOntoBrokerServiceManager" />
      <provide interface="com.ontoprise.ontobroker.webservice.OntoBrokerService"/>
   </service>
   <property name="address" type="String" value="/myservice"/>
</scr:component>
```

The rest is similar to the above description for the built-in bundle.

12.5.4 Use OntoBroker Examples

You can easily create custom Built-ins, commands, connectors, rewriters or webservices with the CustomerIDE. Got to

```
interfaces\development\com.ontoprise.ontobroker.examples.api\ReadMe.txt
```

Just follow the steps discribed in the ReadMe.txt file.

12.6 UIMA

Definition:

UIMA - Unstructured Information Management Architecture - is a technical infrastructure for processing natural language documents. It has been initiated by IBM and has become an OASIS standard.

"Unstructured information may be defined as the direct product of human communication. Examples include natural language documents, email, speech, images and video. It is information that was not specifically encoded for machines to process but rather authored by humans for humans to understand. We say it is “unstructured” because it lacks explicit semantics (“structure”) required for applications to interpret the information as intended by the human author or required by the end-user application. "

Source: http://docs.oasis-open.org/uima/v1.0/uima-v1.0.html

The basic concept is a pipeline of three stages.

1. **Collection Reader** - a collection reader provides information as text. A simple collection reader could just read a text file from the file system and provide the contained text as it is.
2. **Analysis Engine** - an analysis engine processes the text provided by the collection reader. It annotates pieces of text to be a particular kind of information. As an example, an analysis engine may scan the text for phone numbers. The information is stored in indexes.
3. **Consumer** - a consumer accesses the structured information in the indexes and processes them in some way. A typical consumer may store the information somewhere, e.g. in a database.

UIMA provides an extensible type system which includes primitive types like integers or strings. In addition there is a built-in type for (textual) annotations which consists of a start and an end index within a text. Usually each analysis engine extends the type system by one or more new types. For example, a new type 'PhoneNumber' can be added as a sub type of the annotation type. The analysis engine then annotates the text by creating new instances of PhoneNumber, including the start and end index in the text that the covered text is a textual representation of a phone number.

Each type may have associated features. This is a field for each instance of a type that contains a particular part of information. For example, the start and the end index are features of the annotation type. An analysis engine can define new features on its types, so the PhoneNumber type could have a feature 'normalizedNumber' which contains a normalized version of the phone number found in the text, for example by removing spaces.

For more information about UIMA take a look at http://uima.apache.org/documentation.html

12.6.1 Semantic Content Analytics (SCA)

Natural language processing is a large field with many useful applications. OntoBroker provides the capability to go beyond normal content analysis by adding semantics on top.

As an example, think of a hospital with an accounting center having an eye on invoices to avoid a loss of money. The amount an insurance company pays for an operation may depend on the cause of disease. Let's say that the data set for a particular operation includes natural language information about the symptoms and also a diagnosis.

Now, for a concrete operation, $1000 can be billed, if A is the cause of disease, while for B as cause of disease, $2000 are allowed. Sadly the diagnosis might be unclear or wrong. If the symptoms indicate A, but the diagnosis is B, the hospital would bill more than it is allowed to. On the other hand, if the symptoms indicate B but the diagnosis is A, the hospital would loose money by not going for B.

This is where semantic content analytics comes in. The symptoms are taken from the natural language text. Together with an ontology modeling symptom, descriptions and diagnosis, reasoning can then be used to derive the most likely cause of disease or to check for inconsistencies.

12.6.1.1 Using OntoBroker as CAS Consumer

OntoBroker provides an UIMA CAS consumer which iterates over the data in the UIMA indexes and pushes it to an OntoBroker Collab Server. Features are mapped to ObjectLogic concepts, feature types to properties of the related concepts. For an entry in an UIMA index of a particular type, a new instance of the concept corresponding to the type is created. The consumer allows the filtering of particular types or features. For more information, see the related parameter descriptions in the descriptor files of the consumer.

12.6.1.2 CAS Consumer Deployment

The OntoBroker installation contains a script under

`<OntoBroker Installation Folder>/interfaces/uima`

which creates a UIMA pear file including the CAS consumer. The pear file includes both, a CAS Consumer and an Analysis Engine descriptor file. The pear file can be installed using a pear installer tool or it can be used directly from within a third party application supporting UIMA. Please see your third party application documentation for more information.

The OntoBroker CAS consumer is currently based on Apache UIMA version 2.2.2.

12.6.1.3 Importing UIMA Facts into a Domain Ontology

Usually the types and features used by the annotators will not directly map to the vocabulary of a domain ontology. If a renaming of concepts is sufficient, you may use the OntoStudio mapping tool to map the UIMA-related concepts to the concepts in the domain ontology. For more advanced mappings, rules may be used.

An import structure for the involved ontologies may look like this:

```
Domain ontology
|- Mapping ontology - including mapping rules
    |- UIMA facts ontology - containing the facts from the consumer
```

13 Role-Based Security

OntoBroker provides role-based security features which allow you to:

- Use existing security infrastructures (e.g. LDAP, ActiveDirectory)
- Give roles read/write access to individual ontologies (modules)
- Restrict the access to OntoBroker server commands

OntoBroker is using Apache Ki for the security. See http://incubator.apache.org/ki/ for further details about this project and the features it provides.

13.1 Role-Based Security

Role-based security allows you to specify users, their roles and their permissions in a security ontology.

Example:

```
:- module = $security.
:- importmodule $'security-core'.

Joe:User[hasRole->ResearchAndDevelopmentMember].
Jane:User[hasRole->HumanResourcesMember].

ResearchAndDevelopmentMember:Role[
    hasReadPermission->Developers].

ControllingMember:Role[
    hasReadPermission->Salaries;
    hasWritePermission->Salaries].
```

This means

- Joe has the role "ResearchAndDevelopmentMember" and has read permission to the module "Developers" (which probably contains information about other developers, their projects and favorite programming languages).
- Jane has the role "HumanResourcesMember". So she has access to the "Salaries" module which contains interesting information about the salaries of all people working in the company.

Read more about permissions here 152.

Now consider that Joe is interested in the salaries of the other developers in his team. He will execute

```
?X ?- ?X:Developer@Developers AND ?X[hasSalary->?Y]@Salaries.
```

The first part of this query is OK: He has the rights to access the "Developers" module. But the second part of the query is not OK: He does not have the required read access rights to module "Salaries", so he gets no answers.

13.2 Realms

A realm is basically a resource which is able to authenticate users. Apache Ki (and therefore OntoBroker) support the following realms:

- `JDBC realm` (allows to use a SQL database for storing users and passwords)
- `LDAP realm`
- `ActiveDirectory realm`

OntoBroker also provides its own realm: `SecurityModuleRealm`. This realm is using an ontology (written in ObjectLogic) for specifying users, roles and permissions to modules. It also allows you to create powerful rules (e.g. all users with read permissions on module A also have read permissions on module B).

13.3 Module Permissions

We have three types of module permissions:

- Read permission (hasReadPermission)
- Write permission (hasWritePermission)
- Temporary write permission (hasTempWritePermission)

When a user has read permission he is allowed to read all of the facts from the corresponding module and he can use rules with body formulas which access the module.

Write permission grants a user write access to a module. This also means that the user is allowed to use rules with head formulas which access the corresponding module.

Temporary write permission for a module means that the user is not allowed to change facts of the module, but it is possible to execute queries with temporary facts which add new facts and rules for the duration of the query.

Important note for ontology imports:
User U has r-rights for ontology A which imports ontology B -> no "inherited" rights for ontology B.

13.4 Property Permissions

As for modules, there are three different permissions for properties

- Read permission (hasPropertyReadPermission)
- Write permission (hasPropertyWritePermission)
- Temporary write permission (hasPropertyTempWritePermission)

Permissions for properties are only activated if these three switches in the `OntoConfig.prp` are all activated:

```
Security.LoginRequired = on
Security.AccessControl = on
Security.PropertyAccessControl = on
```

In this case, the users need read access for every single property they are using.

Example:

Assuming you have the following module

```
:- default prefix = "http://company.com/".
:- module module1.

boss[name->"TheBoss"].

@{rule1} ?X[name2->?Y] :- ?X[name->?Y].
```

You can define roles with a set of property permissions in the $security.obl:

```
role1:Role[hasPropertyReadPermission-> <http://company.com/name>,
 hasPropertyTempWritePermission-> <http://company.com/name>,
 hasPropertyWritePermission-> <http://company.com/name>,
 hasPropertyReadPermission-> <http://company.com/name2>
].

user1:User[hasRole->role1].
```

This means user1 has read/write permissions on the name property and read permission on the name2 permission.

If you want to allow all properties, you can use "*":

```
readAnyProperties:Role[hasPropertyReadPermission-> "*"].
```

This means that a member of role readAnyProperties can read all properties.
Permissions on properties are not inherited across the property hierarchy. Permissions need to be set for every single property.

Example:

If a user has read permissions for property a, but no read permissions for property b, and b::a, then the query

```
?- ?X[a->?Y].
```

also returns the values for property b.

There is a built-in _isPropertyPermitted/2 to explicitly check the permission for a property.

Examples:

```
?- _isPropertyPermitted(<http://company.com/name>, read). // check read
permission for property <http://company.com/name>

?- _isPropertyPermitted(<http://company.com/name>, write). // check write
permissions

?- _isPropertyPermitted(<http://company.com/name>, temp_write). // check write
permission for temporary facts and rules
```

13.5 Basic Configuration

The security configuration consists of three parts:

1. The OntoConfig.prp.
2. The security module.
3. The security context.

First we configure the OntoConfig.prp: In order to turn on the access control you have to set two switches

```
Security.AccessControl = on
Security.LoginRequired = on
```

and the path to the other configuration files (this is optional, per default the "conf" folder is used):

```
dir.conf = /home/projects/someProject/conf
```

13.6 Authentication/Login

If OntoBroker is configured for access control, a user has to login. The given username and password is then used by the security backend to verify the identity. When successfully authenticated, the user receives a ticket which allows him to execute any OntoBroker command (depending on his rights)

Authentication via webconsole

When authentication is switched on, then the webconsole shows you a dialog which allows you to specify your username and password. After the login the webconsole automatically handles the handling with tickets.

Authentication from commandline

If you work from the commandline then you have to specify the user name and the password for each command. The following examples assume that the user name is 'root' and the password 'open4me'. Execute the "isalive" command:

```
c:\test>command.cmd -l root:open4me isalive
[INFO]: Inference Server on port 2267 is up and running ...
```

Execute a query:

```
c:\test>query.cmd -l root:open4me "?-?X::Y@M."
["X","Y","M"],
[SomeSubConcept,SuperConcept,m1],
```

13.7 Configuration of the Security Context

The "conf" folder contains a file "security-context.xml". This configuration file allows the specification of the realms which should be used and the authentication strategy.

Properties Realm

The simplest scenario is to define users, their passwords and roles in a text file. For this scenario the XML file looks like:

```
<source lang="xml"> <?xml version="1.0" encoding="UTF-8"?> <beans>
   <bean id="OntoBrokerSecurityManagerConfig" class="com.ontoprise.security.
OntoBrokerSecurityManagerConfig">
      <property name="realms">
         <list>
            <ref local="PropertiesRealm"/>
         </list>
      </property>
      <property name="authenticationStrategy">
         <bean class="org.apache.shiro.authc.pam.FirstSuccessfulStrategy"/>
      </property>

</bean>
<bean id="PropertiesRealm" class="com.ontoprise.security.realm.PropertiesRealm"> <
property name="resourcePath"><value>security.properties</value></property> </bean> </
beans> </source>
```

The last few lines specify the path to the text file with the user configuration: "security.properties". The next section contains an example for the content of a "security.properties" file.

Security Module Realm

The security-context.xml file for the security module realm is

```
<source lang="xml"> <?xml version="1.0" encoding="UTF-8"?> <beans>
   <bean id="OntoBrokerSecurityManagerConfig" class="com.ontoprise.security.
OntoBrokerSecurityManagerConfig">
      <property name="realms">
         <list>
            <ref local="PropertiesRealm"/>
            <ref local="SecurityModuleRealm"/>
         </list>
      </property>
      <property name="authenticationStrategy">
         <bean class="org.apache.shiro.authc.pam.FirstSuccessfulStrategy"/>
      </property>

</bean>
```

```
<bean id="SecurityModuleRealm" class="com.ontoprise.security.realm.
SecurityModuleAuthorizationRealm"> </bean>
<bean id="PropertiesRealm" class="com.ontoprise.security.realm.PropertiesRealm">
<property name="resourcePath"><value>conf/security.properties</value></property> </
bean> </beans> </source>
```

Note that also the properties realm configuration is included because it is (currently) not possible to specify passwords in the security module realm. When the inference server starts up and you configured the security module realm you should see something like

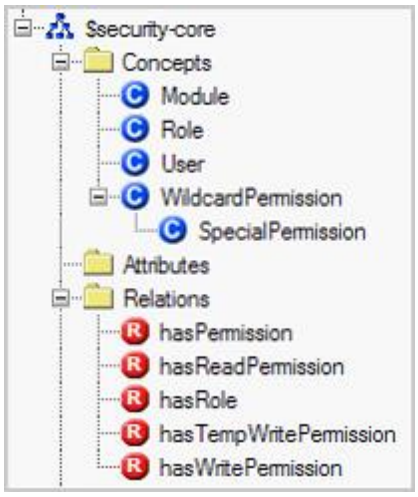
```
INFO [CONFIG] Loading security ontology: file:/C:/OntoBroker/conf/$security.obl
```

13.8 Modeling the Security Module Realm with OntoStudio

It is possible to edit the security ontology realm (which is ObjectLogic based) directly with OntoStudio. The schema of the security ontology is defined in $security-core.obl (which is per default in the "conf" directory). If you want to be able to edit the security ontology via OntoStudio you have to import the schema ontology. E.g.

```
:- module = '$security'.
:-prefix = "http://ontoprise.de/".
:-import = '$security-core'.

demo:User[hasRole->{m1all,m2read,foo}].
root:User[hasRole->allModules].

m1all:Role[hasReadPermission->m1; hasTempWritePermission->m1; hasWritePermission-
>m1].

m3read:Role[hasReadPermission->m3].
m3write:Role[hasWritePermission->m3].
foo:Role[hasPermission->'foo:*'].
```

The third line shows the required import. Here is a screenshot of the schema of the security ontology:



The following excerpt shows the ObjectLogic source of the schema ontology:

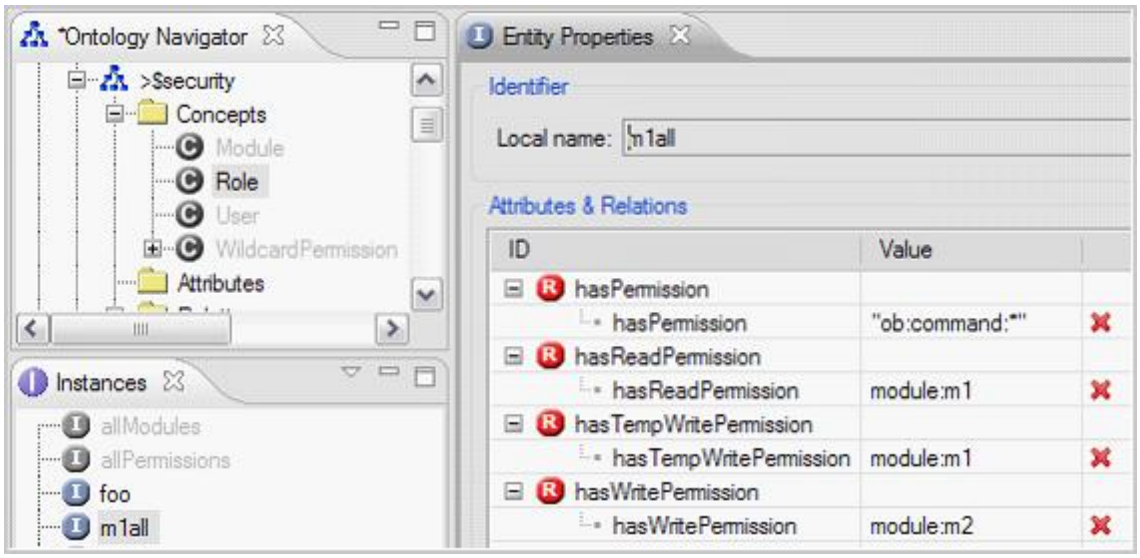
```
:- version("2.0").
:- encoding("Cp1252").

:- module = $'security-core'.


// Concept hierarchy ----------------------------------

Module[].
Role[].
User[].

Role[hasReadPermission {0:*} *=> Module].
Role[hasWritePermission {0:*} *=> Module].
Role[hasTempWritePermission {0:*} *=> Module].
Role[hasPermission {0:*} *=> WildcardPermission].
User[hasRole {0:*} *=> Role].

SpecialPermission::WildcardPermission.

// Instances ------------------------------------------
```

```
"*":Module.
allModules:Role.
allPermissions:Role.
"ob:command:apicommands":SpecialPermission.
"ob:command:busy":SpecialPermission.
"ob:command:clear":SpecialPermission.
"ob:command:commands":SpecialPermission.
...
```

The most important concepts are:

- User (a user can have multiple roles)
- Role (a rule consists of zero or more permissions)
- Module (known modules are instances of this concept)

To model your security ontology you also have to open the schema ontology and the ontologies for which you want to set the roles and permissions. Here is a screenshot of OntoStudio in which a security ontology is edited:



The changes in the security ontology (roles and permissions) are executed immediately when a permission/ role is changed. Hence, it is possible to try out new permissions immediately by executing queries.

13.9 Mapping Groups and Roles from External Realms like LDAP

If users and groups are managed externally, e.g. on an LDAP system, you often want to use the existing groups and attach permissions to them. This can be done in two steps. First of all, you write a rule which maps an external role of another realm to a role of the security module. Secondly you define permissions for the role. Currently we only support ActiveDirectory out-of-the-box, other LDAP systems need some project effort (a few hours to a few days) to adapt for the concrete LDAP schema. Let's assume your security configuration uses ActiveDirectory in the following way:

```
<source lang="xml"> <?xml version="1.0" encoding="UTF-8"?> <beans>
<bean id="OntoBrokerSecurityManagerConfig" class="com.ontoprise.security.
OntoBrokerSecurityManagerConfig">
      <property name="realms">
         <list>
            <ref local="ActiveDirectory"/>
            <ref local="PropertiesRealm"/>
            <ref local="SecurityModuleRealm"/>
         </list>
      </property>
      <property name="authenticationStrategy">
         <bean class="org.apache.shiro.authc.pam.FirstSuccessfulStrategy"/>
```

```
      </property>

</bean>
```

```
<bean id="SecurityModuleRealm" class="com.ontoprise.security.realm.
      SecurityModuleAuthorizationRealm">
</bean>
<bean id="ActiveDirectory" class="org.apache.shiro.realm.
      activedirectory.ActiveDirectoryRealm">
      <property name="ldapContextFactory">
       <bean class="org.apache.shiro.realm.ldap.DefaultLdapContextFactory">
      <property name="searchBase">
       <value>dc=ads,dc=yourcompany,dc=com</value>
      </property>
      <property name="systemUsername">
            <value>admin</value>
      </property>
      <property name="systemPassword">
            <value>pass</value>
      </property>
      <property name="principalSuffix">
            <value>@ads.yourcompany.com</value>
      </property>
      <property name="url"><value>ldap://ads1.yourcompany.com:389/</value>
      </property>
</bean>
      </property>
      <property name="searchBase">
            <value>DC=ads,DC=yourcompany,DC=com</value>
      </property>
      <property name="principalSuffix">
            <value>@ads.yourcompany.com</value>
      </property>
      <property name="groupRolesMap">
  <map>
   <entry key="CN=group1,OU=Karlsruhe,DC=ads,DC=yourcompany,DC=com" value="role1"/>
   <entry key="CN=group2,OU=Karlsruhe,DC=ads,DC=yourcompany,DC=com" value="role2"/>
  </map>
      </property>
 </bean>
 <bean id="PropertiesRealm" class="com.ontoprise.security.realm.PropertiesRealm">
      <property name="resourcePath">
            <value>conf/security.properties</value>
      </property>
 </bean>
</beans>
</source>
```

In this example configuration for the userPrincipalName the memberOf attribute is looked up and group memberships in the "groupRolesMap" are mapped to role names (here "role1" and "role2"). To use these ActiveDirectory (AD) groups in the realm SecurityModuleRealm you write rules to map a group to one or more roles of the SecurityModuleRealm. In the following example the AD role "role1" is mapped to the SecurityModule roles #m1all and #m3read. For this purpose the builtins _hasRole/1 and _currentUser/1 are used. The builtin _hasForeignRole is true, if the user has a role with the given name in any of the configured realms. The builtin _currentUser/1 returns the user in the given query/session context.

```
- module = '$security'.
- prefix = "http://ontoprise.de/security#".
- import = '$security-core'.
... // similar as above
// if current user U has the role "Administrator" (role name of any configured realm)
// then the user should be assigned to the internal roles #m1all and #m3read ?- U U:
#User[hasRole->{m1all,m3read}] <- _currentUser(U) and _hasForeignRole(role1).
```

In OntoBroker the _hasRole/1 Builtin has been renamed to _hasForeignRole/1 and it only applies to external

roles, e.g. roles defined in ObjectLogic are not considered.

```
// ObjectLogic
- module = $security.
- importmodule '$security-core'.
... // similar as above
// if current user U has the role "Administrator" (role name of any configured
realm)
// then the user should be assigned to the internal roles #m1all and
#m3read ?U:User[hasRole->{m1all,m3read}] :- _currentUser(?U) and _hasForeignRole
(role1).
```

13.10 Access Restrictions for Commands

It is also possible to restrict the OntoBroker server commands. For example, the command "shutdown" should only be executed by an administrator.

OntoBroker Commands Via Property File Realm

In this case you can put into your "security.properties" file something like

```
user.Joe = pwd_joe, manager
user.Jane = pwd_jane, powerUser

role.manager = ob:command:*,ob:collabserver:*
role.powerUser = ob:command:reload,ob:command:add,ob:command:del,ob:command:query
```

This configuration basically says that Jane is allowed to execute the following OntoBroker commands and nothing else:

- reload
- add
- del
- query

Joe is allowed to execute any OntoBroker commands (including "shutdown"). The permission "ob:command:*" is a so-called wildcard permission.

Note:
If you are working with security access and you are using the wildcard permission, users who have only reading permission can also read the security ontology and therefore know which user has which permission. If you do not want this behaviour, you must not use the wildcard permission. Instead you have to manually set permission for each module.

OntoBroker Commands Via Security Module Realm

The access rights configuration of commands can also be configured with the security module realm. In this case the security ontology for the scenario above is

```
:- module = $security.
:- importmodule $'security-core'.

... // similar as above

// if current user U has the role "role1" (role name of any configured realm)
// then the user should be assigned to the internal roles m1all and m3read
?U:User[hasRole->{m1all,m3read}] :- _currentUser(?U) and _hasForeignRole(role1).
```

13.11 Support for multiple principalSuffix in ActiveDirectory

The "userPrincipalName" attribute is used for login and querying user roles. It is built by concating username and a principalSuffix. Large ActiveDirectory installations may use multiple user principal name suffices. For such scenarios, OntoBroker 6.3 now supports a list of user principal name suffices. In this case, the configuration for the ActiveDirectory realm may look like this:

```
<bean id="ActiveDirectory" class="com.ontoprise.security.realm.ActiveDirectoryExRealm
">
       <property name="ldapContextFactory">
          <bean class="com.ontoprise.security.realm.AdLdapContextFactory">
             <property name="searchBase"><value>dc=mycompany,dc=com</value></property
>
             <property name="systemUsername"><value>admin</value></property>
             <property name="systemPassword"><value>admin</value></property>
             <property name="principalSuffixList">
                <list>
                   <value>@ads.mycompany.com</value>
                   <value>@ads2.mycompany.com</value>
                </list>
             </property>
             <property name="url"><value>ldap://domaincontroller.mycompany.com:389/</
value></property>
          </bean>
      </property>
      <!-- search base for groups -->
      <property name="searchBase"><value>OU=Karlsruhe,DC=ads,DC=mycompany,DC=com</
value></property>
      <property name="groupRolesMap"><map>
         <entry key="CN=group1,OU=Karlsruhe,DC=ads,DC=mycompany,DC=com" value="group1
"/>
         <entry key="CN=group2,OU=Karlsruhe,DC=ads,DC=mycompany,DC=com" value="group2
"/>
      </map></property>
   </bean>
```

13.12 Other Access Restrictions

As a default it is no longer possible to use query options like "trace". But it is possible to give some roles the appropriate access rights.

```
Joe:User[hasRole->m1all,debug].
debug:Role[hasPermission->'ob:reasoner:debug'].
```

This security ontology specifies that the user "Joe" has the rights to use debugging query options (e.g. "trace").

13.13 Permissions

If `Security.LoginRequired = on` and `Security:AccessControl = on`, these permissions authorize special actions and commands when the OntoBroker is accessed via Collabserver, socket API or OntoBroker web service.

ob:collabserver:addontology	For collabserver / remote API you now need the permission "ob:collabserver:addontology" to create or import ontologies. Alternatively you are allowed to perform these actions if you have write permissions on the ontology to be created or imported.
ob:command:query	Permission holder is allowed to perform query command
ob:command:xy	Permission holder is allowed to perform command named "xy". xy is a placeholder here for any of the OntoBroker commands like query, insert, delete, ...
ob:command:*	Permission holder is allowed to perform any command
ob:collabserver:cancelpendingopen	Permission holder is allowed to call method OntologyManager. cancelPendingOpen() via Collabserver
ob:collabserver:longtransaction	Permission holder is allowed to start long transactions via Collabserver
ob:collabserver:sessioninfo	Permission holder is allowed to call method ClientSessionManager.getActive() via Collabserver
ob:collabserver:addontology	Permission holder is allowed to load or create any new ontology via Collabserver
ob:collabserver:*	Permission holder has all collabserver permissions
ob:reasoner:debug	Permission holder is allowed to activate tracing on query execution
ob:reasoner:*	Permission holder has all reasoner permissions

Log User Login/Logout

This feature will do the following: If (and only if) the following line in conf/log4j.properties

```
log4j.category.[LOGIN]=WARN
```

is changed to

```
log4j.category.[LOGIN]=INFO
```

then a user login/logout is logged. This feature only makes sense when the server is started with

```
Security.LoginRequired = on
```

Note:
The logging is not failsafe: E.g. the log-out is typically not logged when a timeout occurs.

13.13.1 Custom Permissions

You can define and use custom permissions on module level as follows:

1. Add definitions of custom permissions to the `security-context.xml`.

Add a property customPermissionDefinitions to the bean SecurityModuleRealm in security-context.xml.

Example:

```
<bean id="SecurityModuleRealm"
class="com.ontoprise.security.realm.SecurityModuleAuthorizationRealm">
        <property name="customPermissionDefinitions">
            <list>
                <value>deploy</value>
                <value>undeploy</value>
```

```
            </list>
        </property>
    </bean>
```

In this example definitions for the custom permissions "deploy" and "undeploy" have been added.

2. Use these custom permissions in roles of the security ontology($security.obl).

Example:

```
role1:Role[customPermission("deploy")->module1,customPermission("undeploy")->"*"].
```

In this example a role "role1" is defined with custom permission "deploy" for module module1 and custom permission "undeploy" for all modules.

3. a) Use _isCustomPermitted/2 to use the custom permission in your rules or queries:

Example:

```
?- _isCustomPermitted(module1, "deploy").
```

This query would return true if user has above defined role1

3. b) Use custom permissions programmatically with the OntoBroker API.

```
import com.ontoprise.security.CustomPermission;
SessionFacade sf = ....;
Term module = ...;
CustomPermission custperm = new CustomPermission(module, "deploy");
boolean permitted = SessionFacadeSecurityHelper.isPermitted(sf, custperm);
```

13.14 Terminology Glossary

A short overview of the terminology. To use advanced features of Apache Ki you should be familiar with some of the terminology.

- Authentication

  Authentication is the act of confirming the identity of a subject (a user). The most common method of confirming an identity is with a username/password combination (it is checked if the password is correct).

- Authorization

  This is the act of confirming if a subject has the right permissions to execute some action. For example, a check is made that user "Joe" has the permission to execute the "shutdown" command.

- Realm

  A realm is a resource which allows access to the security components of an application (users, passwords, roles, permissions). So realms are responsible for both authentication and authorization.

- Subject

  A subject represents an user with all his roles and permissions.

- Role

  A user can have multiple roles. For example, a user can have the role "developer", but he can also be responsible for the administration (role: administrator) or for the sales (role: sales). A role is basically a collection of permissions.

- Principial

  Principials are the identifying attributes of a subject (name, id, ...).

- Credidentials

Information which is used to verify the identity of a user.

- Permission

The ability to perform an action. In OntoBroker permissions could be

- Read access to a module
- Write access to a module
- Command execution access

OntoBroker also allows wildcard permissions for some features. For example, you can use '*' as a wildcard to specify modules:

Joe:User[hasRole-> ResearchAndDevelopmentMember].

ResearchAndDevelopmentMember:Role[hasReadPermission->*].

It is also possible to use wildcard permissions for command executions:

role.admin = ob:command:*

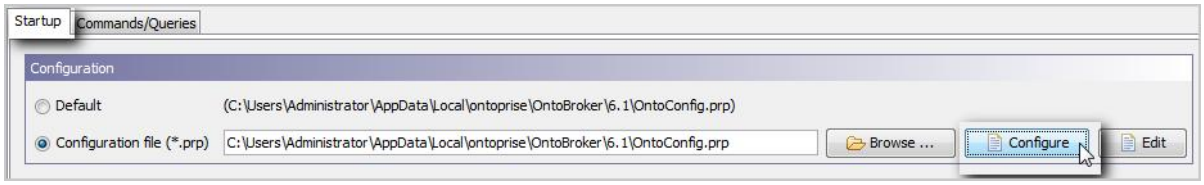
Instead of specifying a list of allowed commands we grant the "admin" role access to all commands.

14 Configuring OntoBroker

OntoBroker has several switches to influence its runtime behaviour. All switches can be set in the configuration file <OntoConfig.prp> which resides in the starting directory of the server (usually the installation base directory of the server). The switches can be grouped into mainly three categories. First the user can enable the use of the internal database. A second group enables the user to modify the performance behaviour of the server. These switches depend on the use of the server and have to be used carefully. The third group allows the evaluation mode and features to be influenced. This chapter gives an overview of the different switches, well as a brief description.

14.1 Configuration GUI

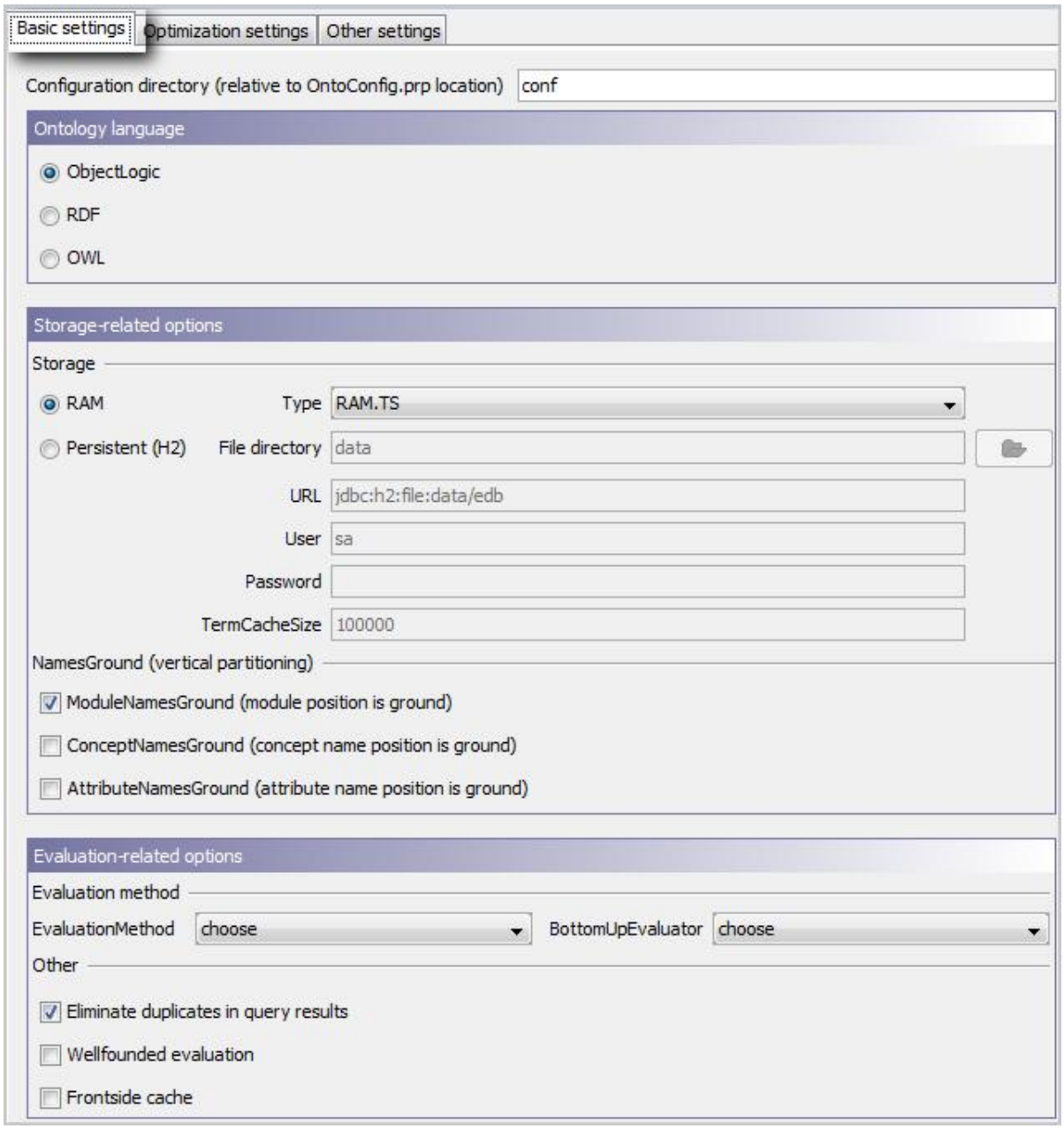
OntoBroker offers you an easy to use configuration interface. Start OntoBroker and click on "Configure". The configuration GUI appears. Every new configuration can be saved. Simple choose **Save** and your custom configuration will be saved to the configuration file.



14.1.1 Basic Settings

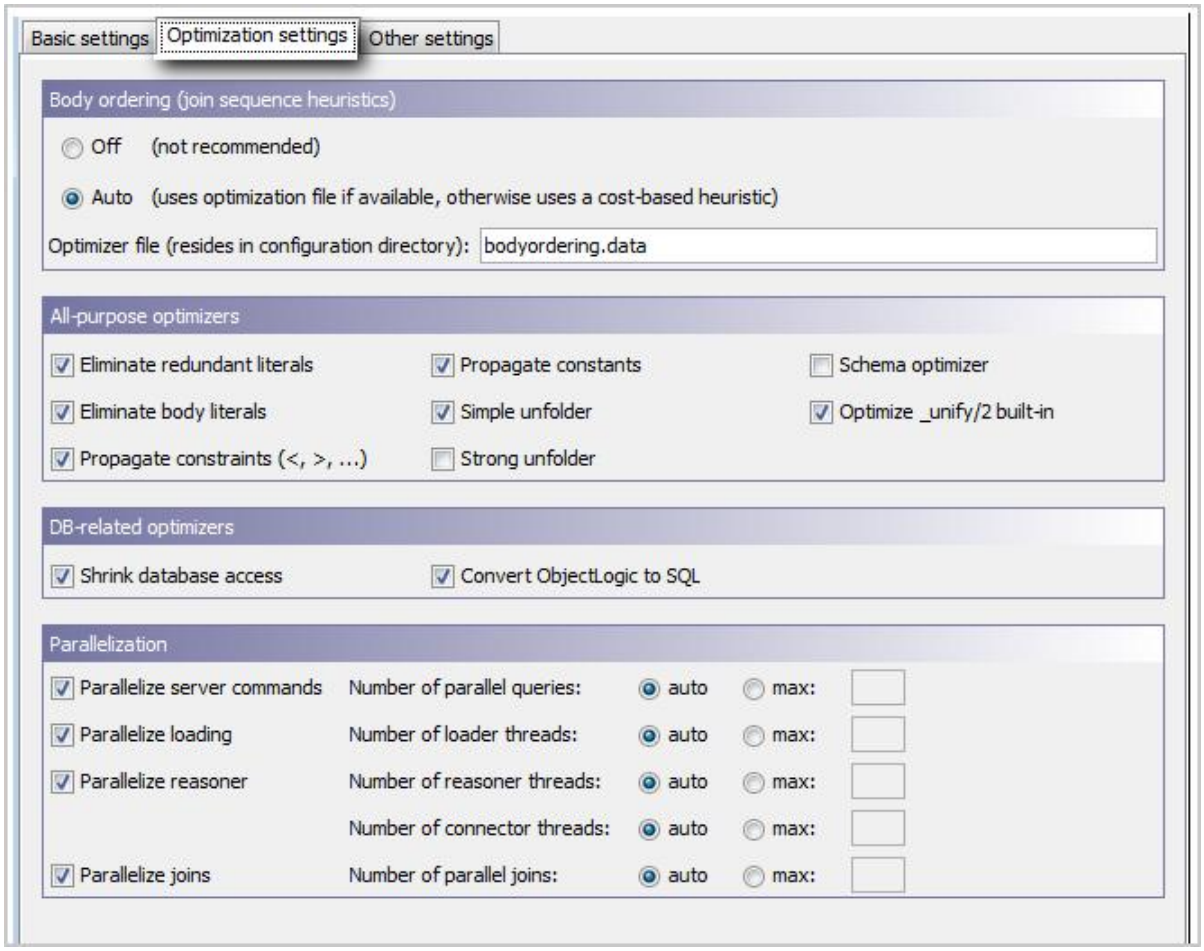
The basic settings configuration allows you to change various preferences to suit your needs. For most users, the default settings are adequate. For more information, see:

- Ontology Language 12
- Storage-related Options 44
- Evaluation-related Options 167



14.1.2 Optimization Settings

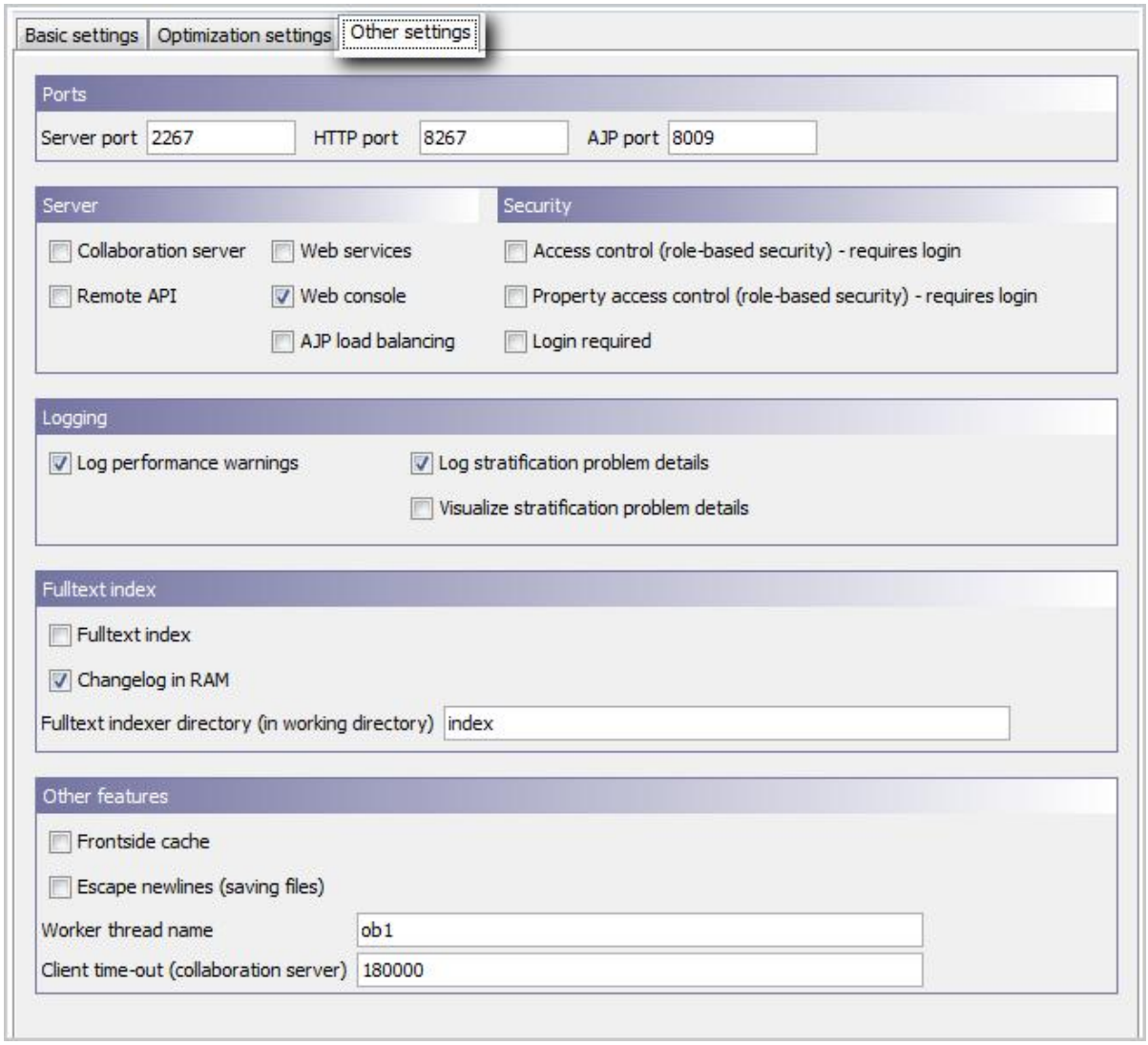
OntoBroker supports multiple optimizers (most of them just rewrite a set of rules to an optimized set of rules). However, in some cases optimizers just waste time and space. For example, if you are not accessing a database then you do not need to use the database optimizers. You can find all information on optimizers here [170].



14.1.3 Other Settings

Within the 'Other settings'-tab further OntoBroker settings can be configured. Furthermore the OntoBroker Socket Communication and the Administration Console HTTP port OntoBroker should use on start-up, can be set. For more information see

- Server
- Security 144
- Logging 273
- Fulltext Index 274



14.2 Parallelization Settings

OntoBroker can use multiple CPUs for processing executing server commands, for loading ontologies, and for parallelizing evaluation of a query. Each of these three areas can be configured separately using configuration switches.

Command	Function
`Parallelize.ServerCommands = <on | off>`	If this switch is on, OntoBroker can process multiple server commands (e.g. queries) from clients at the same time.
`Parallelize.ServerCommands.MaxThreads = <integer|auto>`	This switch determines the maximum number of threads used for parallelizing server commands (only if Parallelize.ServerCommands = on). The default value is "auto", i.e. automatic configuration depending on the number of available CPUs (cores).
`Parallelize.Loader = <on | off>`	This switch controls the loading of multiple ontologies. If it is turned on, OntoBroker can load multiple ontology files on a parallel basis. For persistent storages (e.g. H2), this feature is always turned off automatically, as it has a negative impact on performance.
`Parallelize.Loader.MaxThreads = <integer| auto>`	This switch sets the maximum number of threads used for loading ontology files.
`Parallelize.Reasoner = on | off`	This switch controls the evaluation of queries. If it is turned on, the evaluation of a query may be distributed on multiple threads wherever possible.
`Parallelize.Reasoner.MaxThreads = <integer|auto>`	This switch determines the overall number of threads available for executing query evaluation tasks.
`Parallelize.Reasoner.Connectors. MaxThreads = <integer|auto>`	This switch determines how many threads are available for connectors. Every connector call is processed in a seperate thread. As connectors communicate with external resources, this switch controls the load OntoBroker can produce on these resources.
`Parallelize.Joins = on | off Parallelize. Joins.MaxThreads = <integer> | auto`	Some evaluation methods support two types of parallelization: operatornet parallelization (this is the OntoConfig switch "Parallelize.Reasoner") and join-level parallelization. For the latter there is now an explicit switch.

14.3 Performance-Related Configuration Switches

14.3.1 Genetic Optimizer

The genetic optimizer allows the optimization of specific queries. This optimizer is typically used before deploying ontologies on a production server. The order of the body literals can be changed/permutated and this might improve the inference speed dramatically. If you create all possible permutations, you may find the perfect result, but this would take an extremely long time. For 10 rules with 4 body literals you would have 104! possible solutions, for 100 rules with 8 body literals you would have 1008! possible solutions. The idea is, that you do not need the best result, but a result that is good enough. If you get one change that improves the query, it should be even better when combined with other positive changes. For optimizations in such high dimensional search spaces genetic algorithms are applicable. Thus the genetic optimizer optimizes the sequence of body literals in the rules using a genetic approach. In the initialization process a random population will be created. This population is a defined number of individuals with each representing exactly one permutation. These individuals together are called a generation, as it is the first one this one, is generation 0. Then the fitness of all individuals will be determined. For this, the permutations will be applied and the query will be executed. The smaller the evaluation time is, the better the fitness. The better the fitness of an individual is the better is its chance to contribute to the following creation of the individuals of the next generation. Their characteristics, the ordering of body literals, will be mixed and a new population will be created. New random changes may be made. The fittest individual survives

unchanged. This continues until the wanted number of generations has been calculated.

Limits

As the continuum of all possible reordering is very big, you will need many individuals to test. For complex rule scenarios several thousand individuals should be created, so that it takes a lot of time to optimize a long-running query. Another problem may be the memory demand of the query evaluation. Some permutations will need much more memory than the average. If you already operate on the limits OntoBroker can handle with the available main memory, you might encounter out of memory" errors when using genetic optimizer.

NOTE: If you optimize a specific query with the genetic optimizer and you change the rules or the facts then you will need to execute the genetic optimizer again. If you only change the constants or the order of literals in a rule you do not need to optimize the query again.

Usage

Command	Description
`opt_genetic <queryID>`	Commands to send an optimized query to OntoBroker
`opt_genetic status`	The return value is a string. The status of the last complete generation will be returned. At least one generation has to be evaluated before valid results can be delivered. OntoBroker Return Values: The return value is a "string" without line separators. Use | as separation character.
`opt_genetic kill`	Stops a running optimization. It may take a short time, you can use the -isoptimizing command to see when it is done. The timeout window has to fit into the timeout of the socket connection, so that it may be too small for some cases.
`opt_genetic isoptimizing`	Returns if an optimization process is currently running or not. OntoBroker Return Values: The return value is a "string". Example: `opt_genetic file, "test.data", rewrite, timeout, 1000, module,"<http://www.NewOnto1.org/ontology>", "queryID"`

Options	Description
`file filename`	file name for optimisations data
`rewrite`	rewrites the file with optimisations data if exists
`timeout value`	optimization timeout (ms)
`kill`	stops the optimization
`isoptimizing`	checks whether the optimization is already running
`status`	returns the best result of the running optimization

14.3.2 EliminateDuplicates

This switch can be set in OntoConfig.prp via:

```
EliminateDuplicates = <on|off>
```

During the evaluation process of a query (sub-) results may be inferred in different ways (e.g. a tuple can be stored in some facts table or it can be inferred by a rule). Duplicates can also occur as the result of join operations. If EliminateDuplicates=on (this is the default) then all duplicates are removed from the result tuples. Removing duplicates might be a costly operation and there are some use cases where you simply do

not care about duplicates. In this case it is recommended to set EliminateDuplicates=off as the result is returned faster.

14.3.3 ConceptNamesGround

This switch can be set in OntoConfig.prp via:

```
ConceptNamesGround = <on|off>
```

This switch can improve performance significantly in most cases, but in some cases it can also deteriorate the performance.

Normally OntoBroker has one table for all concept-instance pairs called <$assertedisa>. This means if you have the following instances:

```
Tom:Cat.
Garfield:Cat.
Jerry:Mouse.
SpeedyGonzales:Mouse.
```

then they are stored in a single table "$assertedisa":

```
$assertedisa
-----------------------
Tom             Cat
Garfield        Cat
Jerry           Mouse
SpeedyGonzales  Mouse
```

The ConceptNamesGround switch will change this to

```
$assertedisa_Cat
--------
Tom
Garfield

$assertedisa_Mouse
--------------
Jerry
SpeedyGonzales
```

When using `ConceptNamesGround=with` the consequence that no rules are no longer allowed which have nonground arguments at concept positions in rule heads.

The reason is that OntoBroker would have to create a new table for each value of Y which is not yet in the database. To keep it simple: If you have such rules in your ontology you cannot use the ConceptNamesGround switch (OntoBroker will detect this and will refuse to start). Why would you want to use the ConceptNamesGround switch? Well, there are two reasons. The first one is the improved performance. It is simply cheaper to query the tables if you have fewer restrictions. E.g. if you want to retrieve all cats from OntoBroker then you pose the query:

```
?- ?X:"Cat".
```

When ConceptNamesGround is off, then OntoBroker must scan the table "$assertedisa" and check for each element if the second parameter is "Cat". (Of course OntoBroker uses indices for accessing tables but the general idea is that the queries are cheaper if the tables are smaller.) If ConceptNamesGround is used OntoBroker only needs to query a single table to get the results. So this is a typical example of a query which can be evaluated faster if ConceptNamesGround is used. The more concepts you have the cheaper a query for instances becomes. However, there are some types of queries which are slower when ConceptNamesGround is used. If you do not use a concept restriction for your queries then the performance degrades. Consider the following query:

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

This kind of query is slower when `ConceptNamesGround` is used because in this case OntoBroker must query all "$assertedisa" tables ($assertedisa_Cat, $assertedisa_Mouse, ...). When `ConceptNamesGround` is turned off then OntoBroker must only query a single table. So basically you have to decide between good performance for queries with restrictions or good performance for queries without restrictions. The second reason for choosing `ConceptNamesGround=on` is that it is much easier to deal with negations in rules. If you

ever encounter a "StratificationException" then switching to ConceptNamesGround=on might help.

14.3.4 AttributeNamesGround

This switch can be set in OntoConfig.prp via:

```
AttributeNamesGround = <on|off>
```

This switch works exactly in the same way as "ConceptNamesGround", but now attribute names are ground and not concept names. The attributes are stored in the table <$att_>. If you have statements like

```
Cat[likes=>Mouse].
Mouse[isAfraidOf=>Mouse].
```

then we have the table "$att_" for AttributeNamesGround=off and "$att_likes" and "$att_isAfraidOf" for AttributeNamesGround=on.

The performance characteristics regarding queries with restrictions on attribute names and regarding queries without restrictions on attribute names are basically the same as with "ConceptNamesGround". If <AttributeNamesGround> is used there may not be a rule with a head containing a variable for the relation name as these cannot be reinterpreted during compile time. Note that this option is not supported for OntoStudio.

14.4 Wellfounded Evaluation

OntoBroker is able to deal with rules which contain rule cycles over a negated literal. In order to evaluate a query over these rules OntoBroker must use a special evaluation strategy called "well-founded evaluation". A well-founded evaluation needs much more time and should be avoided if possible. You can also dissolve cycles using negations and aggregations. If you must use such rules you need to set the following configuration switch:

```
WellfoundedEvaluation = on
```

However, you should be aware of the performance penalty for such ontologies ("unstratified models").

A typical example of an unstratified model is

```
person(a).
male(?X) :- person(?X) AND NOT female(?X).
female(?X) :- person(?X) AND NOT male(?X).
```

Here it is not clear which rule to evaluate first. If you query for "person" the result is "a", but if you query for "male" or "female", the result set is empty. The reason is a cycle over negations. This rule graph can be displayed by setting the switch "VisualizeStratificationProblemDetails=on" and starting OntoBroker with "WellfoundedEvaluation=off". If OntoBroker then detects such cycles the cycles are displayed.

14.5 Using OntoBroker with a Persistent Datamodel

OntoBroker currently supports one persistent datamodel: H2 (Embedded SQL engine)

The primary reasons for using a persistent datamodel are:

- The amount of facts and rules is too big to store everything in the main memory.
- The server should store data persistently.

Of course OntoBroker will only fetch the data which is required for reasoning from the persistent storage. This allows the handling of data models which are larger than the available main memory. When you start the OntoBroker server then the given files are stored directly in the persistent storage. Hence, you do not need to give the files as commandline options the next time you start OntoBroker.

14.5.1 Configuration

To use H2 as storage in OntoBroker, you have to set at least two values in your OntoConfig.prp:

```
Storage = H2
H2.URL = jdbc:h2:file:<path>/<dbname>
```

The path can be omitted or can be relative to the working directory. As a folder separator you should use the normal slash / instead of the backslash \.

1. Example:

```
H2.URL = jdbc:h2:file:data/edb
```

This will create the database files in the subdirectory data (relative to the working directory) and all database files will start with "edb".

2. Example:

```
H2.URL = jdbc:h2:file:d:/home/storage/mydb
```

This will create the storage files in the directory d:\home\storage and all storage files will start with "mydb".

H2 may create, enlarge and delete files all the time, even if you are using OntoBroker with reading access only. Therefore OntoBroker with H2 needs create and write permissions on the directory where the storage files are located.

14.5.2 Database File Layout

This is an excerpt of the H2 website describing the database file layout. The following files can be created by the database:

File Name	Description	Number of Files
`test.h2.db`	Database file. Contains the transaction log, indexes, and data for all tables. Format: <database>.h2.db	1 per database
`test.lock.db`	Database lock file. Automatically (re-)created while the database is in use. Format: <database>.lock.db	1 per database (only if in use)
`test.trace.db`	Trace file (if the trace option is enabled). Contains trace information. Format: <database>.trace.db Renamed to <database>.trace.db.old is too big.	0 or 1 per database
`test.lobs.db/*`	Directory containing one file for each BLOB or CLOB value larger than a certain size. Format: <id>.t<tableId>.lob.db	1 per large object
`test.123.temp.db`	Temporary file. Contains a temporary blob or a large result set. Format: <database>.<id>.temp.db	1 per object

14.5.3 Configuration Values for Performance Tuning

The most important parameter influencing the performance is the number of database pages, which H2 can keep in the main memory. By default this value is set to 16384 KB. To change this value, add the CACHE_SIZE option to the H2.URL. The value is the maximal cache size in KB.

Example (set cache size to 100000 KB, e.g. about 100 MB):

```
H2.URL = jdbc:h2:file:data/edb;CACHE_SIZE=100000
```

Furthermore, you may change the following configuration values in OntoConfig.prp:

```
H2.TermCacheSize = 100000
```

These parameters set the cache size for the translation between term ids used in the database and term objects in OntoBroker. There are two caches, one used mainly for the first argument of predicates and one for all other arguments. If you have lots of main memory, you may increase these cache sizes.

14.5.4 Backup

If OntoBroker is not running, you can backup the H2 database simply by copying all of the database files. In fact it is even enough to backup the <dbname>.data.db file. However, in this case H2 will regenerate the database index file <dbname>.index.db containing all indices during the next startup of OntoBroker. Recreating indices may take some time, depending on the database size and computing power of your computer (typically a few minutes per GB).

14.5.5 Online Backup

An online backup can be triggered using the command "h2backup".

Note:
The command "h2backup" is only available if OntoBroker is running with H2 as storage.

Usage:

```
h2backup <output-name>
```

The single argument specifies the target filename of the database copy.

Example:

```
h2backup d:\\backup\\mybackup.data.db
```

This backups the database file to "d:\\backup\\mybackup.data.db."

CAUTION:
All files like "d:\\backup\\mybackup.*" are deleted during the copy action to prevent the possible mixing of old and new temporary database files.

Restoring a backup requires following steps:

1. Shutdown OntoBroker.
2. Delete all H2 database files, e.g. if the database is named "edb", you have to delete "edb.*".
3. Copy the backup file to the old location and rename it to the original name. E.g if the database is named "edb", then the database file must be named "edb.data.db".
4. Start OntoBroker. On opening the database file, the indices will reconstructed. This procedure may take some time (typically a few minutes/ GB).

14.6 Configuration Switches for the Server

OntoBroker runs as a server and hence has to listen to a given port for queries and commands to execute. The standard port for OntoBroker is port <2267> which is registered at the IANA organization. Nevertheless the user is able to change the port in order to run the server on a different port anyway. This might become necessary if several instances of the server have to run on the same host.

```
CommandPort = 2267
```

Setting Socket Timeout

This is an advanced property which should not be changed without good reasons. The default socket timeout for the command port is set to 0, meaning "no timeout". If there are problems with unusual client behaviour, e.g. connecting OntoBroker without sending a command, a solution can be to set the timeout to an appropriate value (which depends on the number of clients and the network latency).

```
SocketTimeout=<milliseconds>
```

Switch for the Configuration of the Ports

If OntoBroker is started with one of the commandline options "-webservice", "-collabserver", or "-

webconsole", OntoBroker listens to an additional port for providing the web service API. This port is 8267 by default and can be changed using the switch WsHttpPort = <port>.

```
WsHttpPort= 8267
```

This switch can be overruled by the commandline option "-wsport <port>".

15 OntoBroker Performance Tuning

The performance of OntoBroker can be tuned by trying out different configuration switches (see the chapter "Configuring OntoBroker"). It is often useful to check where the bottleneck of a specific query really is. You can use the tracing features to, for example, find out if a specific built-in is the cause of the performance problem (see "Tracing the evaluation process"). Besides this basically you have the following options:

- Using tools to optimize the sequence of body literals in rules
- Materializing the rules on startup in order to reduce the query execution times during runtime

Profiling

If you execute a query and you want to know where OntoBroker spends most of the query time then use the query option "profile". The chapter "Query Option profile" 67 gives detailed information about the profiling feature. This is usually the best starting point for understanding the performance characteristics of queries.

15.1 Choosing an Evaluation Method

15.1.1 Evaluation Methods

The OntoBroker reasoning engine supports different evaluation strategies. This allows the user to choose the evaluation method which provides the best performance characteristics for a specific ontology with a specific set of queries. The evaluation method may also be chosen individually for every query. The following sections give an overview of the different evaluation methods.

15.1.1.1 BottomUp

BottomUp is the simplest evaluation strategy. In many cases this method provides the best performance of all of the evaluation methods. Its biggest disadvantage is that no variable bindings are propagated top-down.

E.g. if you have a rule and a query

```
RULE r1:  p(?X,?Y) :- q(?X,?Y) AND r(?X,?Y).
QUERY q1: ?- p(?a,"Y").
```

then first the rule "r1" is evaluated and all of the implications are computed. Then the body literal "p(a,Y)" of the query "q1" filters all tuples generated by "r1" which have an "a" in the first position. So, if rule "r1" produces many tuples, but only few tuples which contain an "a" in the first position, then a lot of time will be wasted in "r1". Besides this performance problem note that it is not possible to evaluate all rules bottom-up (e.g. if built-ins are involved). In order to start OntoBroker with this evaluation method set

```
EvaluationMethod=BottomUp
```

in OntoConfig.prp.

OntoBroker 6 ships with the new BottomUp3 evaluator. It only works for

```
Storage = RAM.TS
```

If RAM.TS is used, the default bottomup evaluator will be BU3 (BU3 only works with RAM.TS). You should not mix RAM.TS with DynamicFiltering or DynamicFiltering2 or with the BottomUpEvaluator BottomUp2.

15.1.1.2 MagicSet

The MagicSet method (we use the so-called "generalized supplementary magic sets") tries to improve the performance of the BottomUp evaluation by propagating bindings top-down. For that purpose additional rules are generated which simulate the top-down propagation. For example, if you have the rules

```
RULE r1:  p(?X,?Y) :- q(?X,?Y) AND r(?X,?Y).
QUERY q1: ?- p(a,Y).m_p_bf(a).
RULE r1:  pbf(?X,?Y) :- m_pbf(?X) AND q(?X,?Y) AND r(?X,?Y).
QUERY q1: ?- p_bf(a,Y).
```

This strange-looking program optimizes the original program by propagating the "a" to then the MagicSet transformation transforms this rule into something like the rule "r1". This means rule "r1" will only create

tuples which already have an "a" at the first position. This program is then evaluated with a bottom up evaluator. So MagicSet additionally requires you to choose an appropriate bottom up evaluation method (see below). In order to start OntoBroker with this evaluation method set

```
EvaluationMethod=MagicSet
```

in OntoConfig.prp

15.1.1.3 DisjunctiveMagicSet

This is another variant of the MagicSet algorithm. This algorithm implements the same idea as the original MagicSet algorithm, but generates less rules. Again the rewritten program is then evaluated with a bottom up evaluator. So DisjunctiveMagicSet additionally you requires to choose an appropriate bottom up evaluation method (see below). In order to start OntoBroker with this evaluation method set

```
EvaluationMethod=DisjunctiveMagicSet
```

15.1.1.4 DynamicFiltering2

This is a more memory-efficient and fast version of the proven DynamicFiltering evaluation method. In most cases you should use DynamicFiltering2 instead of DynamicFiltering. This evaluation method basically implements the ideas of the MagicSet algorithm directly in the inference kernel without generating additional rules. This has two important advantages:

- Less rules have to be evaluated
- The evaluation operates on the original rules which makes it easier to understand the evaluation

Currently DynamicFiltering2 is the best choice for persistent storage systems. In order to start OntoBroker with this evaluation method set

```
EvaluationMethod=DynamicFiltering2
```

in `OntoConfig.prp`.

15.1.1.5 Choose

This evaluation method checks the program characteristics and decides which evaluation method (BottomUp, DisjunctiveMagicSet or DynamicFiltering) should be used for a specific query. Some of the characteristics which will be checked are:

- Is the program bottomup evaluable?
- Does the program contain function symbols?
- Does the program contain stratified or wellfounded negations?
- Do we need to create explanations?
- Do the rules have disjunctions in the head?
- How many rules are in the program?
- How selective is the query?

Typically "choose" will select an appropriate evaluation method, so setting. You have to be aware that “choose” uses heuristics to decide on an evaluation method. Heuristics can fail. So it may be the case that “choose” does not find the optimal evaluation strategy.

```
EvaluationMethod=choose
```

15.1.2 BottomUp Evaluators

Some of the evaluation methods described above can be evaluated with an (additional) bottom-up evaluator only:

- BottomUp
- DisjunctiveMagicSet

It's pretty clear that the "BottomUp" evaluation method can be evaluated using a bottom-up evaluator. But why can the magic set methods be evaluated with a bottom-up evaluator? The reason is that the magic set methods just rewrite the original program to another program which can be evaluated faster using a bottom-up evaluator. So "MagicSet" and "DisjunctiveMagicSet" are just optimizers for the bottom-up evaluation of rules.

OntoBroker 6 ships with the new BottomUp3 evaluator. It only works for

```
Storage = RAM.TS
```

If RAM.TS is used, the default bottomup evaluator will be BU3 (BU3 only works with RAM.TS). RAM.TS is more memory efficient than RAM.AVL.Packed. In this case BottomUp3 might be the best choice for you.

15.1.2.1 BottomUp Evaluator

This evaluator is the fastest bottom-up evaluator which is available in OntoBroker. It is

- Very memory efficient
- Highly efficent on multicore/multi CPU systems (it provides in-kernel parallelization)

BottomUp2 is able to execute multiple join operations at the same time. BottomUp2 is the recommended bottom-up evaluator for the inference server. In order to start OntoBroker with this evaluator set

```
BottomUpEvaluator=BottomUp2
```

in OntoConfig.prp. In most cases it is sufficient just to select "auto" (then the number of parallel reasoning threads is automatically adapted to the available number of CPUs).

15.1.2.2 DBBottomUp Evaluator

This bottom-up evaluator will execute all join and match operations inside the storage system. It is currently the best evaluator for materializing very large ontologies with a database. However, it is also an efficient evaluator for the RAM storage systems. DBBottomUp works for all RAM models and for H2. In order to start OntoBroker with this evaluator set

```
BottomUpEvaluator=DBBottomUp
```

in `OntoConfig.prp`.

Note:
You can run the evaluation methods with all storage systems.

15.1.3 Choosing a Storage System/Evaluation Method/Bottom-Up Evaluator

First you should choose a storage system. This is easy as you just have to think about the required functionality (persistence, renaming capabilities).

15.1.3.1 Persistent Storage Systems

For the persistent storage systems you should always set

```
EvaluationMethod=DynamicFiltering2
BottomUpEvaluator=choose
```

in `OntoConfig.prp`.

15.1.3.2 RAM Storage Systems

The recommended setup for the OntoBroker inference server is

```
Storage=RAM.AVL.Packed
EvaluationMethod=choose
BottomUpEvaluator=choose
```

or

```
Storage=RAM.TS
EvaluationMethod=choose
BottomUpEvaluator=choose
```

15.1.4 Config Optimizer

The config optimizer checks all of your settings (related to evaluation method, bottomup elvaluator) and chooses the very best configuration for a great performance. Read more about the config optimizer in „Server commands for configuration optimizer“ 198.

15.2 Optimizers

OntoBroker supports multiple optimizers (most of them just rewrite a set of rules into an optimized set of rules). However, sometimes optimizers just waste time and space. E.g. if you are not accessing a database then you do not need to use the database optimizers. Since OntoBroker 5.2 it is possible to activate or deactive these optimizers. Find all optimizer settings here [270].

15.2.1 Eliminate Redundant Literals

If literals occur in both the body and head of the rule then they can be removed from the head. E.g. if you have

```
?X:"Person" AND ?X["hasName"->?N] :- ?X:"Person" AND ?X["hasLastName"->?N].
```

then this rewriter eliminates the "X:Person" in the head and replaces the rule by

```
?X["hasName"->?N] :- ?X:"Person" AND ?X["hasLastName"->?N].
```

It is recommended that you leave this optimizer always activated.

15.2.2 Propagate Constraints

This rewriter propagates constraints like "less" or "greater" topdown through the rule graph. A simple example:

```
QUERY q1: ?- p(?X) AND ?X < 5.
RULE r1:  p(?X) :- q(?X).
```

will be rewritten to

```
QUERY q1: ?- p(?X) AND ?X < 5.
RULE r1:  p(?X) :- q(?X) AND ?X < 5.
```

Consider the case when you have 1 million tuples in "q" and only one fact is less than 5. Then this rewriter filters out these tuples in r1 and not later in q1. However, if the constraint does not filter out many results then this optimizer might be counter-productive because the "less" builtin is executed twice.

15.2.3 Rule Unfolding

Rule unfolding means substitution of rule heads by rule bodies in other rules. Lets look for a simple example. Given the following rule and query:

```
p(?X) :- q(?X,?Y) AND r(?Y).
?- p(?Z) AND t(?Z).
```

Then unfolding substitutes the literal p(Z) in the query by the body of the rule and deletes the rule:

```
?- q(?Z,?Y) AND r(?Y) AND t(?Z).
```

This unfolding has some advantages:

- If q and r and t are extensional predicates (there are no rules with q,r,t in the head) then the cost-based dynamic reordering of the rule literals provides better results. As q,r,t are extensional predicates, their extensions are directly known (they are not derived) and thus statistics on these extensions help to reorder the rule body literals in an optimal way.
- Indices must not be created dynamically during query evaluation for intensional predicates but are created once and are retained for all extensional predicates (they are not derived by rules) for future query evaluations.
- If the storage model is persistent, i.e. the H2 relational database of the SQL rewriter creates SQL expressions out of the body literals. This works for extensional predicates only.
- For integration scenarios with external databases rules are also directly rewritten into sql-statements which access the external databases. Because the performance is strongly improved if a lot of external database accesses are combined to a few sql-statements, the use of the StrongUnfolder is very often a very effective option.

In contrast to a weaker version of unfolding, the StrongUnfolder also unfolds body literals for which several rules exist. This is shown in the following example

```
?- q(?X) AND r(?X).
q(?X) :- t(?X).
q(?X) :- s(?X).
```

This set of rules is unfolded to the following set of queries:

```
?- s(?X) AND r(?X).
?- t(?X) AND r(?X).
```

Unfortunately the interplay of different rules together with the StrongUnfolder sometimes leads to an explosion of rules. In these cases generation of the resulting rules and the evaluation of these rules may take a lot of time. It is hardly predictable when this effect occurs. Therefore the StrongUnfolder is switched off by default, but it is strongly recommended to try it especially in integration scenarios.

Simple Unfolder

Given the following rule and query:

```
p(?X) :- q(?X,?Y) AND r(?Y).
?- p(?Z) AND t(?Z).
```

Then unfolding substitutes the literal p(Z) in the query by the body of the rule and deletes the rule:

```
?- q(?Z,?Y) AND r(?Y) AND t(?Z).
```

The switch

```
Rewriter.SimpleUnfolder = on
```

is per default set to "on". In most cases this option should never be set to "off".

15.2.4 Equal Unify Rewriter

This rewriter optimizes the equal and unify builtins. E.g. if you have

```
RULE r1: p(?X) :- q(?X) AND _unify(?X,5).
```

then this rule will be rewritten to

```
RULE r1: p(5) :- q(5).
```

15.2.5 ShrinkDBAccess (Rewriter for External Database Access)

This rewriter processes dbaccessuser literals. It determines the columns that should be selected from a database and adds the list of these columns to dbaccessuser. So only the columns that are needed for the query will be selected from a database and the number of database queries will be reduced.

Example

```
?- _dbaccessuser("projects","F"("id",?VAR1,"project",?VAR2),"conndatasource").
```

will be rewritten to

```
?- _xdbaccessuser("projects","F"("id",?VAR1,"project",?VAR2),"conndatasource",
["ID","PROJECT","CUSTOMER"]).
```

By execution of the first xdbaccessuser all three columns will be selected and cached, so by the second xdbaccessuser no columns should be selected from a database.

15.2.6 FLToSqlRewriter (Rewriter For External Database Access)

The rewriter opimizes a database access. dbaccessuser literals are combined with some builtins and other dbaccessuser literals in one sql query to have restrictions.

Example

```
?- _dbaccessuser("projects","F"("id",?VAR1,"project",?VAR2),"conndatasource") AND
_unify(?VAR1,"myproject").
```

will be rewritten to

```
?- _sqlexecute("select id, project from projects where project
=’myproject’”,
[VAR1,VAR2],..,"condatasource").
```

15.2.7 Rewriter for Propagation of Ground Terms in Rule Graph

The constants in the rule graph can be propagated and thus sometimes edges may be deleted.

Example

```
p("b","X") :- q("X").
t("X","Y") :- p("X","Y").
 ?- t("a","X").
```

If there are only facts for q this query cannot deliver an answer. This may already be seen in the rule graph by propagating the constant a to the second rule and then to the third rule.

15.2.8 Schema Optimizer

The schema optimizer assumes that the whole schema is available. An example: The schema optimizer assumes that an instance

```
JohnDoe:Person[hasName->John]
```

will always have a corresponding schema information for "Person":

```
Person[hasName=>>xsd#string].
```

When you create ontologies with OntoStudio then it is always safe to turn on the schema optimizer. When you manually edit ontologies you have to check if you have the correct schema information in your ontology before activating the schema optimizer.

If you have a rule:

```
p(?X,?V) :- ?X:"Person" AND ?X["hasName"->?V].
```

If the "hasName" only occurs for the concept "Person" then we just change the rule to

```
 p(?X,?V) :- ?X["hasName"->?V].
```

This might improve performance dramatically, especially if you have many instances of "Person".
But it is possible that no schema information exists. In this example using the schema optimizer will produce wrong results if the

```
Person[hasName->xsd#string].
```

is missing. Using the Schema Optimizer means enormous time-savings in an integration scenario.

Note:
Because of performance issues, do not use the schema optimizer and the frame optimizer at the same time. When ontology imports are used then we recommend using the MergeImports OntoConfig option. MergeImports cannot be used in an OntoStudio context.

15.2.9 Eliminate Bodies

This rewriter only affects "dbaccessuser" literals. It will remove a dbaccessuser literal if it is identical (modulo variable renaming) to another dbaccessuser literal.

15.2.10 FrameOptimizer

In ObjectLogic you may have something like:

```
X:Person[hasName->Y].
```

this will split into

```
X:Person and X[hasName->Y].
```

´This has essential disadvantages during the evaluation! The FrameOptimizer merges these facts if possible:

```
att(X,Person,hasName,Y).
```

This means enormous time-savings in an integration scenario.

The following rule types are supported:

- dbaccess rules as they are generated by the OS db import plugin
- Mapping rules as they are generated by the OS mapping tool
- Rules where each att-literal is accompanied by an appropriate isa literal in the head and body
- Classification rules like ?X:D :- ?X:E and ... where D is a sub concept of E

Restrictions:

- The model must not contain any instances.
- The model must be compatible with CNG=on, ANG=on, MNG=on.
- Fillnull for the query is not supported.
- Queries for identifiers like ?- ?X:C. are not supported, only queries for attribute values.
- the rules involved in a query must be of the type above.

Example

```
… :- ?X:Person[hasAge->?Y] …
```

optimized:

```
… :- att(Person,hasAge,Y) …
```

Note:
Because of performance issues do not use the schema optimizer and the frame optimizer at the same time.

15.2.11 Rewriter for Transitivity Rules

The transitivity property of relations creates rules like:

```
?X[prop->?Y] :- ?X:domain, ?Z:domain, ?X[prop->?Z], ?Z[prop->?Y].
```

It is much more efficient to have tail- or head-recursion instead of double recursion.

The rewriter creates rules like:

```
?X[prop->?Y] :- ?X:domain, ?Z:domain, ?X[directprop->?Z], ?Z[prop->?Y].
```

In addition if ?X is bound the following rule is more efficient:

```
?X[prop->?Y] :- ?X:domain, ?Z:domain, ?X[prop->?Z], ?Z[directprop->?Y].
```

otherwise (if ?Y is bound) the first rule is better.

15.2.12 MergeImports

If the MergeImports parameter is set to "on" in the OntoConfig.prp, imported ontologies are directly merged into the importing root ontology. This means that the imported ontologies are not available as separate ontologies as in OntoStudio.

Example

To activate the MergeImports switch, in the OntoConfig.prp set

```
MergeImports = on
```

Assume that you have three ontologies A,B, and C. A imports B and B imports C. Then, after loading you have only one ontology A' (consisting of rules and facts from A,B, and C).

Note 1:
The flag influences only the loading process on OntoBroker start (or reload command).

Note 2:
The MergeImports flag cannot be used together with project files.

15.3 Optimizing the Sequence of Body Literals in Rules

Uses sophisticated heuristics for optimizing the sequence of join operations, but sometimes the best heuristic is not able to find out the optimal join sequence for a specific query with specific rules. An example should illustrate this. Consider the following sequence of body literals in the rule:

```
QUERY q1: ?- ?P["inTournament"->?WM].
RULE r1: "P"["inTournament"->"WM"].
<-
       P:Player.           // 1000 tuples
       WM:Tournament       AND    // 25 tuples
       WM[match -> M] AND  // 30 tuples
       M[team -> T] AND    // 2 tuples
       T[lineup -> P] AND  // 11 tuples
```

However, this sequence is not the optimal join sequence as many tuples that are created e.g. by the "P: Player" literal are later discarded in join operations. So the following sequence strongly reduces this effect:

```
QUERY q1: ?- ?P["inTournament"->?WM].
RULE r1: "P"["inTournament"->"WM"].
```

```
<-
      WM:Tournament        AND    // 25 tuples
      WM[match -> M] AND   // 30 tuples
      M[team -> T] AND     // 2 tuples
      T[lineup -> P] AND   // 11 tuples
      P:Player.            // 1000 tuples
```

This is exactly the case where features like the cost model or the genetic optimizer come into play. Both features use different strategies to optimize the join sequence, but both are able to improve performance significantly.

15.3.1 Using the Genetic Optimizer

The genetic optimizer allows specific queries to be optimized. This optimizer is typically used before deploying ontologies on a production server. The order of the body literals can be changed/permuted and this might improve the inference speed dramatically. If you create all possible permutations, you may find the perfect result but this would take a very very long time. For 10 rules with 4 body literals you would have 104! possible solutions, for 100 rules with 8 body literals you would have 1008! possible solutions. The idea is that you do not need the best result but a result that is good enough. If you get one change that improves the query, it should be even better when combined with other positive changes. For optimizations in such high dimensional search spaces genetic algorithms are applicable. Thus the genetic optimizer optimizes the sequence of body literals in the rules using a genetic approach. In the initialization process a random population will be created. This population is a defined number of individuals with each representing exactly one permutation. These individuals together are called a generation, as it is the first one this one, is generation 0. Then the fitness of all individuals will be determined. For this, the permutations will be applied and the query will be executed. The smaller the evaluation time is, the better the fitness. The better the fitness of an individual is the better is its chance to contribute to the following creation of the individuals of the next generation. Their characteristics, the ordering of body literals, will be mixed and a new population will be created. New random changes may be made. The fittest individual survives unchanged. This continues until the wanted number of generations has been calculated.

Limits

As the continuum of all possible reordering is very big, you will need many individuals to test. For complex rule scenarios several thousand individuals should be created, so that it takes a lot of time to optimize a long-running query. Another problem may be the memory demand of the query evaluation. Some permutations will need much more memory than the average. If you already operate on the limits OntoBroker can handle with the available main memory, you might encounter out of memory" errors when using the genetic optimizer.

Note:
If you optimize a specific query with the genetic optimizer and you change the rules or the facts then you will need to execute the genetic optimizer again. If you only change the constants or the order of literals in a rule you do not need to optimize the query again.

Usage

```
opt_genetic <queryID>
```

Commands to send an optimized query to OntoBroker

Command	Description
opt_genetic status	The return value is a string. The status of the last complete generation will be returned. At least one generation has to be evaluated before valid results can be delivered. OntoBroker Return Values: The return value is a "string" without line separators. Use | as separation character.
opt_genetic kill	Stops a running optimization. It may take a short time, you can use the -isoptimizing command to see when it is done. The timeout window has to fit into the timeout of the socket connection, so it may be to small for some cases.

opt_genetic isoptimizing	Returns if an optimization process is currently running or not.
	OntoBroker Return Values: The return value is a "string".
	Example:
	`opt_genetic file, "test.data", rewrite, timeout, 1000, module,"<http://www.NewOnto1.org/ontology>", "queryID"`
	`command.cmd "opt_genetic file, \"test.data\", rewrite, module, <http://www.ontoprise.de#testonto>, \"queryID\"`

Option	Description
file filename	file name for optimization data
rewrite	rewrites the file with optimization data if it exists.
timeout value	optimization timeout (ms)
kill	stops the optimization
isoptimizing	checks whether the optimization is already running
status	returns the best result of the running optimization

15.3.2 Hillclimbing Optimizer

The command opt_hillclimbing <queryID> optimizes the bodyordering using a hillclimbing algorithm. You can change the ordering manually.

Usage

```
opt_hillclimbing [file, filename], [rewrite], module, modulename, queryID
opt_hillclimbing isoptimizing
opt_hillclimbing kill
opt_hillclimbing status
```

Example:

```
opt_hillclimbing file, "test.data", rewrite, module, "<http://www.NewOnto1.org/
ontology>", "queryID"
```

```
command.cmd "opt_hillclimbing file, \"test.data\", rewrite, module, <http://www.
ontoprise.de#testonto>, \"queryID"s\"
```

Option	Description
`file filename`	file name for optimization data
`rewrite`	rewrites the file with optimization data if it exists.
`kill`	stops the optimization
`isoptimizing`	checks whether the optimization is already running
`status`	returns the best result of the running optimization

```
for all rules r
       := max
       fomintime r all body permuations  p of r
              evaluate query
              if evaltime < mintime
                     mintime = evaltime
                     minpermutation = p
       set body permutation of r to minpermutation
```

It is displayed as plain text in the file bodyordering.data. You can rename the file in OntoConfig.prp:

```
BodyOrderingDataFile = mybodyordering.data
```

Additonal information about the hillclimbing algorithm:

Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

Limitations

Rule bodies should not have many literals! If you have e.g. 10 rule bodies you have to deal with 10 permutations. In this case it would be better to deactivate the unfolding.

15.3.3 BodyOrdering

BodyOrdering is the sequence of F-atoms executed in a rule or query. E.g. when you have

```
?- ?X:Person AND ?X[hasName->?Y] AND ?X[worksAt->?Z].
```

then possible body orderings are

```
?- ?X:Person AND ?X[hasName->?Y] AND ?X[worksAt->?Z].
?- ?X[hasName->?Y] AND ?X:Person AND ?X[worksAt->?Z].
?- ?X[hasName->?Y] AND ?X[worksAt->?Z] AND ?X:Person.
...
```

When choosing "BodyOrdering = auto" then OntoBroker will try to optimize the sequence of F-atoms in the query. When choosing "BodyOrdering = off" then the sequence of F-atoms will stay it it is:

```
?- ?X:Person AND ?X[hasName->?Y] AND ?X[worksAt->?Z].
```

Note:

This might lead to situations where the whole program is not evaluable anymore or where the query returns wrong results.

15.4 Tracing the Evaluation Process

OntoBroker has some useful features which help to analyze performance and to trace the evaluation process.

```
################################################################
# Trace (only has Effect when using 'trace' Query Option)
# NOTE: For Statistics about Query Execution
#       use the 'profile' Query Option.
################################################################
#Trace.Evaluation = off
#Trace.Evaluation = simple
#Trace.Evaluation = rules
#Trace.Evaluation = full
#Trace.Rewriter   = on
#Trace.Rules      = off
#Trace.Queries    = on
#Trace.Datamodel.Changes    = on
#Trace.Datamodel.Requests   = on
#Trace.Materialization = on
```

This is an expert feature which gives extensive log information:

- Display of selected rules
- Behavior of individual rewriters
- Flow of tuples through the operatornet
- Shows the logged requests on datamodel (via Java-API)

The following OntoConfig options allow external database access to be traced:

```
#Trace.ExternalDatabase     = off
#Trace.ExternalDatabase.SQL = off
```

When these options are set then OntoBroker will show

- Information about the database schema
- Information about SQL database requests (SQL query, number of results)

Example

```
@{q1, options[trace]} ?- ?X::Company.
```

The rules listed in the output are all rules which are relevant for the evaluation (this is in most cases a subset of the total rules). During the evaluation statistics are gathered which help give an impression of the evaluation characteristics of rules. Often the cause for the poor evaluation time of a rule lies in some built-in or connector (especially database connectors can take up a lot of time).

If you are interested in how OntoBroker actually evaluates the rules, you can turn on full tracing "Trace. Evaluation=full". Then you will see the flow of tuples through the operatornet. Note that the output is very hard to read and interpret. The primary use case of this feature is that you can send the OntoBroker output for a specific query to the ontoprise support team, to, for example, help reproduce a bug or explain an unexpected result.

15.5 Use Materialization of Rules and/or Axioms

A good way to optimize the query performance is to materialize the rules and/or the standard axioms. Materialization simply means that rules are executed and that the results are stored as facts. For subsequent queries the materialized rule will no longer be considered.

16 Rule Management

If you want an easy way to

- Find rules by module and rule identifier
- Retrieve rule by rule identifier
- Search for rules containing a ObjectLogic term (concept/instance) in the head or body
- Search for rules containing some text
- Add and remove rules

then you can use the rule management features of OntoBroker. Explanations and explanation rules are described in the *Explanations* chapter.

Find rules & retrieve rule text

Use the built-ins _ruleByID/3 and _queryByID/3 to retrieve the rule text for a given rule ID and module. If you only need the rule IDs of a given module, you can use the built-ins _isRule/2 and _isQuery/2.

Example

"Give me the rule text of all rules in all modules"

```
?- $module(?M) AND _ruleByID(?M,?ID,?TXT).
```

Search for rules containing a ObjectLogic term

The built-ins _ruleContainsTerm/4, _queryContainsTerm/3 can be used to retrieve all rules and queries respectively containing an F-logic term in rule head or body.

Example

"Give me the rule IDs of all rules in module 'm1' containing the concept 'Person' in the rule body"

```
?- _ruleContainsTerm("m1","Person","_false",?ID).
```

Full-text search support for rules

The rule text and the rule metadata are full-text indexed. The built-in queryIndex/10 can be used to search text in rules. Add "+type:u" to the lucene query text to restrict the query to rules or "+type:q" to restrict to queries. The SearchHelper class of the OntoBroker API also supports this functionality.

Commands for adding and removing rules

Commands for ObjectLogic:

```
insert rule { ruleText ...}
delete rule { ruleText ... }
delete rule ruleID1,ruleID2,...
```

Example:

"Add facts q(1,a) and q(2,b) to module m1, and exchange rule r1 with following rule: @{r1} p(?X) :- q(?X,?)."

```
:- module m1.
insert {q(1,a),q(2,b)}
delete rule r1.
insert rule {@{r1} p(?X) :- q(?X,?).}
```

17 Explanations

A feature of OntoBroker is to explain the reason why a query returns specific results. This chapter shows how to enable the explanation feature in OntoBroker and give an example of how to formulate so-called explanation rules.

Basics

Retrieving explanations is done in two steps

- First execute the query with the "explain" query option
- Then execute another query to retrieve the explanations for the first query

If an explanation for a generated answer is requested, it is necessary that each relevant rule of the ontology is accompanied by an explanation rule. These rules contain text blocks to explain the meaning in common language, including place holders for the concrete instantiations of the rule variables.

Example

We have a ObjectLogic ontology describing people and the company they work in:

```
// Concept hierarchy -----------------------------------
Company[].
Employee[].

// Schema ----------------------------------------------
Employee[colleague {0:*} *=> Employee].
Employee[company {0:*} *=> Company].

// Instances -------------------------------------------
christian:Employee.
eva:Employee.
ibm:Company.
ontoprise:Company.
patrick:Employee.
roman:Employee.
saartje:Employee.

// Properties ------------------------------------------
christian[company->ibm].
eva[company->ontoprise].
patrick[company->ibm].
roman[company->ontoprise].
saartje[company->ontoprise].
```

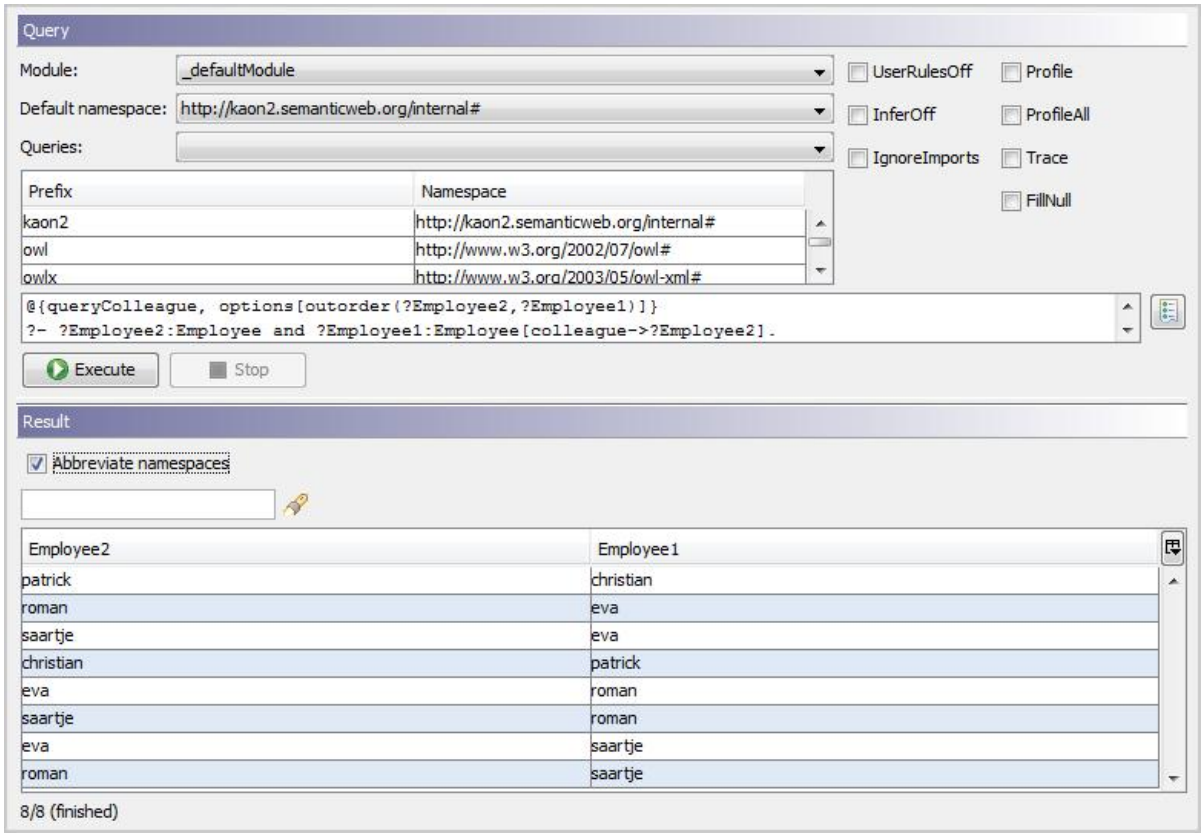
Additionally we have a rule that states that if two employees work at the same company they are colleagues:

```
@{ruleColleague}
?A[colleague->?D] :- ?A:Employee AND ?A[company->?F] AND ?D:Employee AND ?D[company-
>?F] AND ?F:Company AND NOT _unify(?A,?D).
```

You can pose the following query to ask for people and their colleagues:

```
@{queryColleague, options[outorder(?Employee2,?Employee1)]}
?- ?Employee2:Employee and ?Employee1:Employee[colleague->?Employee2].
```

This will return the following results:



Now you might want to explain why two people (represented by the variables denoted by A and D) are colleagues.

```
$A$ and $D$ are colleagues because they are both employees of the company $F$
```

This would be expressed in the ObjectLogic rule

```
 @{ruleColleague_EXPLANATION}
 _explain(?QK,?I,?S) :-
   (?I:<obl:default:Instantiation>[<obl:default:ruleid> -> ruleColleague],
    ?I[<obl:default:variables> -> <obl:default:i>(<obl:default:A>,?var_A)],
    ?I[<obl:default:variables> -> <obl:default:i>(<obl:default:D>,?var_D)],
    ?I[<obl:default:variables> -> <obl:default:i>(<obl:default:F>,?var_F)],
    _applyStringTemplate("$A$ and $D$ are colleagues because they are both employees
of the company $F$",[], [<obl:default:A>(?var_A),<obl:default:D>(?var_D),<obl:
default:F>(?var_F)],?S))@$prooftreefacts(?QK).
```

Note that OntoBroker allows explanation rules to be generated automatically, so you do not have to write this rule manually (see below for details). Querying for all employees and their colleagues would be done in the following way:

```
 ?- ?Employee2:Employee and ?Employee1:Employee[colleague->?Employee2].
```

If result explanations are required, this is done using an additional annotation:

```
 @{options[explain]} ?- ?Employee2:Employee and ?Employee1:Employee[colleague->?
Employee2].
```

This will bring the same results as before but will internally create a new module for the query session. The module contains the information needed to create the explanations. They can be queried in the following way:

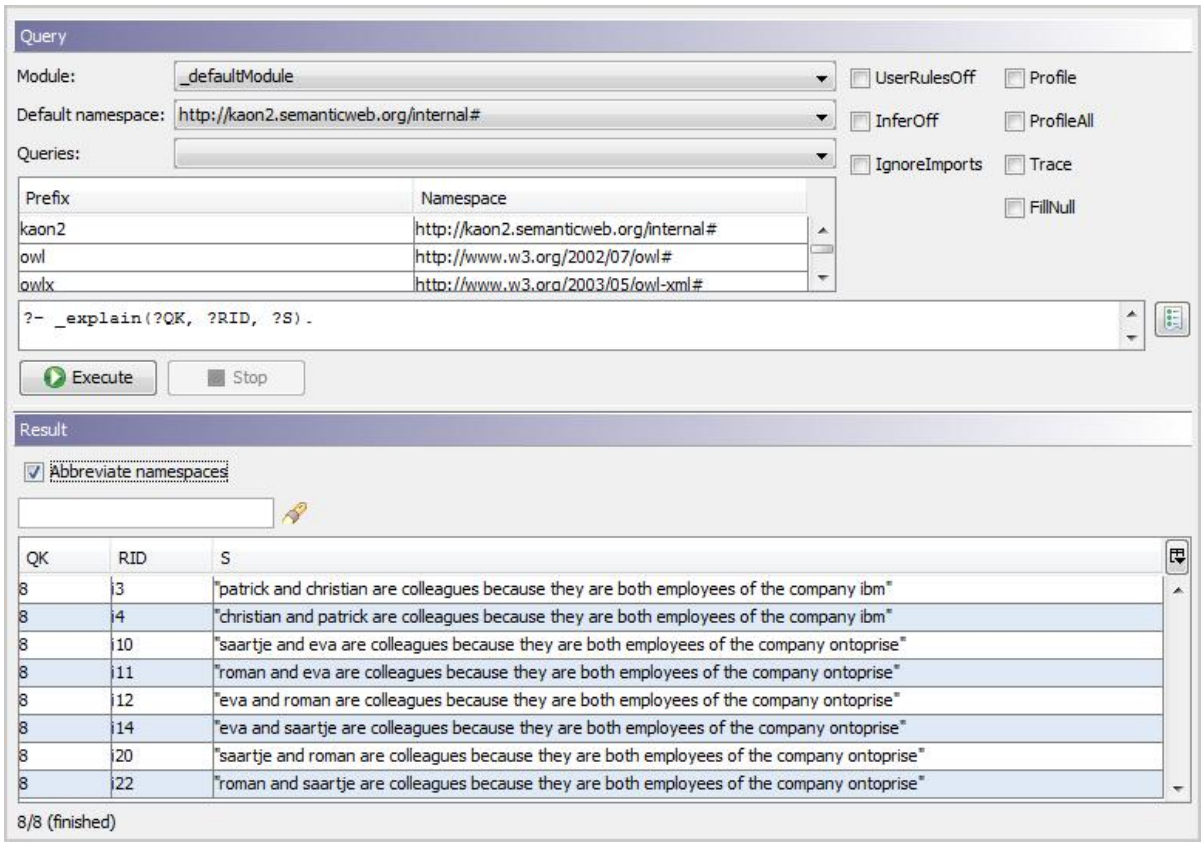
```
 ?- _explain(?QK, ?RID, ?S).
```

Where:

- ?QK: is a variable representing the query session
- ?RID: is the result id

- ?S: is the textual representation of the explanation

This will actually bring the following result for this ontology:



Generating Explanation Rules

OntoBroker provides a built-in for automatically generating explanation templates. Assume you want to generate an explanation rule for the following rule:

```
@{R1}
?-
aPerson1[hatKind->aPerson3]
:-
aPerson1:Person AND
aPerson2:"Person AND
aPerson3:"Person AND
aPerson1[hasChildWith(aPerson2)->aPerson3].
```

First of all you can use the built-in _generateExplanationTemplate/4 to generate the default explanation template:

```
?- _generateExplanationTemplate("m1","R1","",?OUT).
```

Generated template

```
'$aPerson1$ hatKind $aPerson3$
BECAUSE
$aPerson1$ is a Person AND
$aPerson2$ is a Person AND
$aPerson3$ is a Person AND
$aPerson1$ ns_(#,hasChildWith($aPerson2$)) $aPerson3$'
```

Typically you will edit this template to abstract from the rule details and focus on the statement of the rule. With the final template you can then use the built-in _generateExplanationRuleText/4 to generate the rule text of the explanation rule:

```
?- _generateExplanationRuleText("m1","r1","""$aPerson1$ hatKind $aPerson3$
BECAUSE
$aPerson1$ is a Person AND
```

```
$aPerson2$ is a Person AND
$aPerson3$ is a Person AND
$aPerson1$ ns_(#,hasChildWith($aPerson2$)) $aPerson3$""",?OUT).
```

Generated explanation rule

```
explain_(QK,I,S) <-
    I:Instantiation[ruleid->R1]@prooftreefacts_(QK)
    AND I[variables->i(aPerson1,var_aPerson1)]@prooftreefacts_(QK)
    AND I[variables->i(aPerson3,var_aPerson3)]@prooftreefacts_(QK)
    AND I[variables->i(aPerson2,var_aPerson2)]@prooftreefacts_(QK)
    AND _applyStringTemplate(\'$aPerson1$ hatKind $aPerson3$
BECAUSE
$aPerson1$ is a Person AND
$aPerson2$ is a Person AND
$aPerson3$ is a Person AND
$aPerson1$ ns_(#,hasChildWith($aPerson2$))
       $aPerson3$\',[],[aPerson1(var_aPerson1),aPerson3(var_aPerson3),aPerson2
(var_aPerson
       2)],S)@prooftreefacts_(QK).'
```

Note:

- **Querykey**: As the OntoBroker runs as a server there may be several queries at a time. To be able to identify the requested explanation, each query is uniquely identified by a key. This key allows the explanation for the query to be accessed. The querykey is accessible via the API of the QueryClient (please see the JavaDocs for further informations).
- **Caching**: When the explanation feature is used, each executed query generates explanation information stored at the server. To prevent the system running out of memory it is necessary to remove the generated explanation using the removeExplanation command. This command takes the querykey as an argument and will tell the server to clean up all information related to the explanation of this concrete query.
- **Explanations**: OntoStudio provides an graphical user interface for dealing with explanations.

18 File Encodings and Unicode

OntoBroker supports the loading of files of different encodings, e.g. UTF-8 or language (and platform) specific encodings like <CP1251> for Cyrillic Windows systems. The following rules apply:

- OntoBroker interprets all ObjectLogic files using the system standard encoding. XML-based formats (RDF (S), OWL) are interpreted according to the file-encoding specified in the XML file itself.
- Upon loading, OntoBroker can be forced to use a special encoding for a file by using the -fenc parameter, e.g.

```
start-ontobroker -fenc cp1251 russianCars.obl
```

- When serializing F-logic files, the standard system encoding or (when specified) the selected encoding is chosen. XML-based formats will always be serialized in UTF-8.

The OntoBroker Java API provides several save/export methods for storing ontologies. These methods have parameters which allow the output encoding to be specified.

19 OntoBroker Server

19.1 Launching OntoBroker

OntoBroker32.exe and OntoBroker64.exe start with a 32bit and 64 bit Java VM respectively. In contrast to OntoBroker 5.3, these launchers do not launch a separate Java process and can therefore be used for launching OntoBroker as a service (this requires third party tools).

Starting with OntoBroker 6.0, OntoBroker is embedded in Equinox, an OSGi framework. To keep the configuration of OntoBroker simple, the launching involves multiple steps. Before the Equinox framework itself is started and the OSGi bundles are activated, several system properties are set either from parameters of the OntoConfig.prp, or they are calculated. In detail the following takes place at this stage:

Parameters from OntoConfig.prp are translated to System properties for the Equinox framework

- HttpPort => org.eclipse.equinox.http.jetty.http.port
- Extensions.Directory => felix.fileinstall.dir
- log4j.configuration => log4j.configuration
- WorkerName => com.ontoprise.ontobroker.jetty.customizer.workername
- AJP => com.ontoprise.ontobroker.jetty.customizer.ajp
- AJP.Port => com.ontoprise.ontobroker.jetty.customizer.ajp.port
- launcher.* => *
  (this means all config parameters in OntoConfig starting with the prefix "launcher." are copied to the system properties after removing the prefix)

Determining the OntoBroker home directory:

1. If the environment variable ONTOBROKER_HOME is set, it is used.
2. If the System property ontobroker.home is set, it is used.
3. If the System property java.class.path contains the com.ontoprise.ontobroker.launcher jar file, OntoBroker home is reconstructed from its path.
4. As a last fall-back the current directory is tried.

The Ontobroker home directory is also added as "ontobroker.home" to the system properties.

Determining the log4j configuration:

1. If the OntoConfig.prp contains the parameter "log4j.configuration", its value is used.
2. If there is a "log4j.properties" in the working directory, it is used.
3. If there is a "log4j.properties" in the ONTOBROKER_HOME directory, it is used.

Determining the extensions directory:

1. If the OntoConfig.prp contains the parameter "Extensions.Directory", its value is used.
2. If there is a "extensions" subdirectory in the working directory, it is used.
3. If there is a "extensions" subdirectory in the ONTOBROKER_HOME directory, it is used.

Additionally, the following default system properties are added if they are not defined as "launcher.*" parameter in the OntoConfig.prp:

- osgi.requiredJavaVersion=1.6
- osgi.noShutdown=true
- eclipse.ignoreApp=true
- osgi.install.area=$ONTOBROKER_HOME/plugins

- osgi.configuration.area=$ONTOBROKER_HOME/conf
- felix.fileinstall.poll=3000
- org.eclipse.equinox.http.jetty.customizer.class=com.ontoprise.ontobroker.jetty.customizer.OntoBrokerJettyCustomizer

Hint for OSGi-related problems:

If the collaboration server could not be started (the message `INFO [COLLAB_SRV] Collaboration Server initialized` is not displayed) it often helps to delete the conf/org.eclipse.osgi folder of your OntoBroker installation. This folder contains cached configuration data.

19.1.1 New Configuration Parameters

All switches can be set in the configuration file `OntoConfig.prp` which resides in the starting directory of the server (usually the installation base directory of the server).

New Configuration Parameter	Description	Values	Replacement
HttpPort	Port for Web server used for the collaboration server, remote API, web console and web services.	Default: 8287 HttpPort = 8287	This replaces the parameter WsHttpPort and the command line option -wsport.
Extensions.Directory	Directory for dropping external builtins, rewriter, webservices and dynamic service JAR files.	Default: extensions Extensions.Directory = c:/myextensions	This replaced the obsolete parameters PathToBuiltinDir and PathToRewriterDir.
CollaborationServer	If set to on, the collaboration server is started.	Default: off CollaborationServer = on	This is an alternative to the command line option -collabserver.
RemoteAPI	If set to on, the remote API is available. Remote API and collaboration server cannot be set to started at the same time.	Default: off RemoteAPI = on	This is an alternative to the command line option -remoteAPI.
WebServices	If set to true, the Web services are deployed and available on the HTTP port under the path /services.	Default: off WebServices = on	This is an alternative to the command line option -webservice.
WebConsole	If set to true, the web console is available on the HTTP port under the path /console/console.html.	Default: off WebConsole = on	This is an alternative to the command line option -webconsole.
log4j.configuration	The path to the log4j.properties.	Default: log4j.properties log4j.configuration = c:/mylog4j.properties	
Configuration.Directory	The path to the configuration directory.	Default: conf Configuration.Directory = conf	This replaces the old configuration parameter dir.conf.
AJP	Enables the AJP protocol (used for load balancing). Note that the AJP protocol cannot be used if HTTPS is enabled.	AJP = on	
AJP.Port	Port used for the AJP protocol (defaults to 8009). This parameter is only active if the AJP protocol is enabled.	AJP.Port = 12345	
WorkerName	Worker name used for identifying an OntoBroker instance in a load balancing cluster.	WorkerName = ob3	

19.2 Use the OntoBroker Server from commandline

OntoBroker can be started from commandline by either using the script start-ontobroker.cmd (on Windows) or start-ontobroker.sh (on Linux). On Windows platforms you may also use OntoBroker32/64.exe. This executable does the same as the start-ontobroker.cmd: it automatically reads all JAR files within the lib directory and executes the inference server. Using OntoBroker.exe makes it easy to integrate OntoBroker in your own scripts.

The following command is used to launch the server:

```
start-ontobroker.cmd [options] [filenames]
start-ontobroker.sh [options] [filenames]
```

-configFile <configFile>	**The configuration file to be used (default: OntoConfig.prp).**
-project <projectFile>	Use project file for advanced load configuration.
-webservice	Enables OntoBroker web service. Alternatively use the parameter WebServices = on
-ajp	Enables OntoBroker web service for load balancing. Alternatively use the parameter AJP = on
-webconsole	Enables OntoBroker web console. Alternatively use the parameter WebConsole = on
-collabserver	Enables OntoBroker collaboration server (implicitly sets Indexer = on); alternatively use the parameter CollaborationServer = on
-collabserverNoIndexer	Enables OntoBroker collaboration server (leaves Indexer settings unchanged)
-remoteAPI	This has nearly the same effect as the option -collabserver. The OntoBroker is tuned for query performance and some special functionalities needed for OntoStudio are not switched on. This means that an OntoStudio cannot use this instance as a collaboration server. If OntoBroker is started with -remoteAPI it behaves as a stand alone server, i.e. the inferencing mode is turned off. Alternatively use the parameter RemoteAPI = on
-fenc <encoding>	File encoding used for reading files. (e.g. chinese codes are "big5", "csBig5" and "iso_ir-58"). This is necessary when the file encoding of a concrete file differs from the default of your operating system.
-m	Materialize the following file or directory
-m:update	Materialize the following file or directory (updatable)
-commitDuringImport	Executed multiple commits while loading a file
-silent	Turns off logging
-rdf, -rdfs, -obl, -owl, -nt, -nt3 <directory or filename>	Explicitly specifies the data format for reading
-help, -h	Prints usage text and exits
-diagnostics	Writes information to the log which could help to analyze bugs.

Note:
If a parameter has been specified both in the OntoConfig and on the command line, the command line option overrules the OntoConfig settings.

Example:

```
start-ontobroker.cmd -webservice -collabserver -configFile OntoConfig.prp -m -flo
examples/flo/Unternehmen_US.obl -obl example/flo/Process_US.obl.
```

If the environment variable ONTOBROKER_HOME is set, it is possible to run OntoBroker from any directory. As OntoBroker looks in the working directory for OntoConfig.prp, log4j.properties and for various files in the conf subdirectory by default, you have to change these defaults explicitly or copy these files in your working directory.

- Changing location of OntoConfig.prp: use the command line option -configFile <configFile>
- Changing location of log4j.properties: set the Java system environment -Dlog4j.configuration=file:<filename>
- Changing location of the conf directory: Explicitly set the dir.conf parameter in the OntoConfig.prp

A query is sent to the server using the command line client:

```
query.cmd [-h <hostname>] [-p <portno>] [-l <user>:<pwd>] <query>
query.sh [-h <hostname>] [-p <portno>] [-l <user>:<pwd>] <query>
```

A command is sent to the server using the command line client:

```
command.cmd [-h <hostname>] [-p <portno>] [-l <user>:<pwd>] <command>
command.sh [-h <hostname>] [-p <portno>] [-l <user>:<pwd>] <command>
```

The optional arguments <hostname> and <portno> allow accessing the server on a different host on the specified port. If no <hostname> is given the local host is used. If no port number is given the standard port 2267 is used. When login is required credentials can be given with the optional arguments <user> and <pwd>.

19.3 Server Commands

You can use the "command.sh" or "command.cmd" script to execute OntoBroker server commands.

Command Client (command.cmd/command.sh)

The command client provides an additional option for specifying a command file:

```
command.cmd -cf myCommandFile.txt
```

The file is loaded and the command is executed.

Writing Your Own Commands

External commands can now be added as extension plugins to OntoBroker. Java classes implementing the SimpleAPICommand interface are packaged in a plugin exporting a CommandProvider service and can be used with OntoBroker by deploying the jar file to the extensions directory.

Command class

An external command is implemented as a Java class implementing the interface `com.ontoprise.api.command.SimpleAPICommand`.

```
public interface SimpleAPICommand {
    String getCommandName();
    Object execute(OntologyManager ontologyManager, Term[] args) throws
KAON2Exception, InterruptedException;
}
```

The comma separated argument list is provided as term array. Typically, a command returns a String, but it can theoretically return any kind of Java object. However, there some restrictions to consider:

- If using the method OntologyManager.execute from a collab server client, the returned class must be known to the Collabserver marshaller. This effectively means that only basic Java data types and collections and classes generated by the KAON2Factory can be used in this case.
- If the command is executed by the socket protocol or OntoBroker web service the result is always translated into a string by calling toString().

CommandProvider

In additionally to the command class itself you need a CommandProvider service, which announces the command(s) to OntoBroker. Example:

```
public class CommandProviderImpl implements CommandProvider {
    @Override
    public Collection<Class<? extends APICommand>> getCommands() {
        ArrayList<Class<? extends APICommand>> list = new ArrayList<Class<? extends
APICommand>>();
        list.add(MyCommand.class);
        return list;
    }
}
```

This provider must be registered as an OSGi service, e.g. defining a Declarative service. Add to the OSGI-INF directory of your plugin a file commands-provider.xml:

```
<?xml version="1.0"?>
<component name="com.ontoprise.ontobroker.examples.api.commands">
   <implementation class="com.yourcompany.x.y.CommandProviderImpl" />
   <service>
      <provide interface="com.ontoprise.api.command.CommandProvider"/>
   </service>
</component>
```

The package of the command class must be exported from META-INF/manifest.mf of the plug-in and the service component must also be added to the manifest, e.g.

```
Service-Component: OSGI-INF/commands-provider.xml
```

-runscript command line option

OntoBroker now supports a -runscript command line option for executing a command or command list from a file during launching.

Note:
This option is not supported for OWL.

Syntax

```
-runscript <filename>
```

Example

This example shows how to insert some facts, export the module, and exit OntoBroker automatically.

```
ontobroker32.exe -runscript sample.oblcmd
```

`sample.oblcmd`

```
:- default prefix = "http://yourcompany.com#".
:- module m1.

// try block to ensure execution of final shutdown command
try {
  // first insert some facts
  insert {
    Person[name {0:1} *=> _string, age {0:1} *=> _int].
    p1:Person[name->"Susan",age->11].
    p2:Person[name->"Mike",age->12].
  }
  // save Ontology <http://yourcompany.com#m1> to the file m1.obl
  save m1 to "m1.obl".
}
// shutdown OntoBroker
shutdown
```

19.3.1 Basic Commands

Command	Function / Explanation
analyzeStorage	This command can be used to update the statistics of the OntoBroker storage. Recommended to execute after large updates when H2 storage is used. Example: `_ontobrokerManager.executeCommand("analyzeStorage")` Result: "Storage analyzed in 42 ms.W
busy	Returns the server status. syntax: busy OntoBroker Return Values: true, false client-side output: "[INFO]: Server busy state: [true|false]"
clear	All stored answers in the cache are removed syntax: clear OntoBroker Return Values: cache cleared
commands	Returns a list of all available commands in the server syntax: commands OntoBroker Return Values: A list with all server commands

delete	The command del has been removed since Ontobroker 6.0. Instead, please use the delete command which has a richer syntax.
	Note:
	It is allowed to specify query options (since OB 6.0 Build 382).
	Here is an example of how to use delete instead of del.
	Old syntax in OntoBroker 5.3
	`del otto[age->18]. // syntax from OB 5.3, invalid in OntoBroker 6.3`
	In OntoBroker 6.1 the corresponding command is
	`delete otto[name->"Otto"].`
	The delete command is used to remove facts and rules from an ontology. Its syntax is very similar to the insert syntax.
	Example with single fact:
	`:- module <http://yourcompany.com/ontology1>.`
	`delete Person[name *=> _string].`
	Example with multiple facts and rules:
	`:- module <http://yourcompany.com/ontology1>.`
	`delete {`
	`  Person[name *=> _string, age {0:1} *=> _integer].`
	`  otto:Person[name -> "Otto Waalkers"].`
	`  fritz[name -> "Fritz", age->9].`
	`  @{rule1} ?X:Child :- ?X:Person, ?X.age < 18 .`
	`}`
	Example with where clause:
	`delete {p(?X,?Y),q(?X),q(?Y).} where {@{options[inferOff]} ?- p(?Y,?X).}`

delete clauses	This command can be used to delete clauses (rules, queries, constraints) by their label. For example to delete the rule with the id r1 and the query with the id q1: `delete clauses r1,q1`
dump	Writes a full dump of the extensional and the intensional database (internal representation).

<table>
<tr><td></td><td>The target folder is the so called dump folder where OntoBroker is installed.</td></tr>
<tr><td>exportAllModules</td><td>Writes a dump of all modules to the files system. The target folder is the folder where OntoBroker is installed.
The command "exportAllModules" supports additional options:
• encrypt - modules are saved encrypted
• sort - facts of modules are saved in sorted order

Note:
If, in the command line, you specify a directory to load the ontologies from like "ontobroker -obl decrypt:v1:dir", the decrypt handler first of all checks if the file is encrypted. Example: ontobroker -obl decrypt:v1:C:\ontologies.

Example:
Save modules <http://yourcompany.com#ontology1> to file c:\data\onto1.obl with sorted option
<pre>save <http://yourcompany.com#ontology1> to "c:\\data\\onto1.obl",sort.</pre>Save modules <http://yourcompany.com#ontology1> to file c:\data\onto1.obl with sort and encrypt option
<pre>save <http://yourcompany.com#ontology1> to "c:\\data\\onto1.obl",sort,encrypt.</pre>Export all modules sorted
<pre>exportAllModules sort.</pre>Export all modules sorted and encrypted
<pre>exportAllModules sort,encrypt.</pre>
Note:
If "SerializerEscapeNewLines = on" is set in the OntoConfig.prp, new line characters in string constants are escaped on serialization (i.e. on commands like "exportAllModules" and "save").

This switch can be set in OntoConfig.prp via:
<pre><SerializerEscapeNewLines> = <on|off></pre></td></tr>
<tr><td>generateDynamicSvc</td><td>There are now two commands for dynamic web services:
<pre>generateDynamicSvc "path-to-specification-module.obl".</pre>The string argument must be a valid URL or path to the specification module file.
and
<pre>deleteDynamicSvc <module></pre>Here <module> is the module term of the specification module.

Note:
OntoBroker must have been started with the "-webservice" option. The command is only available for Objectlogic.

Example:
<pre>:- module echo.

:- prefix in = "http://schema.ontoprise.com/dynamic-svc/in#".
:- prefix out = "http://schema.ontoprise.com/dynamic-svc/out#".
:- prefix meta = "http://schema.ontoprise.com/dynamic-svc/meta#".

// Spec for Input
in#Parameters[in#message {1:1} *=> _string].

// Spec for Output</pre></td></tr>
</table>

	`out#Results[out#echo {0:1} *=> _string].`
	
	`// Generation of Output`
	`theResults:out#Results.`
	`theResults[out#echo -> ?X] :- ?P:in#Parameters[in#message->?X].`

`h2 backup <output-name>`	The single argument specifies the target filename of the database copy. This backups the database file to "d:\\backup\\mybackup.data.db" **CAUTION:** **All files like "d:\\backup\\mybackup.*" are deleted during the copy action to prevent the possible mixing of old and new temporary database files.** Restoring a backup requires the following steps: 1. Shutdown OntoBroker 2. Delete all H2 database files, e.g. if the database is named "edb", you have to delete "edb.*". 3. Copy the backup file to the old location and rename it to the original name. E.g if the database is named "edb", then the database file must be named "edb.data.db". 4. Start OntoBroker. On opening the database file, the indices will be reconstructed. This procedure may take some time (typically a few minutes/GB). **Note:** The command "h2backup" is only available if OntoBroker is running with H2 as storage. Example: h2backup "d:\\backup\\mybackup.data.db"

`info`	This command checks some properties about the rules and facts and returns a (multiline) info message about the extensional database (EDB, the facts) and the intensional database (IDB, the rules). The output is similar to the output of the "ontology checker" commandline tool.
	syntax: info
	Example output:
	`----------`
	`IDB`
	`----------`
	`[ ] program contains aggregation cycles (not allowed)`
	`[ ] program is not bottomup evaluable (could be bad for performance)`
	`[ ] program is not stratified and must be evaluated in wellfounded mode (bad for performance)`
	`[ ] program contains rules with cross products (bad for performance)`
	`[ ] program contains rules which apply aggregations incorrectly (not allowed)`
	`# user rules: 0`
	`# internal rules (including standard axioms): 48`
	`# rules with negations: 3`
	`# rules with aggregations: 0`
	`# rules with connectors: 0`
	`# rules with other builtins: 4`
	`----------`
	`EDB: RAM.Packed (RAM.Packed)`
	`# of tables: 1`
	`----------`
	`1 tuples in obl:intern:module/1`
	`...`

<table>
<tr><td>insert</td><td>The command <add> has been removed since Ontobroker 6.0. Please use the insert command which has a richer syntax instead. Here are some examples of how to use insert instead of add.

Old syntax in OntoBroker 5.3:
<pre>add otto[name->"Otto"]. // syntax from OB 5.3, invalid in OntoBroker 6.1
add RULE r1: FORALL X p(X) <- q(X). // syntax from OB 5.3, invalid in OntoBroker 6.1</pre>In OntoBroker 6.1 the corresponding commands are:
<pre>insert otto[name->"Otto"].
insert {@{r1} p(?X) :- q(?X).}</pre>The insert command is used to add facts and rules to an ontology. You can add multiple facts or rules with a single command if you use braces {...} around them.

Example with single fact:
<pre>:- module <http://yourcompany.com/ontology1>.
insert Person[name *=> _string].</pre>Example with multiple facts and rules:
<pre>:- module <http://yourcompany.com/ontology1>.
insert {
  Person[name *=> _string, age {0:1} *=> _integer].
  otto:Person[name -> "Otto Waalkers"].
  fritz[name -> "Fritz", age->9].
  @{rule1} ?X:Child :- ?X:Person, ?X.age < 18 .
}</pre>Note:
You can add directives for the module and namespace alias as an header to the command.

You can also generate additional facts by inferencing if you add a where part to the insert statement. In this case, all variable bindings resulting from the query of the where part are replaced in the insert part.

Simple example:
<pre>:- module <http://yourcompany.com/ontology1>.
insert { ?X[lovesToys->true]. } where { ?- ?X:Child. }</pre>Assuming that the results of the query ?- ?X:Child are fritz and susan, this example would insert the facts fritz[lovesToys->true] and susan[lovesToys->true].</td></tr>
<tr><td>isalive</td><td>Checks whether the server is up and running

syntax: isalive

OntoBroker Return Values: Inference Server on port 2267 is up and running ...</td></tr>
<tr><td>kill</td><td>Kills the server. Behaves like shutdown now, but ends the OntoBroker process after 70 seconds if "shutdown now" was not successful within this time. Also see the shutdown command for alternative behavior.

syntax: kill

OntoBroker Return Values: inference server shutting down NOW ...</td></tr>
<tr><td>modify</td><td>With the modify command you can add and remove facts in a single step. As the variable bindings of the where part are used for both insert and delete, this can be quite useful and is easily covered by separate delete and insert commands.

Example:
<pre>// increment the age of all age attributes for persons
modify delete { ?P[age->?AGE]. } insert { ?P[age->?NEWAGE]. } where { ?- ?P:Person[age->?AGE], ?NEWAGE=?AGE + 1 .}</pre></td></tr>
<tr><td>reload</td><td>Causes the server to reload all files from the start-up

syntax: reload</td></tr>
</table>

	OntoBroker Return Values: -
removeExplanation	Removes an explanation with the given name. If no explanation exists no error is reported. syntax: removeExplanation <explanation module> OntoBroker Return Values: Module prooftreefacts_(<module1>) deleted
shutdown	Shutdown without argument causes OntoBroker to process all outstanding queries and commands, but no new queries and commands are accepted anymore. The variant "shutdown NOW" causes OntoBroker to drop all waiting queries and commands and to stop all running queries and commands immediately in a controlled way. Also see the kill command. syntax: shutdown [NOW] OntoBroker Return Values: [INFO]: inference server shutting down ...
stopQuery	Sets an interrupt flag for the currently running query with the given querykey. The query will stop in a controlled way. The argument <querykey> identifies the query to stop. This value is returned immediately after starting a query command. syntax: stopquery <querykey> OntoBroker Return Values: "Interrupt flag set" or "No query found for querykey <querykey>"
Check for OWL 2 RL profile	This command checks if an OWL ontology is conformant with the OWL RL profile. Syntax: checkOWLRLProfile <ontology-uri> Each found violation of the OWL RL profile is listed as output.
Connector cache clear command	This command cleans a connector cache. This command can be executed only if Ontobroker is started with the option Connector.*.Cache.Lifetime = unlimited. Syntax: clear_connector_cache all | cache key all - all tuples will be deleted from the cache. cache key - only tuples for the cache key will be deleted. Cache key is a map term and can contain different values for different connectors.
saveSessionSnapshot	This command saves all session facts and attributes to the given filename To encrypt the session use: saveSessionSnapshot "filename", encrypt. Syntax: saveSessionSnapshot "filename".
loadSessionSnapshot	This command loads all session facts and attributes from the given filename. Syntax: loadSessionSnapshot "filename".
showUserRoles	The command simply runs the query `?- currentUser[hasRole -> ?R].`

19.3.2 Prepared Queries

For complex queries the preparation of the rules and the optimization of the evaluation strategy may take a lot of time (up to 60% of the whole answering process was reported in some cases). To improve the performance of query preparation, the query answering has been divided into two steps:

- The preparation phase and
- The execution phase

This allows the re-usage of prepared programs when querying the same or a similar query. Similar means in terms of given value restrictions that can be changed before the execution takes place.

Note:
Prepared queries are not supported when access control (role-based security) is enabled.

Prepared queries can be maintained with the following commands:

- preparequery
- removepreparedquery
- preparedquerylist

Command	Function	Syntax	OntoBroker Return Values
preparequery	Sends the given query to the server. The name will be used to identify the query for further usage in commands.	preparequery {temporary facts { {flogic-query { [fillnull]	Program prepared for query: <queryname>
preparedquerylist	Returns a list of names of all prepared queries in the server. If no queries exist an empty list is returned.	preparedquerylist	[<list-of-prepared-query-names>]
Removepreparedquery	Removes a prepared query with the given name. If no query exists no error is reported.	removepreparedquery <queryname>	PreparedQuery <queryname> removed from server

Aim	Syntax	Semantics
Preparing a query	preparequery [temporary facts] <objectlogic-query> [fillnull] where <objectlogic-query> := QUERY <queryname>: <objectlogic> and <objectlogic> := any valid objectlogic query that may contain constants _ti which supply placeholders	This command sends the given query to the server. The name will be used to identify the query for further usage in commands. When it is expected to execute the query with temporary facts a sample set of these temporary facts should also be supplied for the preparation process. The preparation of the query depends on the characteristics of the facts in the EDB. If the characteristics change between the preparation and the execution of a query, the server delivers an error and it is necessary to recompile the query. As temporary facts may change these characteristics, we recommend that you send a sample set of these facts.
Executing a prepared query A prepared query is executed like a normal query, but a special syntax is used.	[temporary facts] [substitute (<substitutions>)@pq.] ?- <vars> <- PREPARED_QUERY (<queryname>,<vars>). [<query options>] ?- ?X:?Y.	Executes a query with the given name. If no prepared query is available an error is reported. Temporary facts are added to the EDB for this query if given. Appended query options (orderedby etc.) work as defined. If the prepared query contains substitutions, the values for these placeholders are provided by the special temporary fact with the predicate "substitute". This allows special constants to be replaced as defined in (1) with concrete

		values for a given execution. For example, if there are two substitution placeholders, i.e. _t1 and _t2, the substitution facts must look like this: `substitute(<value for _t1>, <value for _t2>)@pq.`
Removing a prepared query	`removepreparedquery <queryname>`	Removes a prepared query with the given name. If no query exists no error is reported.

Getting a list of all prepared queries	`preparedquerylist`	Returns a list of names of all prepared queries in the server. If no queries exist an empty list is returned.
		Sample
		Prepare a query by executing the following command:
		`preparequery QUERY myQuery: - p(?X) AND ?X < "_t1".`
		Show list of prepared queries by executing the command:
		`preparedquerylist`
		Run the prepared query using any interface available to run a normal query:
		`p(1). p(2). p(3). substitute(2.5)@pq. ?- PREPARED_QUERY ("myQuery",?X).`
		Remove the prepared query by executing the command:
		`removepreparedquery myQuery`

19.3.3 Optimizer Commands

Options	Description
`file filename`	file name for optimization data
`rewrite`	rewrites the file with optimization data, if it exists
`timeout value`	optimization timeout (ms).
`kill`	stops the optimization.
`isoptimizing`	checks whether the optimization is already running
`status`	returns the best result of the running optimization

19.3.3.1 Genetic Optimizer

```
opt_genetic <queryID>
```

Commands to send an optimized query to OntoBroker

Command	Description
opt_genetic status	The return value is a string. The status of the last complete generation will be returned. At least one generation has to be evaluated before valid results can be delivered. OntoBroker Return Values: The return value is a "string" without line separators. Use | as separation character.
opt_genetic kill	Stops a running optimization. It may take a short time, you can use the -isoptimizing command to see when it is done. The timeout window has to fit into the timeout of the socket connection, so it may be to small for some cases.
opt_genetic isoptimizing	Returns if an optimization process is currently running or not. OntoBroker Return Values: The return value is a "string". Example: `opt_genetic file, "test.data", rewrite, timeout, 1000, module,"<http://www.NewOnto1.org/ontology>", "queryID"`

Options	Description
file filename	file name for optimization data
rewrite	rewrites the file with optimization data, if it exists
timeout value	optimization timeout (ms).
kill	stops the optimization.
isoptimizing	checks whether the optimization is already running
status	returns the best result of the running optimization

19.3.3.2 Hillclimbing

The command

```
opt_hillclimbing <queryID>
```

optimizes the bodyordering using a hillclimbing algorithm. You can change the ordering manually.

Usage:

```
opt_hillclimbing [file, filename], [rewrite], module, modulename, queryID
opt_hillclimbing isoptimizing
opt_hillclimbing kill
opt_hillclimbing status
```

Example:

```
opt_hillclimbing file, "test.data", rewrite, module, "<http://www.NewOnto1.org/
ontology>", "queryID"
```

Options	Description
file filename	file name for optimization data
rewrite	rewrites the file with optimization data, if it exists
kill	stops the optimization.
isoptimizing	checks whether the optimization is already running
status	returns the best result of the running optimization

It is displayed as plain text in the file bodyordering.data. You can rename the file in OntoConfig.prp:

```
BodyOrderingDataFile = mybodyordering.data
```

Additional information about the hillclimbing algorithm:
Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, improving it a little each time. When the algorithm cannot see any more improvements, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

19.3.3.3 record_bodyordering

With the command *record_bodyordering <query id>* you are able to record the bodyordering. It is displayed as plain text in the file bodyordering.data. You can rename the file in OntoConfig.prp:

```
BodyOrderingDataFile = mybodyordering.data
```

The file is equal to all optimizer files and is displayed as follows:

Syntax:

```
record_bodyordering <http://www.NewOnto4.org/Test#query>
```

First, the query is shown, and then the evaluation method and then, for each rule, the different orderings for different variables.

```
{} [2,0,1]        //e.g. means no variable instantiation, then the ordering is 2,0,1
{0} [0,1,2]       // e.g. means that the first variable is instantiated in the body
then we have the ordering 0,1,2
```

Note:
The directory of the file depends on the used OntoConfig.prp: if you use the OntoConfig.prp from c:/Ontoconfig.prp, the bodyordering.data will be written in c:/conf/.

19.3.4 Server Commands for Configuration Optimizer

Usage:

```
check_opt_config [option(FULL|QUICK|DBACCESS)], query, module
```

Example:

```
check_opt_config FULL, "?-p(?X)", "<http://www.NewOnto1.org/ontology>"
```

Result:

The result of the command is a time statistic for different configurations (or for the fastest configuration by quick optimization). The fastest configuration won't be saved automatically so the user should set it manually.

Option	Description
FULL	Configuration optimization with generation of time statistics. All configurations are tested. The execution time and speed up factor for all configurations are displayed.
QUICK	Configuration optimization without generation of time statistics. All configurations are tested but only the fastest configuration is displayed.
DBACCESS	Quick configuration optimization for external database access. Only configurations with a minimal number of dbaccessuser are tested. It's not guaranteed that the proposed configuration will be optimal.

19.3.5 Load Save

Save command (saves an ontology to a file)	
Syntax	`save <module> to <file>.`
Example	`save "http://acme.com#"#ontology1 to '/home/me/ontology.obl'.`
OntoBroker reply	`Module "http://acme.com#"#ontology1 saved to file:/home/me/ontology.obl`
Side-effect	Ontology is saved to given filename.

Load command (loads an ontology from a given URL or file)	
Syntax	`load <URL>.` `load "file".`
Example for URL	`load` `<file/sample/sample.obl>.`
OntoBroker reply	`1 ontologies loaded.`
Example for local file	`load` `"E:/me/documents/ontology.obl"`
OntoBroker reply	`1 ontologies loaded.`

About the load command:

Keep in mind that the <load> command only sends an order to the server to load an ontology. The server itself must have access to the ontology!

You have two different options:

- Send an accessible URL to the server and the sever will be able to load the ontology.
- Send a link to an accessible file.

The supported protocols are http, https (bei WebDAV), file and ftp.

19.3.6 Drop Module

Deletes one or more ontologies.

Drop Module	
Syntax	`dropModule <module1> [,<module2>,...].`
Example	`dropModule "http://acme.com#"#ontology1,"http://acme.com#"#ontology2.`
OntoBroker reply	`Modules "http://acme.com#"#ontology1, "http://acme.com#"#ontology2 have been dropped.`

19.3.7 Delete Rule

Removes one or more rules from an ontology.

Delete Rule	
Syntax	`deleteRule <ruleID>@<module>.`
Example	`deleteRule "http://acme.com#"#rule2@"http://acme.com#"#ontology1.`
OntoBroker reply	`Rule "http://acme.com#"#rule2@"http://acme.com#"#ontology1 deleted.`

19.4 Sessions in OntoBroker

As of version 6.1, OntoBroker now supports sessions with private session facts. This means that the session user can add facts to any already existing modules, but nobody but him/her can see these additional facts. On querying, OntoBroker does not differentiate between static facts and session facts, so evaluation of rules works on both static and session facts. Besides storing facts, a session can also store additional session settings such as the default module, default namespace and additional namespace prefix definitions.

An OntoBroker session can be started using various interfaces

- **Socket client**: A new session starts automatically on opening an OntoBrokerConnection, by calling the method.

```
OntoBrokerConnectionManager.getConnection(…)
```

- **OntoBroker Web service:** Calls the operation `startSession(…)`
- **OntoBroker API:** If your code runs in the same VM as OntoBroker, call the method `startSession(…)` of the SessionManager from the OntoBroker API

```
import org.semanticweb.kaon2.api.OntologyManager;
import org.semanticweb.kaon2.api.SessionManager;
import org.semanticweb.kaon2.api.SessionFacade;

OntologyManager om = …;
SessionManager sm = om.getSessionManager();
SessionFacade sf = sm.createSession(…);
```

Note:
In OntoBroker 6.1 the sessions of the Collaboration server are a different kind of session and cannot be used in the way described above.

Using Sessions with OntoBroker API

Sessions have two main interfaces in the OntoBroker API. The SessionManager interface is used to create new sessions or to retrieve existing sessions by the session id.

```
// create new session
SessionFacade sf1 =
       sm.createSession(user,password,params);

// retrieve existing session by id
SessionFacade sf2 = sm.getSession(sessionId);
```

The SessionFacade interface provides all session functionalities:

- Adding and removing facts to/from the session
- Running queries within the session
- Executing commands within the session
- Closing sessions
- Getting/setting session attributes like default module, default namespace and namespace prefixes
- Information about the session like session id, session user, host, last access timestamp, creation timestamp
- Checking session validity

See JavaDoc for `org.semanticweb.kaon2.api.SessionFacade`, for details about its methods. The SessionFacade interface is not the session itself, i.e. there can be multiple SessionFacade objects referencing the same session.

Commands

Several commands have been extended for usage with sessions. The INSERT, DELETE and MODIFY commands can now optionally work on the session facts by adding a "INTO SESSION" or "FROM SESSION" to the command.

Examples:

```
INSERT INTO SESSION { alice:Person[age->18]. }
DELETE FROM SESSION { alice:Person[age->18]. }
       MODIFY DELETE FROM SESSION { ?P[age->?X0]. }
                INSERT INTO SESSION { ?P[age->?X1]. }
                WHERE { ?- ?P[age->?X0], ?X1 = ?X0 + 1. }
```

Variants of modify can also be used for moving data between session and static storage

```
MODIFY DELETE {...} INSERT INTO SESSION {...} WHERE {...}
MODIFY DELETE FROM SESSION {...} INSERT {...} WHERE {...}
```

For setting sessions attribute there is a new SET command:

```
set default prefix = "http://ontoprise.de/".
set prefix a = "http://ontoprise.de/sg#".
set default module = inferencelayer.
```

These default namespace, namespace prefix and default module attributes can be used in subsequent commands and queries without needing to state them again.

New Query Options

When running queries within a session, there are three "data sources" for facts:

- Static facts (i.e. the storage EDB defined by the storage parameter in the OntoConfig.prp)
- Session facts (i.e. the session EDB)
- Temporary facts (facts and rules provided together with the query)

In some situations queries should exclude one of these data sources. For this purpose there are now two new query options:

- **ignoreSession** - ignore all session facts

  Example
  `@{options[ignoreSession]} ?- ?P:Person.`

- **ignoreStorage** - ignore all facts from the storage EDB

  Example
  @{options[ignoreStorage]} ?- ?P:Person.

Built-ins

There are two built-ins to deal with sessions: _activeSessions/4 and _sessionInfo/1

_activeSessions/4

This built-in returns all OntoBroker and Collaboration server sessions.

Example:

```
?- _activeSessions(?Id,?Type,?User,?Host).
```

The variables are bound as follows:

?Id - session id

?Type - "session" or "collab"

?User - user name (may be null if user is unknown)

?Host - host the client is connected from (may be null if host is unknown)

_sessionInfo/1

This built-in returns information on the current session, like the default module, default namespace and other namespace prefixes, and number of facts in the session EDB.

Example:

```
?- _sessionInfo(?InfoMap).
```

The built-in returns a map with following key value pairs:

Key	Value
"id"	session id
"user"	user name
"host"	host
"startTimestamp"	start timestamp of session
"defaultModule"	default module
"namespaces"	map with namespace prefixes
"facts"	number of facts stored in this session

_loginUser/2

This built-in is not new, but if the user name was provided on session creation, it returns user name and host.

Example:

```
?- _loginUser(?User, ?Host).
```

The variables are bound as follows:

`?User` - user name of the current session (may be null if the user is unknown)

`?Host` - host of the client of the current session (may be null if the host is unknown)

Limitations

- Sessions can only store facts, i.e. it cannot store rules, queries or constraints.
- It is only possible to delete facts of the session, but not to hide static facts. i.e. if a fact is stored in the static storage of OntoBroker, deleting it from the session has no effect.
- Sessions can only store facts in existing modules, i.e. at least one fact or rule must exist in this module in the static storage of OntoBroker.
- Persistence of sessions is not supported in OntoBroker 6.1, i.e. session facts are only held in the memory.

19.5 Run Groovy Scripts

With the runGroovyScript command you can run Groovy scripts from within OntoBroker and perform any operations using the OntologyManager. The Groovy script must be located in the groovy subdirectory of the extensions directory.

Syntax

```
runGroovyScript "pathToScript",arg1,...
```

The runGroovyScript command takes at least one argument. The first argument is the path to the Groovy script, relative to the groovy subdirectory of the extensions directory, i.e. the Extensions.Directory property of the OntoConfig.prp. By default this is located at $ONTOBROKER_HOME/extensions/groovy.

Arguments

You can specify more comma-separated arguments. These are just forwarded to the script in the variable args as Term objects. args[0] is always the script name itself. Example:

```
runGroovyScript "myscript.groovy",1.0,2.0 .
```

Bindings

You can use the following predefined variable bindings in the Groovy script:

Variable name	Description
ontologyManager	The instance of org.semanticweb.kaon2.api.OntologyManager
log	Log4j Logger logging into the category [DATAMODEL]
args	The command arguments as org.semanticweb.kaon2.api.logic.Term array (args[0] is the script name)
command	Closure to execute a command. It expects exactly one string argument containing the command in ObjectLogic syntax. Example: command("insert a[b->c]@m1.")
query	Closure to execute query It takes one or two arguments: query(queryText) or query(ontology,queryText). The ontology argument must be either an Ontology, Term or String object, specifying the ontology directly, by module, or by ontologyURI respectively. Example: query("?- ?X::?Y@?M.").each { println it.X,it.Y,it.M }

19.6 Query Client

Query Client (query.cmd/query.sh)

The query client also supports specifying a query file for easier handling of multiline queries:

```
query.cmd -qf myQueryFile.obl
```

or

```
query.cmd -queryFile myQueryFile.obl
```

A query file contains a query (as you would type one in the Web console):

```
:- prefix b = "http://www.ontoprise.com/ontologies#".
:- module = b#ontowebtravel.
:- default prefix = "http://www.ontoprise.com/ontologies/ontowebtravel#".
?- ?X:?Y.
```

When you execute the query file you get your results as usual (this example uses the "travel" example ontology):

```
X     Y
----------
<http://www.ontoprise.com/ontologies/ontowebtravel#h2>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Hotel>
<http://www.ontoprise.com/ontologies/ontowebtravel#john>    <http://www.ontoprise.
com/ontologies/ontowebtravel#Customer>
<http://www.ontoprise.com/ontologies/ontowebtravel#d2>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Date>
```

```
<http://www.ontoprise.com/ontologies/ontowebtravel#d3>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Date>
<http://www.ontoprise.com/ontologies/ontowebtravel#d1>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Date>
<http://www.ontoprise.com/ontologies/ontowebtravel#washington>    <http://www.
ontoprise.com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#madrid>    <http://www.ontoprise.
com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#newyork>    <http://www.ontoprise.
com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#t1>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Transport>
<http://www.ontoprise.com/ontologies/ontowebtravel#h1>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Hotel>
<http://www.ontoprise.com/ontologies/ontowebtravel#h2>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#washington>    <http://www.
ontoprise.com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#madrid>    <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#newyork>    <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#h1>    <http://www.ontoprise.com/
ontologies/ontowebtravel#Location>
```

But it is also possible to abbreviate the namespace terms with the namespaces given in the query text:

```
query.cmd -queryFile myQueryFile.obl -abbreviateNamespaces
```

In this case you get

```
X    Y
----------
h2    Hotel
john          Customer
d2            Date
d3            Date
d1            Date
washington    City
madrid        City
newyork       City
t1            Transport
h1            Hotel
h2            Location
washington    Location
madrid        Location
newyork       Location
h1            Location
```

If you additionally specify the "-formatting" option then the result will be formatted to be easily readable. However, in this case, the whole query will be executed before the result will be printed. The result looks like this:

```
X          Y
---------------------
h2         Hotel
john       Customer
d2         Date
d3         Date
d1         Date
washington City
madrid     City
newyork    City
t1         Transport
h1         Hotel
h2         Location
washington Location
```

```
madrid     Location
newyork    Location
h1         Location
```

If you prefer to get the result without abbreviated namespaces, but formatted the result looks like this:

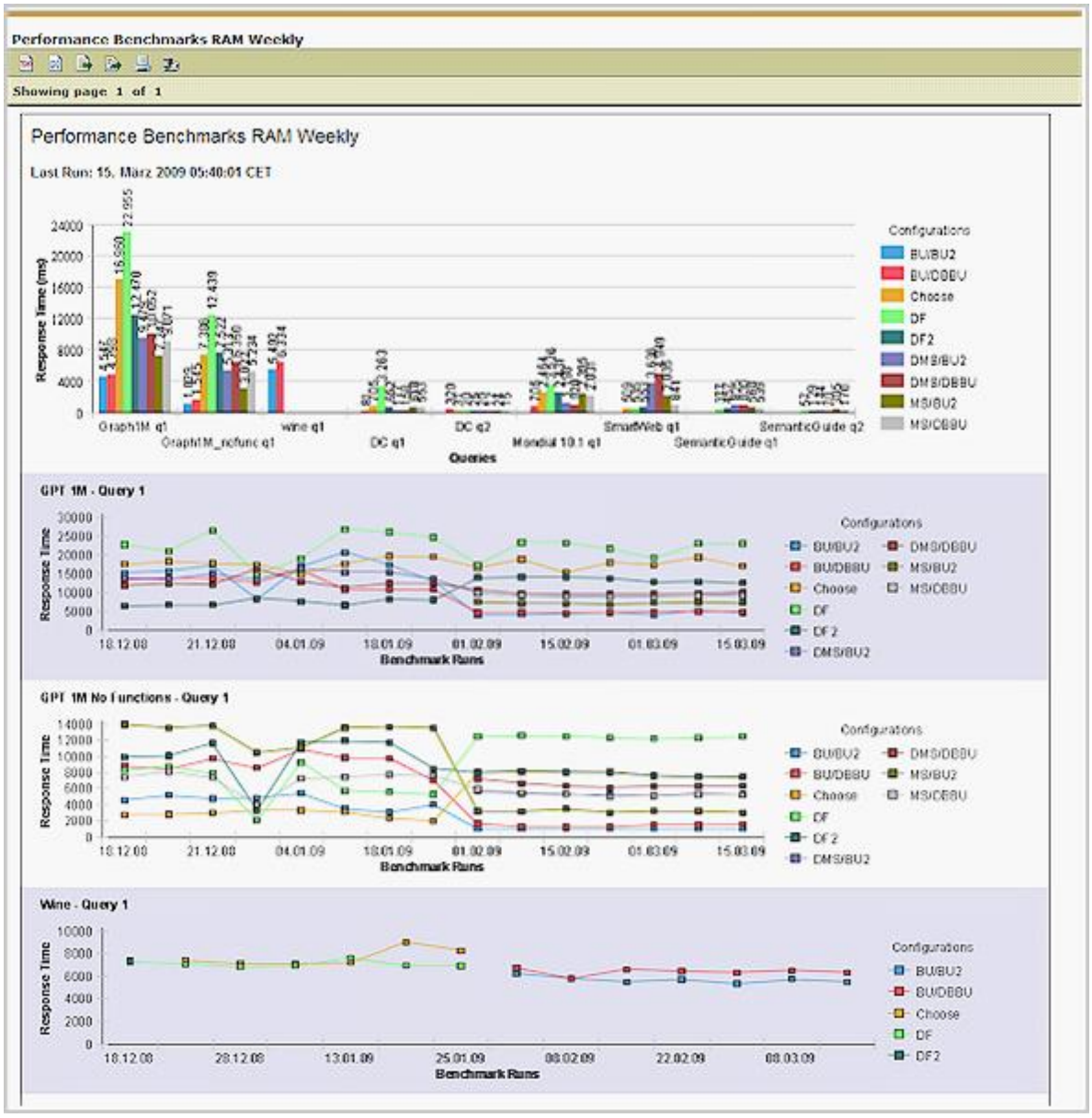
```
X                                                                    Y

-------------------------------------------------------------------------------------------
----------------------------------------
<http://www.ontoprise.com/ontologies/ontowebtravel#h2>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Hotel>
<http://www.ontoprise.com/ontologies/ontowebtravel#john>       <http://www.ontoprise.
com/ontologies/ontowebtravel#Customer>
<http://www.ontoprise.com/ontologies/ontowebtravel#d2>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Date>
<http://www.ontoprise.com/ontologies/ontowebtravel#d3>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Date>
<http://www.ontoprise.com/ontologies/ontowebtravel#d1>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Date>
<http://www.ontoprise.com/ontologies/ontowebtravel#washington> <http://www.ontoprise.
com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#madrid>     <http://www.ontoprise.
com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#newyork>    <http://www.ontoprise.
com/ontologies/ontowebtravel#City>
<http://www.ontoprise.com/ontologies/ontowebtravel#t1>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Transport>
<http://www.ontoprise.com/ontologies/ontowebtravel#h1>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Hotel>
<http://www.ontoprise.com/ontologies/ontowebtravel#h2>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#washington> <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#madrid>     <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#newyork>    <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
<http://www.ontoprise.com/ontologies/ontowebtravel#h1>         <http://www.ontoprise.
com/ontologies/ontowebtravel#Location>
```

20 OntoBroker Tools

You can find all of the tool scripts in the "tools" subdirectory of your OntoBroker installation.

Tool for Performance Benchmarks: (benchmark cmd / sh)

OntoBroker has a built-in tool for measuring query times, which is an important task when designing or optimizing an application. A typical use-case is to find the best configuration for your application by comparing different query evaluation methods. Or you might want to discover regression problems in your application. In this case you can execute the benchmarks regularly and compare them with each other. The results of a benchmark run can be viewed in a graphical report or in a textual output file. Below there is an example of a graphical report with some suites in different configurations.



Benchmark reports are created in sub directories of benchmark.xml

1. Configure your benchmarks

Create a benchmark configuration file with the XML structure shown below. You might reuse the provided example file benchmark/benchmark.xml.

If using relative paths, these paths must be in relation to the startup directory from where the benchmark is run.

Example:

```
<ontobrokerBenchmark>
  <property name="benchmarksuite.config.dir" value="resources/testfiles/
performancetest/testsuites/benchmarksuite/configurations" />
  <property name="benchmark_files.dir" value="resources/testfiles/performancetest/
testsuites/benchmarksuite/benchmarks/benchmark_files" />


  <!-- ================================================
       Format since OntoBroker 6.1
       ================================================ -->
  <suite name="Graph1M">
    <arguments>-obl ${bigfiles.base.dir}/gptInterfaceTest/graph1000000.obl</arguments
>

    <!-- New: Inline query file! -->
    <queryfile>@{options[skipSendingAnswers]} ?- closure(?X,?Y,?Z).</queryfile>
    <configurations>

      <!-- New: Use attribute "name" and "file" instead of xml element "configName"
and "configFile" -->
      <configuration name="BU/BU2" file="${benchmarksuite.config.dir}/BU_BU2.prp" />
      <configuration name="BU/DBBU" file="${benchmarksuite.config.dir}/BU_DBBU.prp" /
>
      <configuration name="BU/BU3" file="${benchmarksuite.config.dir}/BU_BU3.prp" />

      <!-- New: Inline config file! -->
      <configuration name="DF2">
OntologyLanguage = ObjectLogic
Storage=RAM.AVL.Packed
EvaluationMethod=DynamicFiltering2</configuration>
    </configurations>
  </suite>

  <!-- ================================================
       Format before OntoBroker 6.1 - still valid!
       ================================================ -->
  <suite>
    <name>Graph1M_nofunc</name>
    <arguments>-obl ${bigfiles.base.dir}/gptInterfaceTest/graph1000000nofunctions.obl
</arguments>
    <queryfile>${benchmark_files.dir}/GPT.queries.obl</queryfile>
    <configurations>
      <configuration>
        <configName>BU/BU2</configName>
        <configFile>${benchmarksuite.config.dir}/BU_BU2.prp</configFile>
      </configuration>
      <configuration>
        <configName>BU/DBBU</configName>
        <configFile>${benchmarksuite.config.dir}/BU_DBBU.prp</configFile>
      </configuration>
      <configuration>
        <configName>BU/BU3</configName>
        <configFile>${benchmarksuite.config.dir}/BU_BU3.prp</configFile>
      </configuration>
      <configuration>
        <configName>DF2</configName>
        <configFile>${benchmarksuite.config.dir}/DF2.prp</configFile>
      </configuration>
    </configurations>
  </suite>
</ontobrokerBenchmark>
```

- The suite's name tag must be unique for the benchmark configuration
- The arguments tag must contain the loaded ontologies (e.g. -flo ...), other startup parameters are allowed
- The queryfile tag must contain the path to a text file which contains the queries; within this query file the queries must be written one per line
- The configName tag must be unique within the <suite> part but can be reused in different suites
- The configFile tag must contain the path to a OntoBroker configuration file

2. Execute your benchmarks

The startup script "benchmark.cmd" (resp. "benchmark.sh" for Linux/Unix) - located at [Ontobroker_HOME\tools] - executes the benchmarks and stores the results in a given output directory. The following arguments are allowed:

Argument	Function
`benchmark.cmd <benchmarksuiteFileName>`	stores the results at default output directory <benchmarksuiteFileName>_output\
`benchmark.cmd <benchmarksuiteFileName> <benchmarksuiteOutputDir>`	stores the results at the given output directory
`benchmark.cmd <benchmarksuiteFileName> -clean`	deletes the default output directory (<benchmarksuiteFileName>_output\)
`benchmark.cmd <benchmarksuiteFileName> -clean <benchmarksuiteOutputDir>`	deletes the given output directory

The provided example benchmark can be run via this command:

```
tools/benchmark.cmd benchmark/benchmark.xml
```

3. Look at the benchmark results

The benchmark results can be found in the given output directory: As a simple text file results.txt or as a HTML report diagrams_benchmark/benchmark.html (in conjunction with the results database benchmarkdata.h2.db, that contains the results of all benchmark runs for historical comparisons).

Script location:

Windows: tools\benchmark.cmd
Linux: tools/benchmark.sh

Tool to convert files

This commandline tool allows files to be converted from one ontology format to another. Conversions from and to ObjectLogic, OWL and RDF(S) are supported. The options are:

```
convert [-fenc <ENCODING>] [-all] <inputFile> <outputFile>
```

Converting a file from ObjectLogic to OWL can be done via

```
tools\convert.cmd -fenc UTF-8 examples/objectlogic/travel.obl travel.owl
```

or

```
convert.cmd -fenc UTF-8 ../examples/objectlogic/travel.obl travel.owl
```

Note:
The output format is given by the target filename.

Script location:

Windows: `tools\convert.cmd`
Linux: `tools/convert.sh`

Tool for checking NamesGround compliance

This tool loads an ontology and checks various properties of the rules (if all rules are bottomup evaluable, the number of rules, and so on). It also checks if you can safely use the configuration options "ConceptNamesGround", "AttributeNamesGround", or "ModuleNamesGround". Usage:

```
checkontology <file>
```

You should always execute this tool after you have changed the rules. This helps to detect possible problems as early as possible.

Script location:

Windows: `tools\checkfornamesgroundsafety.cmd`
Linux: `tools/checkfornamesgroundsafety.sh`

Tool to dump list with built-ins

This tool dumps all built-ins available in OntoBroker for the current configuration.

Script location:

Windows: `tools\logbuiltins.cmd`
Linux: `tools/logbuiltins.sh`

Tool to compact H term store

If you use H2 as storage, you can use this tool to remove unused, i.e. no longer used terms from the datamodel.

Note:
This will not result in a smaller database file. To shrink the H2 database file, an explicit database compaction has to be performed. See H2 database website for details.

Tool to execute commands
Use this script to execute commands directly from the command line.
Windows:

```
command.cmd [-h <host>] [-p <port>] <command>
```

Linux:

```
./command.sh [-h <host>] [-p <port>] <command>
```

Example:

```
command.cmd "commands"
```

Tool to execute queries

Use this script to execute queries directly from the command line.
Windows:

```
query.cmd [-p <port>] <querytext>
```

Linux:

```
./query.sh [-p <port>] <querytext>
```

Example:

```
query.cmd "?- $module(?M)."
```

21 Appendix

21.1 Ontology Transformation

Transforming **ObjectLogic into RDF** Schema is in most cases straightforward. Basically every ObjectLogic fact is represented as one (or more) RDF triple. Whenever a functional term occurs, it is mapped to an IRI using the predefined namespace 'obl:term#':

```
<myns:myFunction>(arguments)   <=>   <obl:term#%3Cmyns:myFunction%3E(arguments)>
```

While ObjectLogic allows constants of every type as object identifiers, RDF imposes several restrictions on the structure of statements. For example, anonymous nodes are not allowed as a predicate in RDF statements. All of the ObjectLogic facts not conforming to this restriction are discarded.

For every transformation a summary log is provided. It shows potential transformation problems like axioms not supported in the target ontology language.

Schema

ObjectLogic concept and property hierarchies are directly transformed into the RDF Schema class hierarchy. For attribute and relation definitions, the RDF Schema elements domain and range are used. This intentional break with the RDF Schema semantics interpreting domain and range as constraints favors a more "information preserving" instead of a semantically correct transformation.

The global definition of properties in RDF, in contrast to concept-bound in ObjectLogic, implies that attributes/relation using the same identifier will be merged during the transformation, i.e. they share the same domain and range definitions.

Finally, properties of attributes/relations like cardinality restrictions, transitivity, symmetry, and inverse-of definitions are not supported by the RDF Schema. If this information needs to be taken into account, an transformation to OWL should be considered. For the RDF Schema only the declaration of the attribute/ relation itself is stored.

```
Person[].                      <=>   :Person a rdfs:Class.
Person::Human.                 <=>   :Person rdfs:subClassOf :Human.
loves<<knows.                  <=>   :loves rdfs:subPropertyOf :knows.
Person[name{1:2}*=>_string].    =>   :name rdfs:domain :Person; rdfs:range xsd:
string.
Person[p{symmetric}*=>()].      =>   :p a rdf:Property.
Person[p{transitive}*=>()].     =>   :p a rdf:Property.
Person[p{transitive}*=>()].     =>   :p a rdf:Property.
Person[p{inverseOf(q)}*=>()].   =>   :p a rdf:Property. :q a rdf:Property.
```

Instances

Assertions, both for instances and concepts, are transformed preserving the datatype.

```
eve:Person.                           <=>   :eve a :Person.
eve[name->"Eve"].                     <=>   :eve :name "Eve"^^xsd:string.
eve[name->"Eve@"^^_PlainLiteral].     <=>   :eve :name "Eve".
eve[name->"Evita@es"^^_PlainLiteral]. <=>   :eve :name "Evita"@es.
eve[p->true].                         <=>   :eve :p "true"^^xsd:boolean.
eve[p->1].                            <=>   :eve :p "1"^^xsd:int.
eve[p->2.3].                          <=>   :eve :p 2.3e0 .
eve[p->"4.0"^^_decimal].              <=>   :eve :p 4.0 .
eve[p->"11:55:32"^^_time].            <=>   :eve :p "11:55:32"^^xsd:time.
```

21.1.1 Rules and Queries

Rules, queries, and constraints are not supported by the RDF Schema.

21.1.2 RDF Schema to ObjectLogic

Transforming the RDF Schema into ObjectLogic is, in most cases, analogous to the inverse transformation. IRIs using the reserved namespace 'obl:term#' are mapped to the encoded ObjectLogic term:

```
<obl:term#%3Cmyns:myFunction%3E(arguments)>   <=>   <myns:myFunction>(arguments)
```

If a term cannot be encoded due to invalid ObjectLogic syntax the IRI is kept.

A summary log is provided for every transformation. It shows potential transformation problems like issues related to domain/range ambiguity.

Schema

Class and property declarations and concept and property hierarchies are transformed into ObjectLogic equivalent facts.

```
:Person a rdfs:Class.              <=>   Person[].
:Person rdfs:subClassOf :Human.    <=>   Person::Human.
:knows a rdf:Property.             <=>   _property(knows).
:loves rdfs:subPropertyOf :knows.  <=>   loves<<knows.
```

The transformation of domain and range definitions is slightly more complex. All of the related domain/ range information is collected for every RDF property. Properties not contained in at least one domain and in one range definition statement are ignored. All other properties are interpreted as schema restrictions, which is semantically not correct, however the best trade-off in terms of round-tripping.

For example, the triples

```
:age rdfs:domain :Person.
:age rdfs:domain :Building.
:age rdfs:range xsd:double.
:name rdfs:range xsd:string.
```

will be transformed into ObjectLogic

```
Person[age*=>_double].
Building[age*=>_double].
```

Instances

Instance declarations and general assertions are transformed preserving the datatype.

```
:eve a :Person.                 <=>   eve:Person.
:eve :name "Eve"^^xsd:string.   <=>   eve[name->"Eve"].
:eve :name "Eve".               <=>   eve[name->"Eve@"^^_PlainLiteral].
:eve :name "Evita"@es.          <=>   eve[name->"Evita@es"^^_PlainLiteral].
:eve :p "true"^^xsd:boolean.    <=>   eve[p->true].
:eve :p 1 .                     <=>   eve[p->1].
:eve :p 2.3e0 .                 <=>   eve[p->2.3].
:eve :p 4.0 .                   <=>   eve[p->"4.0"^^_decimal].
:eve :p "11:55:32"^^xsd:time.   <=>   eve[p->"11:55:32"^^_time].
:eve :p "foo"^^my:type.         ==>   eve[p->"foo"].
```

21.1.3 ObjectLogic to OWL

The transformation translates ObjectLogic facts to OWL axioms. P-atoms (literals), queries and constraints are not supported. Axioms which are unsupported or contain unsupported parts are ignored. Rules are are not supported.

Transforming Concepts, Properties and Individuals

ObjectLogic does not explicitly differ between concepts, properties and individuals. Any term can be used as one of these entity types.

On the other hand, OWL strictly typifies entities within axioms; entities are identified by URIs. The axioms to be created can thereby depend on the entity type, e.g. if a domain is specified for a property, it needs to be an ObjectPropertyDomain if the entity is an object property, but a DataPropertyDomain if the property is a data

property.

Since ObjectLogic does not explicitly mark the used terms with an entity type, there are ambiguities, e.g. in 'a [p->b]' it is not clear if 'b' is meant to be an individual or a constant, thereby it is not clear if this fact should be translated to an ObjectPropertyAssertion or a DataPropertyAssertion.

The transformation tries to detect the property types for terms and translates the facts accordingly. For example, if the ontology contains a schema fact 'A[p*=>B]' where B does not identify a standard datatype, it assumes that 'p' is an object property. If instead 'A[p*=>_string]' is contained in the ontology, the transformation assumes that 'p' is a data property. If both schema facts are included, there is a conflict and the transformation handles the type of 'p' according to the context.

Transforming ObjectLogic Terms to OWL Entities

A given ObjectLogic term is transformed to an OWL entity as follows:

- If the term is an rif:iri constant, the iri is used as entity URI.
- Otherwise, the term is formatted as ObjectLogic string L and the URI for the entity is obl:term#E, where E is the URL-encoded version of L.

Constraints versus deriving Knowledge

In ObjectLogic most of the schema facts are handled as constraints and will thereby not affect the derived facts but may just make the ontology inconsistent. E.g. 'A[p{1:1}*=>B]' together with 'a:A', 'a[p->b]' and 'a[p->c]' form an inconsistent ontology, but do not derive that 'b' and 'c' need to be the same.

In OWL on the other hand, from 'SubClassOf(A ObjectExactCardinality(1 p B))' together with 'ClassAssertion(A a)', 'ObjectPropertyAssertion(p a b)' and 'ObjectPropertyAssertion(p a c)' it follows that 'b' and 'c' must be the same individual.

The semantic difference between constraints on the one hand and derived knowledge on the other hand is ignored by the transformation, i.e. the transformation handles the ObjectLogic schema facts as if they would actually add to the knowledge rather than just building constraints.

Property Attributes

ObjectLogic binds property attributes to some concepts, e.g. 'A[p{symmetric}*=>()]' binds 'p' to 'A'. In OWL this is not (directly) possible and the transformation skips the information about the concept, i.e. will just declare 'p' to be symmetric, so 'A' is a domain for 'p' (ObjectPropertyDomain(p A).

Inheritance

Basically A[p=>B] is ignored. Certainly it is asked to appropriate the type of 'p'. In this case 'p' is declared as an ObjectProperty.

21.1.4 Basic Transformations

The following table shows the basic transformation under 'natural' assumptions, ignoring several special cases. For schema facts which contains properties, property domain and property range facts might be created.

```
  a:A.                              =>   ClassAssertion(A a)

  a[p->b].                          =>   ObjectPropertyAssertion(p a b) respectively
DataPropertyAssertion(p a b) respectively AnnotationAssertion(p a b)
  a[_comment->"text"{en}].          =>   AnnotationAssertion(rdfs:comment a "text"@en)
  a[_representation ->"text"{en}]. =>   AnnotationAssertion(rdfs:label a "text"@en)
  a[_synonym("en")->"text"].        =>   AnnotationAssertion(<obl:reserved:synonym> a
"text"@en)

  A[].                              =>   Declaration(Class(A))

  B::A.                             =>   SubClassOf(B A)
  p<<q.                             =>   SubPropertyOf(p q)

  A[p{inverseOf(q)}*=>B].           =>   InverseObjectProperties(p q)
  A[p{symmetric}*=>()].             =>   SymmetricObjectProperty(p)
```

```
  A[p{transitive}*=>()].           =>   TransitiveObjectProperty(p)

  A[p{n:*}*=>()].                  =>   SubClassOf(A ObjectMinCardinality(n p))
respectively SubClassOf(A DataMinCardinality(n p))
  A[p{0:n}*=>()].                  =>   SubClassOf(A ObjectMaxCardinality(n p))
respectively SubClassOf(A DataMaxCardinality(n p))
                                       for n ? 1
               .                   =>   FunctionalObjectProperty(p) respectively
FunctionalDataProperty(p)
                                       for n = 1

  A[p*=>B].                        =>   ObjectPropertyDomain(p A),
ObjectPropertyRange(p B)
  A[p*=>_string].                  =>   DataPropertyDomain(p A), DataPropertyRange(p
xsd:string)
```

Domain and Range

	OWL	OWL-in-ObjectLogic	ObjectLogic
Local Range Specification	SubClassOf(:Person ObjectAllValuesFrom(: hasMother :Woman))	Person::_AllValuesFrom (hasMother, Woman)	Person [hasMother*=>Woman]
Comparing Property Type Declarations	• SubClassOf(:Person ObjectAllValuesFrom(: hasMother :Woman)) • Property type declaration by each use	• Person::_AllValuesFrom (hasMother, Woman) • _Declaration (hasMother, owl#ObjectProperty) • Property type declaration per ontology	• Person [hasMother*=>Woman] • Implicit property type declaration by range only
Global Range Specification	ObjectPropertyDomain(: hasMother :Person), ObjectPropertyRange(: hasMother :Woman)	ObjectPropertyDomain(: hasMother :Person), ObjectPropertyRange(: hasMother :Woman)	• Person [hasMother*=>Woman] • Implicit property type declaration by range only
One Domain, no Range	• ObjectPropertyDomain(: hasMother :Person) • Domain: Person • Range: anything		• Person [hasMother*=>rdfs#Resource] • Domain: Person • Range: filler rdfs#Resource (without any special meaning)
No Domain, one Range	• ObjectPropertyRange(: hasMother :Woman) • Domain: anything • Range: Woman		• owl#Thing [hasMother*=>Woman] • Domain: filler owl#Thing (without any special meaning) • Range: Woman
One Domain, one Range	ObjectPropertyDomain(: hasMother :Person), ObjectPropertyRange(: hasMother :Woman)		Person [hasMother*=>Woman]
One Domain, multiple Ranges	ObjectPropertyDomain(: hasMother :Person),		Person [hasMother*=>Woman],

	ObjectPropertyRange(: hasMother :Woman), ObjectPropertyRange(: hasMother :Mother)		Person [hasMother*=>Mother]
Union of Domains, one Range	ObjectPropertyDomain(: hasMother ObjectUnionOf(:Man : Woman)), ObjectPropertyRange(: hasMother :Woman)		Man [hasMother*=>Woman], Woman [hasMother*=>Woman]
Union of Domains, different local Ranges	ObjectPropertyDomain(: hasMother ObjectUnionOf(:Man : Woman)), SubClassOf(:Man ObjectAllValuesFrom(: hasMother :Woman)), SubClassOf(:Woman ObjectAllValuesFrom(: hasMother :Mother))		Man [hasMother*=>Woman], Woman [hasMother*=>Mother]
Multiple Domains, one Range	• ObjectPropertyDomain(: hasMother :Creature), ObjectPropertyDomain(: hasMother :Person), ObjectPropertyRange(: hasMother :Woman) • Domain: intersection of Creature and Person • Range: Woman		• Creature [hasMother*=>Woman], Person [hasMother*=>Woman] • Domain: union of Creature and Person • Range: Woman • Retransformed like 'Union of Domains, one Range'
Multiple Domains, multiple Ranges	• ObjectPropertyDomain(: hasMother :Creature), ObjectPropertyDomain(: hasMother :Person), ObjectPropertyRange(: hasMother :Woman), ObjectPropertyRange(: hasMother :´Mother) • Domain: intersection of Creature and Person • Range: intersection of Woman and Mother		• Creature [hasMother*=>Woman], Creature [hasMother*=>Mother], Person [hasMother*=>Woman], Person [hasMother*=>Mother] • Domain: union of Creature and Person • Range: intersection of Woman and Mother • Retransformed like 'Union of Domains, different local Ranges'

21.1.5 OWL to ObjectLogic

The transformation translates OWL axioms to ObjectLogic facts. SWRL rules are not supported at all. Axioms which are unsupported or contain unsupported parts are ignored.

Transforming named OWL Entities to ObjectLogic Terms

A named OWL entity is translated to an ObjectLogic term by creating a constant of type rif:iri which consists of the URI of the entity. There is one exception: if the URI of the entity starts with 'obl:term#' then the local part

of the URI is assumed to be the (URL-encoded) ObjectLogic serialization of a term which will be parsed and used instead of a rif:iri constant.

Object Property Expressions

The inverse object property expression is unsupported.

Data Ranges

Data ranges besides plain datatypes are unsupported.

21.1.6 Complex Class Expressions

A subset of complex class expressions is supported if used as a super description within a sub class axiom, but no where else. Recursive complex class expressions are unsupported.

Basic Transformations

The following table shows the basic transformation under 'natural' assumptions, ignoring several special cases. Axioms not listed here are unsupported.

The transformation requires that A has been detected as an object property domain for the object property p and B as object property domain of the object property r. Unsupported axioms are skipped by the transformation.

```
ClassAssertion(A a)                              =>   a:A.

ObjectPropertyAssertion(q a b)                   =>   a[q->b].
DataPropertyAssertion(q a "text"@en)             =>   a[q->"text"{en}.
AnnotationAssertion(q a b)                       =>   a[q->b].
AnnotationAssertion(rdfs:comment a "text"@en)    =>   a[_comment->"text"{en}].
AnnotationAssertion(rdfs:label a "text"@en)      =>   a[_representation ->"text"{en}]
AnnotationAssertion(<obl:reserved:synonym> a "text"@en)
                                                   =>   a[_synonym("en")->"text"].

Declaration(Class(A))                            =>   A[].

SubClassOf(B A)                                  =>   B::A.


InverseObjectProperties(p r)                     =>   A[p{inverseOf(r)}*=>B].
SymmetricObjectProperty(p)                       =>   A[p{symmetric}*=>()].
TransitiveObjectProperty(p)                      =>   A[p{transitive}*=>()].

SubClassOf(A ObjectMinCardinality(1 p))          =>   A[p{1:*}*=>()].
SubClassOf(A DataMinCardinality(1 p))            =>   A[p{1:*}*=>()].
SubClassOf(A ObjectMaxCardinality(1 p))          =>   A[p{0:1}*=>()].
SubClassOf(A DataMaxCardinality(1 p))            =>   A[p{0:1}*=>()].

{A domain of p, B range of p}                    =>   A[p*=>B].
```

21.1.7 OWL2 (RL) in ObjectLogic

Predefined and Named Classes

Language Feature	OWL2 Functional Syntax1 RDF Syntax2	ObjectLogic Syntax
named class	CN CN	CN
universal class	owl:Thing owl:Thing	owl:Thing for ObjectLogic
empty class	owl:Nothing owl:Nothing	owl:Nothing for ObjectLogic

[1] Second column, upper part: OWL2 Functional Syntax

[2] Second column, lower part: RDF Syntax

Boolean Connectives and Enumeration of Individuals

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
intersection	ObjectIntersectionOf(C1 … Cn) _:x rdf:type owl:Class. _:x owl:intersectionOf (C1 … Cn).	_IntersectionOf([C1, C2, …,Cn])
union	ObjectUnionOf(C1 … Cn) _:x rdf:type owl:Class. _:x owl:unionOf (C1 … Cn).	_UnionOf([C1, … Cn])
complement	ObjectComplementOf(C) _:x rdf:type owl:Class. _:x owl:complementOf C.	_ComplementOf(C)
enumeration	ObjectOneOf(a1 … an) _:x rdf:type owl:Class. _:x owl:oneOf (a1 … an).	_OneOf([a1, …, an])

Object Property Restrictions

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
universal	ObjectAllValuesFrom(P C) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:allValuesFrom C	_AllValuesFrom(P, C)
existential	ObjectSomeValuesFrom(P C) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:someValuesFrom C	_SomeValuesFrom(P, C)
individual value	ObjectHasValue(P a) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:hasValue a.	_HasValue(P, a)
local reflexivity	ObjectHasSelf(P) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:hasSelf "true"^^xsd:boolean.	_HasSelf(P)
exact cardinality	ObjectExactCardinality(n P) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:cardinality n.	_ExactCardinality(n, P, owl#Thing)
qualified exact cardinality	ObjectExactCardinality(n P C) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:qualifiedCardinality n. _:x owl:onClass C.	_ExactCardinality(n, P, C)
maximum cardinality	ObjectMaxCardinality(n P) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:maxCardinality n.	_MaxCardinality(n, P, owl#Thing)
qualified maximum cardinality	ObjectMaxCardinality(n P C) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:maxQualifiedCardinality n. _:x owl:onClass C.	_MaxCardinality(n, P, C)
minimum cardinality	ObjectMinCardinality(n P) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:minCardinality n.	_MinCardinality(n, P, owl#Thing)
qualified minimum cardinality	ObjectMinCardinality(n P C) _:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:minQualifiedCardinality n.	_MinCardinality(n, P, C)

	_:x owl:onClass C.	

Data Property Restrictions

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
universal	DataAllValuesFrom(R D) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:allValuesFrom D.	_AllValuesFrom(R, D)
existential	DataSomeValuesFrom(R D) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:someValuesFrom D.	_SomeValuesFrom(R, D)
literal value	DataHasValue(R v) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:hasValue v.	_HasValue(R, v)
exact cardinality	DataExactCardinality(n R) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:cardinality n.	_ExactCardinality(n, R, rdfs#Literal)
qualified exact cardinality	DataExactCardinality(n R D) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:qualifiedCardinality n. _:x owl:onDataRange D.	_ExactCardinality(n, R, D)
maximum cardinality	DataMaxCardinality(n R) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:maxCardinality n.	_MaxCardinality(n, R, rdfs#Literal)
qualified maximum cardinality	DataMaxCardinality(n R D) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:maxQualifiedCardinality n. _:x owl:onDataRange D.	_MaxCardinality(n, R, D)
minimum cardinality	DataMinCardinality(n R) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:minCardinality n.	_MinCardinality(n, P, rdfs#Literal)
qualified minimum cardinality	DataMinCardinality(n R D) _:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:minQualifiedCardinality n. _:x owl:onDataRange D.	_MinCardinality(n, R, D)

Restrictions Using n-ary Data Range

In the following table 'Dn' is an n-ary data range.

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
n-ary universal1	DataAllValuesFrom(R1 … Rn Dn) _:x rdf:type owl:Restriction. _:x owl:onProperties (R1 … Rn). _:x owl:allValuesFrom Dn.	not supported
n-ary existential1	DataSomeValuesFrom(R1 … Rn Dn) _:x rdf:type owl:Restriction. _:x owl:onProperties (R1 … Rn). _:x owl:someValuesFrom Dn.	not supported

[1] Unsupported by OWL API and thus unsupported in ObjectLogic.

Properties

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
named object property	PN PN	PN
universal object property	owl:topObjectProperty owl:topObjectProperty	owl#topObjectProperty
empty object property	owl:bottomObjectProperty owl:bottomObjectProperty	owl#bottomObjectProperty
inverse property	ObjectInverseOf(PN) _:x owl:inverseOf PN	_InverseOf(PN)

Data Property Expressions

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
named data property	R R	R
universal data property	owl:topDataProperty owl:topDataProperty	owl:#topDataProperty
empty data property	owl:bottomDataProperty owl:bottomDataProperty	owl:#bottomDataProperty

Individuals & Literals

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
named individual	aN aN	
anonymous individual1	_:a _:a	_#'a'
literal (datatype value)	"abc"^^DN "abc"^^DN	"abc"^^DN

[1] The semantics of anonymous individuals is not supported by OntoBroker, instead anonymous individuals are mapped to unique ids on import.

Data Ranges

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
named datatype	DN DN	DN
data range complement	DataComplementOf(D) _:x rdf:type rdfs:Datatype. _:x owl:datatypeComplementOf D.	_ComplementOf(D)
data range intersection	DataIntersectionOf(D1…Dn) _:x rdf:type rdfs:Datatype. _:x owl:intersectionOf (D1…Dn).	_IntersectionOf([D1, D2, …,Dn])
data range union	DataUnionOf(D1…Dn) _:x rdf:type rdfs:Datatype. _:x owl:unionOf (D1…Dn).	_UnionOf([D1, … ,Dn])
literal enumeration	DataOneOf(v1 … vn) _:x rdf:type rdfs:Datatype. _:x owl:oneOf (v1 … vn).	_OneOf([v1, … ,vn])
datatype restriction	DatatypeRestriction(DN f1 v1 … fn vn) _:x rdf:type rdfs:Datatype. _:x owl:onDatatype DN. _:x owl:withRestrictions (_:x1 ... _:xn). _:xj fj vj. j=1…n	_DatatypeRestriction(DN, [f1, v1, … fn, vn])

Axioms

OWL 2 RL supports all axioms of OWL 2 apart from disjoint unions of classes (DisjointUnion) and reflexive object property axioms (ReflexiveObjectProperty).

Class Expression Axioms

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
subclass	SubClassOf(C1 C2) C1 rdfs:subClassOf C2.	C1::C2
equivalent classes	EquivalentClasses(C1 C2) C1 owl:equivalenClass C2.	_EquivalentClasses(C1, C2)
equivalent classes	EquivalentClasses(C1 … Cn) Cj owl:equivalentClass Cj+1. j=1…n-1	_EquivalentClasses1([C1,…, Cn])
disjoint classes	DisjointClasses(C1 C2) C1 owl:disjointWith C2.	_DisjointClasses(C1, C2)
pairwise disjoint classes	DisjointClasses(C1 … Cn) _:x rdf:type owl:AllDisjointClasses. _:x owl:members (C1 … Cn).	_DisjointClasses1([C1,…, Cn])
disjoint union	DisjointUnionOf(CN C1 … Cn) CN owl:disjointUnionOf (C1 … Cn).	_DisjointUnionOf(CN, ([C1,…, Cn])

Object Property Axioms

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
subproperty	SubObjectPropertyOf(P1 P2) P1 rdfs:subPropertyOf P2.	P1 << P2
property chain inclusion	SubObjectPropertyOf (ObjectPropertyChain(P1 … Pn) P) P owl:propertyChainAxiom (P1 … Pn).	_ PropertyChain([P1, P2, …,Pn])
property domain	ObjectPropertyDomain(P C) P rdfs:domain C.	C[P*=>rdfs#Resource]
property range	ObjectPropertyRange(P C) P rdfs:range C.	
equivalent properties	EquivalentObjectProperties(P1 … Pn) Pj owl:equivalentProperty Pj+1. j=1…n-1	_EquivalentProperties1([P1 … Pn])
disjoint properties	DisjointObjectProperties(P1 P2) P1 owl:propertyDisjointWith P2.	_DisjointProperties(P1 P2)
pairwise disjoint properties	DisjointObjectProperties(P1 … Pn) _:x rdf:type owl:AllDisjointProperties. _:x owl:members (P1 … Pn).	_DisjointProperties1([P1 … … Pn])
inverse properties	InverseObjectProperties(P1 P2) P1 owl:inverseOf P2.	owl#Thing[P1{inverseOf(P2)*=>()]
functional property	FunctionalObjectProperty(P) P rdf:type owl:FunctionalProperty.	owl#Thing[P{0:1}*=>()]
inverse functional property	InverseFunctionalObjectProperty(P) P rdf:type owl: InverseFunctionalProperty.	owl#Thing[P{inverseFunctional}*=>()]
reflexive property	ReflexiveObjectProperty(P) P rdf:type owl:ReflexiveProperty.	Not supported, since not supported in OWL RL
irreflexive property	IrreflexiveObjectProperty(P) P rdf:type owl:IrreflexiveProperty.	owl#Thing [P {irreflexive} *=> ()].
symmetric property	SymmetricObjectProperty(P) P rdf:type owl:SymmetricProperty.	owl#Thing [P {symmetric} *=> ()].
asymmetric property	AsymmetricObjectProperty(P) P rdf:type owl:AsymmetricProperty.	owl#Thing [P {asymmetric} *=> ()].
transitive property	TransitiveObjectProperty(P) P rdf:type owl:TransitiveProperty.	owl#Thing [P {transitive} *=> ()].

Data Property Axioms

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
subproperty	SubDataPropertyOf(R1 R2) R1 rdfs:subPropertyOf R2.	R1 << R2
property domain	DataPropertyDomain(R C) R rdfs:domain C.	C[R*=>rdfs#Resource]
property range	DataPropertyRange(R D) R rdfs:range D.	owl#Thing[R*=>D]
equivalent properties	EquivalentDataProperties(R1 R2)	_EquivalentProperties(R1,R2)
equivalent properties	EquivalentDataProperties(R1 … Rn) Rj owl:equivalentProperty Rj+1. j=1…n-1	_EquivalentProperties1([R1 … Rn])
disjoint properties	DisjointDataProperties(R1 R2) R1 owl:propertyDisjointWith R2.	_DisjointProperties(R1 R2)
pairwise disjoint properties	DisjointDataProperties(R1 … Rn) _:x rdf:type owl:AllDisjointProperties. _:x owl:members (R1 … Rn).	_DisjointProperties1([R1 … … Rn])
functional property	FunctionalDataProperty(R) R rdf:type owl:FunctionalProperty.	

Datatype Definitions

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
datatype definition	DatatypeDefinition(DN D) DN owl:equivalentClass D.	_DatatypeDefinition(DN, D)

Assertions

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
individual equality	SameIndividual(a1 … an) aj owl:sameAs aj+1. j=1…n-1	_SameIndividual(a1, a2) _SameIndividual1([a1 … an])
individual inequality	DifferentIndividuals(a1 a2) a1 owl:differentFrom a2.	_DifferentIndividuals(a1, a2)
pairwise individual inequality	DifferentIndividuals(a1 … an) _:x rdf:type owl:AllDifferent. _:x owl:members (a1 … an).	_DifferentIndividuals1([a1 … an])
class assertion	ClassAssertion(C a) a rdf:type C.	a:C
positive object property assertion	ObjectPropertyAssertion(PN a1 a2) a1 PN a2.	a1[PN ->a2]
positive data property assertion	DataPropertyAssertion(R a v) a R v.	a[R ->v]
negative object property assertion	NegativeObjectPropertyAssertion(P a1 a2) _:x rdf:type owl: NegativePropertyAssertion. _:x owl:sourceIndividual a1. _:x owl:assertionProperty P. _:x owl:targetIndividual a2.	_NegativePropertyAssertion(P, a1, a2)
negative data property assertion	NegativeDataPropertyAssertion(R a v) _:x rdf:type owl: NegativePropertyAssertion. _:x owl:sourceIndividual a. _:x owl:assertionProperty R. _:x owl:targetValue v.	_NegativePropertyAssertion(R, a, v)

Keys

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
Key	HasKey(C (P1 … Pm) (R1 … Rn)) C owl:hasKey (P1 … Pm R1 … Rn). m+n>0	_HasKey(C, [P1,..,Pm, R1 … Rn])

Declarations

Language Feature	OWL2 Functional Syntax RDF Syntax	ObjectLogic Syntax
class	Declaration(Class(CN)) CN rdf:type owl:Class.	CN[].
datatype	Declaration(Datatype(DN)) DN rdf:type rdfs:Datatype.	DN:owl#DatatypeProperty
object property	Declaration(ObjectProperty(PN)) PN rdf:type owl:ObjectProperty.	PN:owl#ObjectProperty
data property	Declaration(DataProperty(R)) R rdf:type owl:DatatypeProperty.	R:owl#DataProperty
annotation property	Declaration(AnnotationProperty (A)) A rdf:type owl:AnnotationProperty.	A:owl#AnnotationProperty
named individual	Declaration(NamedIndividual (aN)) aN rdf:type owl:NamedIndividual.	aN: owl:NamedIndividual

21.2 Throughput Benchmark

The Throughput Benchmark is parameterized by a benchmark configuration file. An example is located in the OntoBroker installation directory:

```
benchmark\benchmark-throughput.xml.
```

Each suite has 2 throughput properties for configuring the benchmark:

```
<suite name="Threads-10" threads="10" rounds="50">
```

The file benchmark.dtd helps to create valid benchmark.xml files, if your xml editor supports dtd validation.

- threads: Number of threads to execute queries. This is the number of clients which execute parallel queries.
- rounds: Number of queries executed by each thread.

In the example above, each of the 10 threads execute 50 queries, which means that 500 queries are executed. The benchmark diagrams show the throughput, which means how many queries are executed per second.

Throughput is the number of queries executed each time. Latency is the time a query needs to run.

Run example

To start the example delivered by the OntoBroker installation, run:

```
OntoBrokerHome> tools\benchmark.cmd benchmark\benchmark-throughput.xml
```

It takes some minutes for the benchmark to finish. To see the report, open:

```
OntoBrokerHome\benchmark\outputDirectory_benchmark-throughput\benchmark-throughput.
html
```

21.3 Range-Restricted Rules

OntoBroker implements negation as the so-called "default negation" (OntoBroker supports both stratified and wellfounded negation). This means that when you have the program:

```
q(a).
q(b).
r(a).
QUERY q1: ?- q(?X) AND NOT r(?X).
```

Then OntoBroker evaluates the rule as follows:

1. Retrieve all tuples for q(X): X=a, X=b
2. Retrieve all tuples for r(X): X=a
3. Subtract the tuples r(X) from q(X).

So the result of the query is "X=b" because b does not occur in r(X) and hence the formula "NOT r(X)" is true.

Imagine what would happen if we had the program:

```
q(a).
q(b).
r(a).
QUERY q1: ?- NOT r(?X).
```

We have nothing as the base where we could subtract the r(X) facts from. So the result would be infinitely large (= the result size is not range restricted).

21.4 Built-ins

In this chapter you find all of the OntoBroker built-ins with their syntax, a short description, their parameters and bindings. In the Bindings-column "b" means that it must be bound and "f" that the argument must be free, usually as a variable.

You can get a complete list of Built-ins if you run the following query:
`?- _builtins(?NAME,?ARITY,?DESCRIPTION,?PARAMETERS).`

Name	Arity	Description	Parameters	Bindings
"_abs"	2	absolute double value	["input","absolute value"]	bf;fb;bb;
"_absDecimal"	2	absolute decimal value	["input","absolute value"]	bf;fb;bb;
"_activeCollaborationUsers"	3	returns active collaboration users and their hostname for a module	["module","user","hostname"]	fff;bff;fbf; bbf;ffb;bfb; fbb;bbb;fff;
"_activeSessions"	4	returns active sessions	["sessionid","type","user","hostname"]	b;
"_add"	3	adds numbers using double arithmetic or durations to date/times	["summand","summand","sum"]	bbf;bfb;fbb; bbb;
"_addDecimal"	3	adds numbers using decimal arithmetic	["summand","summand","sum"]	bf;fb;bb;
"_aggregations"	4	returns all available aggregations	["name","number of input variables","description","input variables"]	
"_append"	3	appends second to first list	["first list","second list","merged list"]	
"_applyStringTemplate"	4	applies string template to the variable bindings	["template","options","varBindings","output"]	bbbf;bbb;

"_bound"	1	Tests if input is bound (i.e. equals NULL value).	["variable"]	
"_builtins"	4	Returns all available built-ins	["name","arity","description","list of parameter names"]	
"_ceiling"	2	Returns the smallest (closest to negative infinity) value that is less than or equal to the argument and is equal to a mathematical integer	["number","mathematical integer (double or decimal)"]	bf;bb;
"_charAt"	3	char at position	["string","position","result"]	bbf;bbb;
"_computeexplicit"	3	transformation plug-ins for mapping tool	["transformation","f(name, var)","result"]	
"_concat"	3	concatenate two strings	["first string","second string","result"]	bbf;bfb;fbb; bbb;
"_concepts"	2	return concepts	["concept","module"]	ff;bf;fb;bb;ff;
"_connectors"	4	returns all available connectors	["name","arity","description","list of parameter names"]	
"_constraintByID"	3	returns constraint text for given ID	["module","constraintID","ruleText"]	bff;bbf;bfb; gbb;
"_constraintContainsTerm"	3	returns aiomxIDs for which the term is contained in the constraint formula	["module","term","constraintID"]	bbf;bbb;
"_contains"	2	first string/list contains second string/list	["string/list","string/list"]	bb;
"_containsN"	2	first string/list contains second string/list	["string/list","string/list"]	bb;
"_containsTerm"	2	is term contained in another term (functional term)	["term","containedTerm"]	bb;
"_currentDateTime"	1	returns the current date time	["currentDatetime"]	f;b;
"_currentUser"	1	returns current login user as IRI (and only if config parameter Security. AccessControl = on)	["user"]	f;b;f;
"_cut"	3	Returns a substring of the string-object truncated by the number of characters specified in the method argument. Truncation takes place on the right.	["content","offset","result"]	bbf;bbb;
"_datatype"	2	Gets the data type IRI of typed literals, xsd:string for untyped literals, fails otherwise.	["value","datatype"]	
"_day"	2	returns day	["date/datetime","day (integer)"]	
"_days"	2	returns days	["duration","days (integer)"]	bf;bb;
"_diffAt"	3	compares two strings and	["content","input","positionOfFirs	

		returns the position of the first not matching character	tDiffence"]	bbf;bbb;
"_difference"	3	Creates difference of two maps	["map1","map2","result map"]	
"_ebv"	1	Generates the effective boolean value.	["value"]	
"_endsWith"	2	first string/list ends with second string/list	["string/list","string/list"]	bb;
"_equals"	2	equals comparison (left == right)	["left","right"]	bb;
"_false"	0	always returns false Example: ?- _'false'. // single quotes needed as false is keyword in ObjectLogic	[]	
"_fill"	4	fill String with n-times of the other string	["baseString","fillString","timesToFill","resultString"]	bbbf;bbbb;
"_filter"	2	filters values	["variable","right"]	bb;
"_filterN"	2	filters values	["variable","right"]	bb;
"_filterdups"	2	stores the list and is true if list is not a duplicate	["variable instantiations","provenance list"]	
"_firstIndexOf"	3	returns first occurance of the second string/item within the first string/list	["content","part","listOfPositions"]	bbf;bbb;
"_flatten"	2	flattens a list of lists into a flat list and eliminates duplicates	["list of lists","flatted list"]	
"_floor"	2	Returns the largest (closest to positive infinity) value that is less than or equal to the argument and is equal to a mathematical integer	["number","mathematical integer (double or decimal)"]	bf;bb;
"_fractionalSecond"	2	returns fractional second	["time/datetime","fractional second (decimal or null)"]	bf;bb;
"_fragment"	2	returns fragment of IRI	["iri","fragment string"]	bf;bb;
"_generateExplanationRuleText"	4	generates explanation rule text for given ruleId and template	["module","ruleId","template","output"]	bbbf;bbbb;
"_generateExplanati	4	generates explanation	["module","ruleId","templategro	

onTemplate"		template for given ruleId	up","output"]	bbbf;bbbb;
"_geoDistance"	3	Calculation of distance between two geographic coordinates in kilometer.	["coordinate1 [geoCoordinate]","coordinate2 [geoCoordinate]","distance in kilometer [double]"]	
"_greaterOrEquals"	2	greater than or equals comparison (left >= right)	["left","right"]	bb;
"_greaterOrEqualsN"	2	greater than or equals comparison (left >= right)	["left","right"]	bb;
"_greaterThan"	2	greater than comparison (left > right)	["left","right"]	bb;
"_greaterThanN"	2	greater than comparison (left > right)	["left","right"]	bb,
"_hasForeignRole"	1	checks if current user has given foreign role. Roles from the SecurityModuleAuthorizationRealm are not checked. This is only relevant if access control is enabled.	["role"]	b;
"_host"	2	returns host of IRI	["iri","host string"]	bf;bb;
"_hour"	2	returns hour	["time/datetime","hour (integer)"]	
"_hours"	2	returns hours	["duration","hours (integer)"]	bf;bb;
"_indexOf"	3	returns sorted list of offsets from the second string within the first	["content","part","listOfPositions"]	bbf;bbb;
"_info"	2	Retrieve information about OntoBroker. Currently \version\ and \fullVersion\ are supported	["key","info"]	
"_integer2iri"	3	converts between integer and IRI	["integer","iri prefix","iri"]	
"_intersection"	3	Creates intersection of two maps	["map1","map2","result map"]	
"_isBlank"	1	Tests if input is a blank node.	["value"]	
"_isConstraint"	2	true if given id is a constraint	["module","constraintID"]	bf;bb;
"_isCustomPermitted"	2	checks if current user has custom permission for module. This is only relevant	["module","customPermission"]	

		is access control is enabled.		
"_isDatatype"	1	checks if object is a data type	["typename"]	b;
"_isIRI"	1	Tests if input is an IRI.	["value"]	
"_isLiteral"	1	Tests if input is a RDF literal.	["value"]	
"_isModulePermitted"	2	checks if current user has access right for module. This is only relevant is access control is enabled.	["module","accessRight"]	bb;
"_isNull"	1	checks if object is the null value	["term"]	b;
"_isPermitted"	1	checks if current user has given permission. This is only relevant is access control is enabled.	["permission"]	b;
"_isPropertyPermitted"	2	checks if current user has access right for property. This is only relevant is access control is enabled.	["property","accessRight"]	
"_isQuery"	2	true if given id is a query	["module","queryID"]	bf;bb;
"_isRule"	2	true if given id is a rule	["module","ruleID"]	bf;bb;
"_isTypeOf"	2	checks if object is of given type	["typename","object"]	fb;bb;
"_isequal"	2	X is equal Y	["X","Y"]	bb;
"_lang"	2	Gets the language of a RDF literal, if not available it returns an empty plain literal.	["literal","language"]	
"_langMatches"	2	Language range matching as defined in RFC4647 section 3.3.1.	["language","filter"]	
"_language"	2	Returns language part of plain literal	["plain literal","language"]	bf;bb;
"_lastIndexOf"	3	returns last occurance of the second string/item within the first string/list	["content","part","listOfPositions"]	bbf;bbb;
"_latitude"	2	Get the latitude of a geographical coordinate.	["coordinate [geoCoordinate]","latitude [double]"]	
"_length"	2	returns the length of a string or list	["string/list","length"]	

				bf;bb;
"_lessOrEquals"	2	less than or equals comparison (left <= right)	["left","right"]	bb;
"_lessOrEqualsN"	2	less than or equals comparison (left <= right)	["left","right"]	bb;
"_lessThan"	2	less than comparison (left < right)	["left","right"]	bb;
"_lessThanN"	2	less than comparison (left < right)	["left","right"]	bb;
"_localName"	2	returns local name of IRI	["iri","local name"]	bf;bb;
"_loginUser"	2	returns login user and hostname	["user","hostname"]	ff;bf;fb;bb;ff;
"_longitude"	2	Get the longitude of a geographical coordinate.	["coordinate [geoCoordinate]","longitude [double]"]	
"_map2table"	4	Converts maps to flat table	["map","path string","position","result value"]	
"_match"	2	Y matches X	["X","Y"]	
"_match"	3	The string in the first argument is matched with the regular expression in the second argument. The result (third argument) of the method is a list of terms of the form match (matchingPosition, matchingSubstring), which represents matches inside the object-string. The list is sorted in the ascending order of positions of the matches. The counting of positions in the object string starts with 0.	["content","input","variable"]	bbf;bbb;
"_member"	2	_list[*=> _member(_object)]. Binding: bb. The method's argument and the list-object may not be fully ground. In this case, the method succeeds, if the argument to the method unifies with a member of the list.	["list","object"]	bf;bb;
"_memberAt"	3	The value of the method is	["list","index","value"]	

		the object that occurs in the list object at the position given by the argument to the method		bff;bbf;bfb; bbb;
"_memberByPath"	3	Filters with xpath expression	["map","path string","result value"]	
"_metrics"	1	returns metrics about the ontology manager	["metrics"]	f;b;f;
"_minute"	2	returns minute	["time/datetime","minute (integer)"]	
"_minutes"	2	returns minutes	["duration","minutes (integer)"]	bf;bb;
"_modules"	1	returns the available modules	["modules"]	f;b;f;
"_month"	2	returns month	["date/datetime","month (integer)"]	
"_months"	2	returns months	["duration","months (integer)"]	bf;bb;
"_multiply"	3	multiplies numbers using double arithmetic	["factor","factor","product"]	bbf;bfb;fbb; bbb;
"_multiplyDecimal"	3	multiplies numbers using decimal arithmetic	["factor","factor","product"]	bbf;bfb;fbb; bbb;
"_namespace"	2	returns namespace of IRI	["iri","namespace"]	bf;bb;
"_negate"	2	negates a duration	["duration","duration"]	bf; fb; bb;
"_newInstance"	5	creates a new instance of the given classname and assigns instance reference	["module","classname","object","children","instance"]	bbbbf; bbbbb;
"_normalize"	2	Normalize order of key/value pairs for _map input or normalize to UTC timezone for _dateTime, _date or _time input.	["input","normalized"]	bf;
"_normalizeWith"	3	Converts the years and months fields into the days field by using a specific dateTime/date instant as the	["input (duration)","reference point (date or datetime)","result (duration)"]	bbf; bbb;

		reference point		
"_numberToNumber"	4	Converts between floating-point and fixed-point numbers.	["Type URI 1","Type URI 2","Numeric Value 1","Numeric Value 2"]	
"_overlappedBy"	3	True if a suffix of the object string is also a prefix of the string given in the method's argument. The result of the method is the number of characters in the maximal overlap.	["suffixString","prefixstring","numberOfOverlappedChars"]	bbf;bbb;
"_path"	2	returns path of IRI	["iri","path string"]	bf;bb;
"_port"	2	returns explizit port of IRI	["iri","port integer"]	bf;bb;
"_prefices"	3	Returns prefix/namespace definitions per module	["module","prefix","namespace"]	bff;bbf;bfb;bbb;
"_properties"	2	return properties	["property","module"]	
"_query"	2	returns query of IRI	["iri","query string"]	
"_queryByID"	3	returns query text for given ID	["module","queryID","ruleText"]	bff;bbf;bfb;bbb;
"_queryContainsTerm"	3	returns queryIDs for which the term is contained in the query formula	["module","term","queryID"]	bbf;bbb;
"_queryInfo"	4	returns query text and additional information for given ID	["module","queryID","ruleText","info"]	
"_regex"	3	Matches against a regular expression.	["value","regular expression","options"]	
"_replace"	4	Replaces every occurrence of the regular expression inside the object string with the string given in the second argument of the method. The regular expression is given in the first argument.	["input string","regexp","input","result"]	bbbf;bbbb;
"_reverse"	2	reverses list/string	["list/string","reversed list/string"]	bf;fb;bb;
"_rootconcepts"	2	use the concept hierarchy to determine the root concepts	["concept","module"]	ff;bf;fb;bb;ff;
"_round"	2	rounds number	["number","rounded number"]	bf;bb;
"_ruleByID"	3	returns rule text for given ID	["module","ruleID","ruleText"]	bff;bbf;bfb;bbb;
"_ruleContainsTerm	4	returns ruleIDs for which the	["module","term","isHead","ruleI	

"		term is contained either in rule head or body	D"]	bbff;bbbf; bbfb;bbbb;
"_ruleID2ExplanationRuleID"	2	returns transforms ruleID to/ from ruleID of its explanation rule	["ruleID","ruleExplanationRuleID"]	
"_ruleoutlogger"	1	prints input to console	["input"]	
"_scheme"	2	returns scheme of IRI	["iri","scheme string"]	bf;bb;
"_second"	2	returns second	["time/datetime","second (integer)"]	
"_seconds"	2	returns seconds	["duration","seconds (integer)"]	bf;bb;
"_sessionInfo"	1	returns information about session	["infoMap"]	
"_sign"	2	returns sign	["duration","sign (integer)"]	bf;bb;
"_sort"	2	Returns sorted list	["list","sorted list"]	bf;bb;
"_split"	3	splits before the first character of regexp	["input string","regexp","variable"]	
"_startsWith"	2	first string/list starts with second string/list	["string/list","string/list"]	bb;
"_startsWithN"	2	first string/list starts with second string/list	["string/list","string/list"]	bb;
"_statisticsIndex"	2	provides some statistics about an index	["module","statistics text"]	bf;
"_statusIndex"	2	provides up-to-date status for index	["module","status text"]	bf;bb;
"_str"	2	Returns the lexical form of a literal or the codepoint representation of an IRI.	["value","lexical form"]	
"_stringToConstant"	3	converts a lexical value to a constant and backwards	["typename","lexical value","constant"]	
"_subset"	2	true if first list contains all items of second list	["first list","second list"]	bb;

"_substring"	4	returns String.substring(start, end)	["content","start","end","substring"]	bbff;bbbf; bbfb;bbbb;
"_textValue"	2	Returns text part of plain literal	["plain literal","text string"]	
"_timeInMilliseconds"	2	returns time in milliseconds since January 1, 1970 (Gregorian), midnight UTC	["datetime","milliseconds since January 1, 1970 (Gregorian), midnight UTC"]	bf;
"_toLower"	2	lower case of string	["string","lower string"]	bf;bb;
"_toObjectLogicString"	2	returns ObjectLogic representation	["object","ObjectLogic representation"]	bf;
"_toString"	2	returns string representation	["object","string representation"]	bf;bb;
"_toType"	3	returns new instance for given parameter	["typename","param1","object"]	bbf;bbb;
"_toType"	4	returns new instance for given parameters	["typename","param1","param2", "object"]	bbbb; bbbf; bffb;bbfb; bfbb
"_toType"	5	returns new instance for given parameters	["typename","param1","param2", "param3","object"]	bbff;bbbbf; bbbfb; bbbbb;
"_toType"	7	returns new instance for given parameters	["typename","param1","param2", "param3","param4","param5","object"]	bbbbbbf; bbbbbbb;
"_toType"	8	returns new instance for given parameters	["typename","param1","param2", "param3","param4","param5","param6","object"]	bbbbbbbf; bbbbbbbb;
"_toType"	9	returns new instance for given parameters	["typename","param1","param2", "param3","param4","param5","param6","param7","object"]	bbbbbbbbf; bbbbbbbbb;
"_toType"	10	returns new instance for given parameters	["typename","param1","param2", "param3","param4","param5","param6","param7","param8","object"]	bbbbbbbbbf; bbbbbbbbb;
"_toUpper"	2	upper-case of string	["string","upper string"]	bf;bb;
"_tokenize"	3	Breaks the object-string into a	["input string","delim","token"]	

		list of tokens at every occurrence of any delimiter character given in the argument to the method (this argument is a string of delimeter characters). The delimiter characters are not retained as part of the tokens. For instance, 'xyzw. foo'[_tokenize('y.') -> [x,zw, foo]]		bbf;bbb;
"_tokenizeRegExp"	3	splits before the first character of regexp	["input string","regexp","token"]	bbf;bbb;
"_totalOrderLessThan"	2	less than comparison (left < right) providing a total ordering for all terms	["left","right"]	bb;
"_trim"	3	Deletes from the object string all the leading and trailing characters specified in the argument to the method	["string","characters","result"]	bbf;bbb
"_true"	0	always returns true ?- _'true'. // single quotes needed as true is keyword in ObjectLogic	[]	
"_typeName"	2	returns datatype of input	["object","typename"]	bf;bb;
"_unify"	2	Unifies first with second parameter	["left","right"]	bf;fb;bb;ff;
"_union"	3	Creates union of two maps	["map1","map2","result map"]	
"_user"	2	returns user of IRI	["iri","user string"]	bf;bb;
"_workingDaysBetween"	3	Calculates the number of working days between two dates. Public holidays are not considered.	["startDate","endDate","durationInDays"]	bbf (calculate days) bfb (calculate end date) bbb (validate) fbb (calculate start date)
"_write"	1	prints parameters	["X1"]	

				b;
"_write"	2	prints parameters	["X1","X2"]	b;b;
"_write"	3	prints parameters	["X1","X2","X3"]	b;b;b;
"_write"	4	prints parameters	["X1","X2","X3","X4"]	b;b;b;b;
"_write"	5	prints parameters	["X1","X2","X3","X4","X5"]	b;b;b;b;b;
"_write"	6	prints parameters	["X1","X2","X3","X4","X5","X6"]	b;b;b;b;b;b;
"_year"	2	returns year	["date/datetime","year (integer)"]	
"_years"	2	returns years	["duration","years (integer)"]	bf;bb;
"_zoneHour"	2	returns zone hour (with sign)	["time/datetime","zone hour"]	bf;bb;
"_zoneMinute"	2	returns zone minute (with sign)	["time/datetime","zone minute"]	bf;bb;

21.4.1 Information about _queryIndex/10

The queryIndex/10 built-in executes a search in a Lucene full-text index. Typically this index is an automatically managed index of the OntoBroker, but external indexes can also be queried. The built-in has 10 arguments, the first five must be bound.

Argument	Bound/ Free	Description
<module>	b	The module whose index should be queried
<option list>	b	List of optional parameter (see list below for details)
<lucene query text>	b	The query string (Lucene query syntax) see http://lucene.apache.org/java/2_9_3/queryparsersyntax.html for details about the query syntax.
<offset>	b	Index of first hit to return (starts with 0)
<limit>	b	Maximal number of hits to return
<object>	f	Term for object hit
<total count>	f	Total count of hits
<score>	f	Lucene ranking for the hit
<order>	f	Order number to sort the hits in the correct order
<optional output list>	f	Contents depends on <option list>

The <option list> parameter consists of a list of optional parameter. If no optional parameters should be specified, use the empty list, i.e. []

21.4.1.1 Supported optional parameter for <option list> argument

Parameter	Description
return(<field>)	The content of the field is returned for the hit in the <optional output list> variable. Note that the field must be defined as "stored" in the Lucene index, otherwise nothing is returned. The content of the field is returned for the hit in the <optional output list> variable. Note that the field must be defined as "stored" in the Lucene index, otherwise nothing is returned. Example: `?- _queryIndex(module1, [return("name_en")], "name_en:foo", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).` For every "name_en" field of a hit document in the Lucene index, the <optional output list> will contain an item name_en("content of field")
stringmetric(<metric>)	Sets the string metric to be used for the fuzzy search. If this option is not set, the default value is used, either explicitly specified by the property "defaultStringMetric" in the fulltextindex-config.xml or if this is also not set, the string metric "Jaro" is used. Supported string metric values are: "Levenstein", "MongeElkan", "NeedlemanWunch", "QGrams", "Jaro", "JaroScaled", "JaroWinkler", "DamerauLevenshtein", "DamerauLevenshteinScaled", "MaxJaroDamerauLevenshteinScaled", "DamerauLevenshteinSoundex", "Jaccard", "Soundex", "SmithWaterman" Side remark: You can use the built-in distance2 to see how two strings compare using one of these string metrics, e.g. `?- _distance2("Jaro", "good", "food", 0, ?X).`

	?X will return a similarity value (between 0 and 1.0), here 0.833 If you perform a fuzzy search with the string metric Jaro, e.g. Lucene query text "good~0.8", this will match "food", as 0.833 is >= 0.8 Example: ?- _queryIndex(module1, [stringmetric("Jaro")], "name_en:good~0.8", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt). Use string metric "Jaro" for fuzzy search.
`includeall`	Includes all imported modules of <module> (first argument) in the search. Example: `?- _queryIndex(module1, [includeAll], "foo", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).`
`defaultfield (<field>)`	Sets the default field to be used for search terms whose field is not explicitly given. E.g. if you have the Lucene query text "all:foo bar", the search term "foo" is searched in the field "all" and bar is searched in the default field. If the default field is not set in the option, the default field specified in the fulltextindex-schema.xml (tag defaultSearchField) is used. If this is also not set, the default field is "all". Example: `?- _queryIndex(module1, [defaultfield("name_en")], "foo", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).` Use field "name_en" as default field
`solrparam(<name>, <value>)`	Sets additional Apache Solr parameters. The queryIndex built-in uses also the core of Apache Solr on top of Lucene. With this option you can set one or multiple parameters for this layer. Example: `?- _queryIndex(module1, [solrparam("hl", "true"), solrparam("hl.fl","name_en"), solrparam("hl.snippets", "2"), solrparam("hl.fragsize", "200")")], "foo", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).` These parameters enable the Solr highlighting. Please note that only stored fields can be used for highlighting. More details about the Solr parameter for highlighting can be found here: http://wiki.apache.org/solr/HighlightingParameters
`externalindexesonly`	If this option is set, the module in the first argument is ignored. Note that in this case the option externalindex(<path>) must be set.
`externalindex (<path>)`	Adds one or multiple external Lucene indexes for the search. Note that the used fields must nonetheless be defined in the fulltextindex-config.xml. Example: `?- _queryIndex(dummy, [externalindexesonly, externalindex("d:/index1"),externalindex("d:/index2")], "foo", 0, 10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).` Includes the Lucene indexes located in the directory d:\index1 and d:\index2

21.4.1.2 Extended syntax for <Lucene query text> argument

You can use the Solr query syntax extensions in the <Lucene query text> argument. This allows using customized query parsers to add new functionality to the search. A customized query parser is specified by starting the query text with “{!parsername param1=value1 param2=value2}”. Here parsername is the name of the query parser, param1, value1, param2, value2 are sample parameter/value pairs.

OntoBroker currently supports two extended query parsers: lucene and multifield

lucene

This is a normal Lucene query which some additional parameters specified directly in the query text.

Parameter	Description
q.op	Default operator (either AND or OR). The standard default operator is AND
df	Default field (see above)
stringMetric	String metric (see above)
sort	Sort results, e.g. sort='id desc' Important restriction: Sorting can be done on the "score" of the document, or on any multiValued="false" indexed="true" field provided that field is either non-tokenized (ie: has no Analyzer) or uses an Analyzer that only produces a single Term (ie: uses the KeywordTokenizer) See for more details: http://wiki.apache.org/solr/CommonQueryParameters

Example:

```
?- _queryIndex(module1, [], "{!lucene df=name_en q.op=OR sort='id asc'} foo bar", 0,
10, ?Obj, ?Tc, ?Sc, ?Order, ?Opt).
```

Queries for "foo OR bar" in the default field "name_en" and sorting the results ascending by the field id.

multifield

This query parser searches for the search terms in multiple default fields. It supports the same parameters as the lucene query parser plus additionally:

Parameter	Description
fields	Fields to search, e.g. fields='name_en^2 docu_en'. Here a hit in the field name_en is boosted additionally by a factor 2

Example:

```
?- _queryIndex(module1, [return(name_en),return(docu_en)], "{!multifield
fields='name_en docu_en' q.op=OR sort='id asc'} foo bar", 0, 10, ?Obj, ?Tc, ?Sc, ?
Order, ?Opt).
```

Queries for "foo OR bar" in the fields "name_en" and "docu_en" and sorting the results ascending by the field id and returns fields "name_en" and "docu_en".

21.4.1.3 Available fields for objects in modules

If the fulltext indexing is enabled, OntoBroker creates index entries for every ObjectLogic object which is used in the given module as concept, instance, attribute or relation. This means hits are always to the indexed ObjectLogic object, whose term is returned in the <object> parameter.

Side remark

Fulltext indexing is enabled by the OntoConfig.prp parameter, e.g.

```
FullTextIndex  = on
```

The fields in the full-text index are defined in the fulltextindex-config.xml and fulltextindex-schema.xml. (see section “Fulltext indexer settings” in the OntoBroker Manual Appendix for details).

As a default, the following fields are filled for every object:

Field	Stored	Indexed	Description
id	yes	yes (untokenized)	This field stores the untokenized ObjectLogic term representation of the object.
lid	yes	yes	Field for indexing the localname of the ObjectLogic object term (for terms which are not IRI this is the same as the id)
type	yes	yes	This field contains the types of the object: i = Instance c = Concept a = Attribute specification r = Relation specification p = Property specification u = Rule q = Query t = Constraint
assertedisa	yes	yes	For instances this field contains the ids of its concepts
repr_de repr_en …	yes	yes	Contains the language-dependent label for a given object
docu_de docu_en …	no	yes	Contains the language-dependent documentation for a given object
syn_de syn_en …	yes	yes	Contains language-dependent synonyms
name_de name_en …	yes	yes	This field contains the label and the synonyms in the given language. By default the indexer only creates fields for the languages “de” and “en”. If this field is returned (e.g. <option list> = […,return(name_en),…]), the first line always contains the label.
syn	yes	yes	Contains all synonyms for all languages
all	no	yes	Contains all text of the fields lid, name_{lang}, docu_{lang}, attval, syn
axiomtext	yes	yes	Contains the rule text for rules, queries and constraints.

All fields which are indexed can be used in the query text. For all fields which are stored a “return” option can be specified.

Example:

```
?- _queryIndex(<http://company.com#onto1>,
[return(name_en),return(type),includeall], "+name_en:city
+type:i +assertedisa:\"<http://company.com#Region>\"", 0, 20,
?OBJ,?TC,?SCORE,?ORDER,?OPT).
```

This query searches for instances of <http://company.com#Region> in the module <http://company.com#onto1> whose English representation or synonym contains the word “city”.

Here are some more examples for valid Lucene query text:

all:city	searches in the field “all” for the word “city”
city	same as “all:city”
+name_en:village +type:i	searches for instances whose English representation or synonym contains the word “village”
id:"http://company.com#Project"	Searches for the object with the id <http://company.com#Project>

21.4.2 ObjectLogic Built-ins

The descriptions of all of the Built-ins include *arity, signatures* (which indicate the type of the arguments), *bindings* (where f means that the argument must be free and b that it must be bound), plain English *description,* and *examples.* The abbreviations for the various types mentioned in the signatures are described in the following chapter.

21.4.2.1 Comparisons

Comparison	Arity	Signatures	Bindings	Description	Examples
==	2	various datatypes possible	bb, bf, fb, ff	True if argument 2 can be obtained from argument 1 by variable renaming.	?X==?Y =
=	2	various datatypes possible	ff, fb, bf, bb	Argument 1 unifies with argument 2.	?X=?Y ?X=5 !=
!=	2	various datatypes possible	ff, fb, bf, bb	Argument 1 does not unify with argument 2.	?X!=?Y

21.4.2.2 Basic Mathematical Built-ins

Math. Built-in	Arity	Signatures	Bindings	Description	Example
_multiply	3	=>(_number, _number, _number) The arguments can be any number of a supported numeric data type. Conversion rules apply.	fbb, bfb, bbf	Multiplies the numbers in arguments 1 and 3. Binds the result to argument 3.	_multiply(2,3,?X)
_add	3	=>(_number, _number, _number) The arguments can be any number of a supported numeric data type. Conversion rules apply.	fbb, bfb, bbf, bbb	Adds the numbers in arguments 1 and 2. Binds argument 3 to the result.	_add(1,?X,5)
_abs	2	=>(_number, _number, _number) The arguments can be any number of a supported numeric data type. Conversion rules apply.	bf, fb, bb	Binds argument 2 to the absolute value of argument 1.	_abs(-1,?X)

21.4.2.3 Basic Mathematical Functions

These are basic mathematical functions are available in is-statements, e.g. ?X is (?Y+3/?Z+5)*sin(?Z) which means that the value of the expression on the right is assigned to variable ?X. Note that functions are not prefixed with an underscore '_'.

mod will just be available for long and int. sin, cos, tan, asin, acos, atan, exp, pow, log and sqrt will just be availabe for double.

Math. Function	Arity	Signatures	Bindings	Description	Example
+	1	_number	b	Multiplies argument by +1 and returns the result.	+2
-	1	_number	b	Multiplies argument 1 by -1 and returns the result.	-2
+	2	(_number, _number),	bb	Adds the two numbers,	3+2

		(_string,string), (_dateTime, _duration,), (_time, _duration,), (_date, _duration)		strings, or dates/times with duration, and returns the result.	
-	2	(_number, _number,), (_dateTime, _duration,), (_time, _duration,), (_date, _duration)	bb	Subtracts the two numbers or dates/times with duration, and returns the result.	3-2
*	2	_number, _number	bb	Multiplies the two numbers given in arguments 1 and 2 and returns the result.	3*2
/	2	_number, _number	bb	Divides the two numbers given in arguments 1 and 2 and returns the result.	3.2 / 1.1
modmo	2	_integer, _integer	bb	Computes the remainder of the division of the two integers given in arguments 1 and 2 and returns the result.	5 mod 2
abs	1	_number	b	Returns the absolute value of the argument.	abs(-1)
max	2	_number, _number	bb	Returns the greater of the values of arguments 1 and 2.	max(1,2)
min	2	_number, _number	bb	Returns the smaller of the values of arguments 1 and 2.	min(1,2)
round	1	_number	b	Returns the closest integer to the value of argument 1.	round(2.4)
ceil	1	_number	b	Returns the smallest _long	ceil(2.3)

				or _integer value that is not smaller than the value in argument 1. _integer is returned if the argument is _integer or _decimal. Otherwise, _long is returned.	
floor	1	_number	b	Returns the largest _long or _integer value that is not smaller than the value in argument 1. _integer is returned if the argument is _integer or _decimal. Otherwise, _long is returned.	floor(2.3)
tan	1	_double	b	Returns the tangent of the value in argument 1.	tan(0.8)
atan	1	_double	b	Returns the arctangent of the value in argument 1.	atan(8)
sin	1	_double	b	Returns the sine of the value in argument 1.	sin(5)
asin	1	_double	b	Returns the arcsine of the value in argument 1.	asin(0.5)
cos	1	_double	b	Returns the cosine value of the value in argument 1.	cos(8)
acos	1	_double	b	Returns the arccosine value of the value in argument 1.	acos(0.8)
exp	1	_number	b	Returns the Euler's number raised to the power of the value given in argument 1.	exp(3)
sqrt	1	_number	b	Returns the	exp(3)

				positive square root of a positive number.	
log	1	_number, _number	b	Returns natural the logarithm of the value in argument 1.	log(1)
pow	3	_number, _number	bb	Raises argument 1 to the power given in argument 2.	pow(2,3)

from the copyright of the web page http://forum.projects.semwebcentral.org/forum-syntax.html

21.4.3 General Syntactic Changes

In the years since the publication of the original paper on F-logic in 1995, new ideas have been proposed and a number of systems based on F-logic have been developed. This forum attempts to organize these ideas and propose a unifying syntax and semantics that incorporates the best ideas. It is expected that the various systems, such as FLORA-2 and OntoBroker, and the related specifications, such as WSML and SWSL, will be largely compatible with this syntax.

Basic syntax

The original F-logic distinguished between functional and set-valued attributes and methods. In this forum we decided to abandon this distinction and replace it with a more general mechanism of cardinality constraints. We will use only set-valued attributes and eliminate the double-arrows —>> and =>>. The arrows -> and => will be used for everything. The functional property of the attributes will be expressed using cardinality constraints. Instead of using ; to separate the different parts in an F-logic molecule, use the comma or and. The semicolon or or will be used for disjunctions in the rule body.

Cardinality constraints on attributes and methods

Syntax:

```
a[foo {2:3} => type].
```

The symbol * in the upper bound part is used for infinity. The constraint {0:1} enforces the functional property and hence functional methods can be expressed using this type of constraints.

OWL-like properties

The above syntax will be extended to also accommodate OWL-like properties of F-logic attributes: symmetric and inverseOf(Attr). For instance:

```
Person[marriedTo{1:1,symmetric}=>Person], Person[childOf{0:2,inverseOf(parent)}=>
Person]
```

Unlike the cardinality specification, symmetric and inverseOf(...) are not constraints. Instead, they are treated as a shortcut that avoids the need to specify a rule explicitly. For instance:

```
?P1[marriedTo -> ?P2] :- ?P2:Person[marriedTo -> ?P1:Person].
```

General constraints

For more general constraints, we will use the syntax

```
!- constraintBody
```

rather than ?- constraintBody. It will be up to the implementation to decide when constraints are invoked: when the file is loaded or when an update occurs. A constraint will be said to be violated if posing constraintBody as a query yields a solution.

Variables, quantification, and Lloyd-Topor extensions

a) All variables are denoted as ?Var

Uppercasing (as in Prolog) will not be used, so uppercase identifiers will be permitted without the quotes. The name of a variable can be any sequence of alphanumeric symbols plus the underscore _.

The variable is anonymous. As in Prolog, each denote a unique new variable that does not occur anywhere else.

b) Explicit quantification

Explicit quantification in the rule body is allowed to support the Lloyd-Topor extensions. Both universal (forall) and existential (exist) quantification is permitted.

c) Implicit quantification

All variables that are not quantified explicitly are assumed to be quantified implicitly outside of the rule with the forall quantifier. No explicit universal quantifier for the whole rule will be needed and will not be supported.

d) If-then

Allow the if-then-else syntax as well as the a<--b and a-->b alternatives for if-then.

Rule labels and other metadata

Every rule and query can be preceded by a metadata annotation of the form

```
@{metaannotation} rule/query/constraint
```

The annotation has the form:

```
metaannotation ::= '@{' id | frames | id ',' frames '}'
frames ::= frame (',' frame)*)?
```

Id is a constant, which represents the rule/query/constraint label. It should be unique within the rule set. The frames in the conjunction are intended to represent metadata associated with the rule/query/constraint to which the annotation is attached.

There is a predefined meta data object options[...] to control the execution and output behaviour of a query. It can contain following options (specified as Boolean methods):

Option	Description
`sort(asc(?X),desc(?Y),...)`	Sort the output of the query in the ascending order of bindings for ?X, descending order of bindings for ?Y, etc. The asc(...) specification is implied and can be left out. For instance, sort(?X,desc(?Y),...) is the same as sort(asc(?X),desc(?Y),...).
`instance, sort(?X,desc(?Y),...) is the same as sort(asc(?X),desc(?Y),...).`	The output to the query should be a set of tuples (?Y,?X), i.e., the bindings for ?Y go first.
`maxanswers(NUMBER)`	Specifies that only (at most) the first NUMBER of answers needs to be computed.
`maxtime(NUMBER)`	Stop the computation after this NUMBER of seconds.

Examples:

```
@{rule12}
bar :- foo.
```

```
@{rule15,rule15[author->Bob, date-> "1979-09-13T17:31:41-02:30"^^_dateTime]}
foo :- bar.
```

```
@{options[outorder(?Y)]}
?- ?X[?Y->?Z].
```

```
@{query13,
  query13[author->Martin],
  options[sort(asc(?X),desc(?Y)),outorder(?X,?Y),maxnumber(1)]}
?- ?X[name->?Y].
```

Methods with arguments, HiLog syntax

The original F-logic used the @-syntax to separate the method name from the method arguments (e.g., o[m@(a,b)->v]). The purpose of this separation was to allow variables over the method names. In this forum it was decided to abandon the @-syntax and use the HiLog syntax instead.

Structural inheritance

We will use *=> for inheritable types and => for the non-inheritable ones with the following inference rules:

```
C2::C1, C1[M *=> T] |= C2[M *=> T]
O::C, C[M *=> T] |= O[M => T]
```

For data-molecules, both -> and *-> are allowed. In the basic case, they will have the same semantics, except that => will specify the type for -> and *=> the type for *->.

Builtin predicates and functions

These are described in detail in a separate document[243].

Aggregates

This forum adopted the syntax for aggregates proposed in FLORA-2:

```
?Var = count{?Var [GroupingVarList] | Query}
?Var = sum{?Var [GroupingVarList] | Query}
?Var = avg{?Var [GroupingVarList] | Query}
?Var = max{?Var [GroupingVarList] | Query}
?Var = min{?Var [GroupingVarList] | Query}
?Var = collectset{?Var [GroupingVarList] | Query}
?Var = collectbag{?Var [GroupingVarList] | Query}
```

Modules

Modules are described in a separate document.

Name space prefixes and related issues.

To simplify the job of writing IRIs, RDF has proposed a syntax of the form prefix:localPart, which was adopted from the syntax of QNames (qualified names) in XML. Unfortunately, the meaning of such a construct in RDF is different from its meaning in XML and the issue is further confused by the fact that RDF also calls their construct a QNname. In RDF, prefix is a macro that expands to a IRI and prefix:localPart denotes a IRI that is constructed by concatenating the macro-expansion of prefix with localPart. In XML, prefix:localPart does not denote a IRI. Instead, it is a pair of the form (macro-expansion of prefix, localPart). To avoid this confusion, we will call the RDF construct an sQName (serialized QName) and use the syntax prefix#localPart.

Prefixes can be declared as follows:

```
:- prefix prefix-name = "IRI".
```

The syntax for full IRIs is described in the Data Types[250] document.

We will also support default IRI prefixes with the statements of the form

```
:- default prefix "IRI".
```

The default prefix affects IRI specified as localName (without the prefix name).

Anonymous resources

Will adopt the FLORA-2 semantics and syntax for anonymous oids. This has also been adopted by WSML and SWSL. According to this proposal, _# means a completely new identifier (each occurrence) and numbered versions _#1, _#2, ... mean completely new identifiers except that within the same clause different occurrences of the same anonymous oid (e.g., _#2) denote the same identifier.

Data types

Data types will be represented as terms, not strings. A complete list of base data types is found in a separate document.[250]

21.4.4 Data Types

Primitive data types are represented by constants of special form: _datatypeName"literal". The underscore _ and the quotes are part of the syntax. datatypeName is the name of the data type and literal is the value-part of the constant. The data type names and their attributes are listed below. The supported data types closely correspond to XML Schema data types.

Note that the form of the data type constants makes no assumption about the actual internal representation of these constants. A variety of options is possible. The compiler can translate such constants to function terms or to external calls to Java or C programs. For instance, a constant of the time data type such as "12:33:44"^^_time can be compiled into a term like $#%&time(12,33,44) or into $#%&time(12,33,44,45224) (45224 is the number of seconds in the time literal and can be used for comparison with other literals) or into Time#@!([49,50,51,51,52,52]) (here the numbers in the list are the ASCII codes of the digits in the date).

We also define syntax for variables that can be bound only to constants of the corresponding data types. The syntax is ?variableName^^datatypeName, where ^^ and ? are part of the syntax, while datatypeName is the name of a data type. For instance, ?Y^^_time.

Some data types, like time, dateTime, etc., are exact analogues of the corresponding XML Schema types. In this case, their names will be denoted using strings that have the form of a IRI. For instance, 'http://www.w3.org/2001/XMLSchema#time'. However, for convenience, all type names will have one or more F-logic specific abbreviated forms, such as _time or _t. These abbreviated forms are case-insensitive. So, _time and _TiMe are assumed to be equivalent. In addition, when the type names have the form of an IRI, the compact prefix representation is supported. For instance, if xsd is a prefix name for 'http://www.w3.org/2001/XMLSchema#' then "12:33:55"^^<http://www.w3.org/2001/XMLSchema#time> can be written as "12:33:55"^^xsd#time'.

The methods that are applicable to each particular primitive type vary from type to type. However, certain methods are more or less common:

- `_toString`, which applies to a data type constant and returns its printable representation. For instance, if ?Y is bound to "12:44:23"^^_time then ?Y[_toString->'12:44:23'] will be true.
- `_toType( parameters )`, which applies to the any class corresponding to a primitive data type (for instance, _time). Most types will have two versions of this method. One will apply to arguments that represent the components of a data type. For instance, _time[_toType(12,23,45)->"12:23:45"^^_time]. The other will apply to the string representation of the data type. For instance, _time[_toType('12:23:45')->"12:23:45"^^_time].
- `_isTypeOf( constant )`, which applies to every data type class (e.g., _time) and determines whether constant has the given primitive type (_time in this example).
- `_equal( constant )`, which tells when the given datatype constant equals some other term.
- `_lessThan( constant )`, which tells when one constant is less than some other terms. For integers, floats, time, dates, durations, and strings, this method corresponds to the natural order on these types. For other types this method returns false.
- `_typeName`, which tells the type name (and thus also class) of the given data type.

21.4.4.1 Supported Data Types

	_dateTime
Explanation	XML Schema dateTime type
Normal form	"ZYYYY-MM-DDTHH:MM:SS.sZHH:MM"^^_dateTime. The symbols -, :, T, and . are part of the syntax. The leftmost Z is optional sign (-). The part that starts with the second Z is optional and represents the time zone (the second Z is a sign, which can be either + or -; note that the first Z can be only the minus sign or nothing). The part that starts with T is also optional; it represents the time of the specified day. The part of the time component of the form .s represents fractions of the second. Here s can be any positive integer.
Class	_dateTime
Synonyms	_dt, <http://www.w3.org/2001/XMLSchema#dateTime>.
Class methods	• _dateTime[_toType(_integer,_integer,_integer,_integer,_integer,_integer,_decimal, _integer,_integer,_integer) => _dateTime] • _dateTime[_toType(_integer,_integer,_integer,_integer,_integer,_decimal,_integer, _integer) => _dateTime] The meaning of the arguments is as follows (in that order): date sign, year, month, day, hour, minute, second, zone sign,, zone hour, zone minute. All arguments, except date sign and zone sign, are assumed to be positive integers, while date sign and zone sign can be either 1 or -1. For negative time zones both values have to be negative, for dates the same. Date sign and zone sign are not available. • _dateTime[_toType(_string) => _dateTime] • _dateTime[=> _isTypeOf(_object)] Tells if object belongs to the primitive type _dateTime
Component methods	_dateTime[_dateSign *=> _integer] _dateTime[_year *=> _integer] _dateTime[_month *=> _integer] _dateTime[_day *=> _integer] _dateTime[_hour *=> _integer] _dateTime[_minute *=> _integer] _dateTime[_second *=> _integer] _dateTime[_zoneSign *=> _integer] _dateTime[_zoneHour *=> _integer] _dateTime[_zoneMinute *=> _integer]
Other methods	_dateTime[_toString *=> _string] _dateTime[*=> _equals(_object)] _dateTime[*=> _lessThan(_object)] _dateTime[_typeName *=> _string] _dateTime[_add(_duration) *=> _dateTime]
Examples	"2001-11-23T12:33:55.123-02:30"^^_dateTime "2001-11-23T12:33:55.123-02:30"^^<http://www.w3.org/2001/XMLSchema#dateTime> "-0237-11-23T12:33:55"^^_dateTime Note that this date refers to year 238 BCE. "2001-11-23T10:00:00Z"^^_dateTime[_day -> 23] "2001-11-23T10:00:00Z"^^_dateTime.[_toString -> '2001-11-23T00:00:00+00:00'] "2001-11-23T18:33:44-02:30"^^_dateTime[_add("-P22Y2M10DT1H2M3S"^^_duration) -> "1979-09-13T17:31:41-02:30"^^_dateTime].

	_date
Explanation	XML Schema date type
Normal form	"ZYYYY-MM-DDSHH:MM"^^_date. The symbols - and : are part of the syntax. The symbol S represents the timezone sign (+ or -). The timezone part (beginning with S) is optional. The leftmost Z is optional sign (-). Note that unlike _dateTime, which represents a single time point, _date represents duration of a single day.
Class	_date
Synonyms	_d, <http://www.w3.org/2001/XMLSchema#date>.
Class methods	• _date[_toType(_integer,_integer,_integer,_integer,_integer,_integer,_integer) => _date] • _date[_toType(_integer,_integer,_integer,_integer,_integer) => _date] The meaning of the arguments is as follows (in that order): date sign, year, month, day, zone sign, zone hour, zone minute. All arguments, except date sign and zone sign, are assumed to be positive integers, while date sign and zone sign can be either 1 or -1. For negative time zones both values have to be negative, for dates the same. Date sign and zone sign are not available. • _date[_toType(_string) => _date] • _date[=> _isTypeOf(_object)] Tells if object belongs to the primitive type _date
Component methods	_date[_dateSign *=> _integer] _date[_year *=> _integer] _date[_month *=> _integer] _date[_day *=> _integer] _date[_zoneSign *=> _integer] _date[_zoneHour *=> _integer] _date[_zoneMinute *=> _integer]
Other methods	_date[_toString *=> _string] _date[*=> _equals(_object)] _date[*=> _lessThan(_object)] _date[_typeName *=> _string] _date[_add(_duration) *=> _date]
Examples	"2001-11-23-02:30"^^_date "2001-11-23"^^_date, "-0237-11-23"^^_date Note that this date refers to year 238 BCE. "2001-11-23"^^_date[_day -> 23] "2001-11-23"^^_date[_toString -> '2001-11-23+00:00'] "2001-11-23-02:30"^^_date[_add("-P2Y2M10DT1H2M"^^_duration) -> "1979-09-13-03:32"^^_dateTime]

	_time
Explanation	XML Schema time data type
Normal form	"HH:MM:SS.sZHH:MM"^^_time The symbols : and . are part of the syntax. The part .s is optional. It represents fractions of a second. Here s can be any positive integer. The sign Z represents the sign of the timezone (+ or -). The following HH represents time zone hours and MM time zone minutes. The time zone part is optional.
Class	_time
Synonyms	_t, <http://www.w3.org/2001/XMLSchema#time>
Class methods	• _time[_toType(_integer,_integer,_decimal,_integer,_integer,_integer) => _time] • _time[_toType(_integer,_integer,_decimal,_integer,_integer) => _time] The arguments represent hour, minute, second, time zone sign, time zone hour, and time zone minute. • _time[_toType(_string) => _time] • _time[=> _isTypeOf(_object)] Tells if object belongs to the primitive type _time
Component methods	_time[_hour *=> _integer] _time[_minute *=> _integer] _time[_second *=> _integer] _time[_zoneSign *=> _integer] _time[_zoneHour *=> _integer] _time[_zoneMinute *=> _integer]
Other methods	_time[_toString *=> _string] _time[*=> _equals(_object)] _time[*=> _lessThan(_object)] _time[_typeName *=> _string] _time[_add(_duration) *=> _time]
Examples	"11:24:22"^^_time "11:24:22"^^<http://www.w3.org/2001/XMLSchema#time> _time[_toType(12,44,55) -> "12:44:55"^^_time] "12:44:55"^^_time[_minute -> 44] "12:44:55"^^_time[_toString -> '12:44:55'] "12:44:55"^^_time[_add("P2M3S"^^_duration) -> "12:46:58"^^_time]

	_duration
Explanation	XML Schema duration type
Normal form	"sPnYnMnDTnHnMnS"^^_duration Here s is optional sign (-), P indicates that this is a duration type, and Y, M, D, H, M, S denote year, month, date, hour, minutes, and seconds. T separates date from time. The symbol n stands for any positive number (for instance, the number of hours can be more than 12 and the number of minutes and seconds can be more than 60. The part that starts with T is optional and some of the elements in the date and the time parts can be omitted.
Class	_duration
Synonyms	<http://www.w3.org/2001/XMLSchema#duration>
Class methods	• _duration[_toType(_integer,_integer,_integer,_integer,_integer,_integer) => _duration] The meaning of the arguments (in that order) is: year, month, day, hour, minute, second. • _duration[_toType(_string) => _duration] • _duration[=> _isTypeOf(_object)] Tells if object belongs to the primitive type _duration
Component methods	_duration[_years *=> _integer] _duration[_months *=> _integer] _duration[_days *=> _integer] _duration[_hours *=> _integer] _duration[_minutes *=> _integer] _duration[_seconds*=> _integer]
Other methods	_duration[_toString *=> _string] _duration[*=> _equals(_object)] _duration[*=> _lessThan(_object)] _duration[_typeName *=> _string] _duration[_add(_duration) *=> _duration]
Examples	"P5Y5M10DT11H24M22S"^^_duration "-P2Y05M10DT11H24M22"^^_duration[_minutes -> 24]

	_boolean
Explanation	XML Schema Boolean type
Normal form	"true"^^_boolean, "false"^^_boolean or shorter forms: _true, _false
Class	_boolean
Synonyms	<http://www.w3.org/2001/XMLSchema#boolean>
Class methods	• _boolean[_toString => _string] • _boolean[=> _isTypeOf(_object)]
Other methods	_boolean[_toString *=> _string] _boolean[*=> _equals(_object)] _boolean[*=> _lessThan(_object)] **Note:** _false[_lessThan(_true)]) _boolean[_typeName *=> _string]
Examples	"true"^^_Bool, _true, _false

	_double
Explanation	XML Schema double type
Normal form	"value"^^_double where value is a floating point number of double length that uses the regular decimal point representation with an optional exponent. Short forms: same representation without "..."^^_double wrapper.
Class	_double
Synonyms	<http://www.w3.org/2001/XMLSchema#double>
Class methods	• _double[_toType(_decimal) => _double] Converts decimals to doubles. Error, if overflow. • _double[_toType(_long) => _double] Converts long integers to doubles. • _double[=> _isTypeOf(_object)]
Other methods	_double[_floor *=> _integer] _double[_ceiling *=> _integer] _double[_round *=> _integer] _double[_toString *=> _string] _double[*=> _equals(_object)] _double[*=> _lessThan(_object)] _double[_typeName *=> _string]
Examples	"2.50"^^_double, 2.50, 25E-1

	_decimal
Explanation	XML Schema decimal type (arbitrary precision)
Normal form	"value"^^_decimal where value is a number that uses the regular decimal point representation. This type is a supertype of _integer.
Class	_decimal
Synonyms	<http://www.w3.org/2001/XMLSchema#decimal>
Class methods	• _decimal[_toType(_string) => _decimal] Converts string to decimal, if string represents a decimal in textual form. If it does not, then the method fails. • _decimal[_toType(_double) => _decimal] Converts doubles to decimals. • _decimal[=> _isTypeOf(_object)]
Other methods	_decimal[_floor *=> _integer] _decimal[_ceiling *=> _integer] _decimal[_round *=> _integer] _decimal[_toString *=> _string] _decimal[*=> _equals(_object)] _decimal[*=> _lessThan(_object)] _decimal[_toString *=> _string] _decimal[_typeName *=> _string]
Examples	"2.50"^^_decimal, "12"^^<http://www.w3.org/2001/XMLSchema#decimal>

	_integer
Explanation	XML Schema integers (arbitrary length)
Normal form	"value"^^_integer where value is an integer in its regular representation in the decimal system. _integer is a subtype of _decimal. For instance, "12"^^_integer = "12"^^_decimal = "12.0"^^_decimal, etc. **Note:** Although XML schema prescribes that _integer must be a supertype of _long, we do not enforce this rule due to the difficulty in implementing it. However, the _equals method is supposed to recognice that _long is a subtype of _integer, i.e., "18"^^_long [_equals("18"^^_int)] must be true and so must be "18"^^_int[_equals("18"^^_long)].
Class	_integer
Synonyms	_int, <http://www.w3.org/2001/XMLSchema#integer>
Class methods	• _integer[_toType(_string) => _integer] Converts strings to integers, if the string represents an integer in textual form. If it does not then this method fails. • _integer[_toType(_long) => _integer] Converts long to integers. • _integer[=> _isTypeOf(_object)]
Component methods	
Other methods	• _integer[_toString *=> _string] _integer[*=> _equals(_object)] _integer[*=> _lessThan(_object)] _integer[_typeName *=> _string]
Examples	"55"^^_integer, "55"^^_int

	_long
Explanation	XML Schema long integers
Normal form	"value"^^_long where value is an integer in its regular representation in the decimal system. A shorter form without the "..."^^_long wrapper is also allowed. Note that although XML schema prescribes that _long must be a subtype of _integer, we do not enforce this rule due to the difficulty in implementing it. However, the _equals method is supposed to recognice that _long is a subtype of _integer, i.e., "18"^^_long [_equals("18"^^_int)] must be true and so must be "18"^^_int[_equals("18"^^_long)].
Class	_long
Synonyms	<http://www.w3.org/2001/XMLSchema#long>
Class methods	• _long[_toType(_string) => _long] Converts strings to long integers, if the string represents an integer in textual form. If it does not then this method fails. • _long[_toType(_integer) => _long] Converts long integers to arbitrary big integers. • _integer[=> _isTypeOf(_object)]
Component methods	
Other methods	• _long[_toString *=> _string] _long[*=> _equals(_object)] _long[*=> _lessThan(_object)] _long[_typeName *=> _string]
Examples	123, 55, "55"^^_long

	_iri
Explanation	IRI as defined by [RFC 2396], as amended by [RFC 2732]
Normal form	"string IRI representation"^^_iri
Class	_iri
Synonyms	<http://www.w3.org/2007/rif#iri> <http://www.w3.org/2007/rif#uri> _"string IRI representation". For instance, "http://foo.bar.com/moo"^^_iri can be written as _"http://foo.bar.com/moo".
Class methods	• _iri[_toType(_string) => _iri] • _iri[=> _isTypeOf(_object)]

Component methods	_iri[_scheme *=> _string]
	_iri[_user *=> _string]
	_iri[_host *=> _string]
	_iri[_port *=> _string]
	_iri[_path *=> _string]
	_iri[_query *=> _string]
	_iri[_fragment *=> _string]
	
	Note:
	The exact meaning of the above components depends on the IRI scheme. For http, ftp, file, etc., the meaning the first five components is clear. The query is an optional part of the IRI that follows the ?-sign, and fragment is the last part that follows #. Some components might be optional for some IRI schemes. For instance, for the urn and file schemata, only the path component is defined. For mailto scheme, port, path, query, and fragment are not defined. If a scheme is not recognized then the part of the IRI that follows the scheme goes into the path component unparsed.
	`_iri[_localName *=> _string]`
	`_iri[_namespace *=> _string]`
	These two attributes return the local name and prefix of the IRI as defined by RDF best practices. **** Define local names! **** The prefix is the part of the IRI that precedes the local name.

Other methods	_iri[_toString *=> _string] _iri[*=> _equals(_object)] _iri[_typeName *=> _string]
Examples	_"http://foo.bar.com/abc" "http://foo.bar.com/abc"^^_iri _iri[_toType('http://foo.bar.com/abc') -> "http://foo.bar.com/abc"^^_iri] _"http://foo.bar.com/abc"[_host -> 'foo.bar.com']

	_string
Explanation	XML Schema string type Characters can be Unicode characters, excluding the surrogate blocks FFFE and FFFF.
Normal form	"value"^^_string A shorter form 'value' is also allowed. Single-quoted sequence of characters. Alphanumeric strings that start with a letter do not need to be quoted. In the full representation (with the "..."^^_string wrapper), the double quote symbol and the backslash must be escaped with a backslash. In short representation, the single quote symbol and the backslash must be escaped with a backslash.

Class	_string
Synonyms	http://www.w3.org/2001/XMLSchema#string
Class methods	• _string[=> _isTypeOf(_object)]
Other methods	• _string[*=> _equals(_object)]. Binding: bb. • _string[*=> _lessThan(_object)]. Binding: bb. • _string[_typeName *=> _string]. Bindings: bf, bb. • _string[*=> _contains(_string)]. Binding: bb. • _string[_concat(_string) *=> _string]. Bindings: fbb, bfb, bbf, bbb • _string[_cut(_integer) *=> _string]. Bindings: bbf, bbb. Returns a substring of the string-object truncated by the number of characters specified in the method argument. Truncation takes place on the right. • _string[_fill(_string) *=> _string]. Bindings: bbbf, bbbb. Fills the object-string with a letter from the first method argument up to the length specified in the second method argument. For instance, abc[_fill(D,7) -> abcDDDD]. • _string[_match(_string) *=> _list]. Bindings: bbf, bbb. The result of the method is a list of terms of the form match(matchingPosition,matchingSubstring), which represents matches inside the object-string. The match is done against a Perl regular expression given as the method argument. The list is sorted in the ascending order of positions of the matches. The counting of positions in the object string starts with 0. • _string[_indexOf(_string) *=> _int]. Bindings: bbf, bbb. The result of this method is a list of positions at which the string given as argument to the method matches within the object string. The positions in the list are ordered in the ascending order. • _string[_lastIndexOf(_string) *=> _int]. Bindings: bbf, bbb. The result of this method is a list of positions at which the string given as argument to the method matches within the object string. The positions in the list are ordered in the descending order.For this behaviour, use _indexOf together with reverse. This implementation delivers just the last position. • _string[_firstIndexOf(_string) *=> _int]. Bindings: bbf, bbb. This method is added from us. It delivers just the first position. • _string[_replace(_string,_string) *=> _string]. Binding: bbbf, bbbb. Replaces every occurrence of the Perl regular expression inside the object string with the string given in the second argument of the method. The Perl regular expression is given in the first argument. • _string[_split(_string) *=> [_string,_string]]. Binding: bbf, bbb. Splits the object-string by the regular expression given in the method argument. The split occurs at the first character of the first match. The result is a list of two strings obtained by the split. For instance, Eddie[_split(d) -> [E,ddie]]. • _string[_reverse *=> _string]. Bindings: bf, fb, bb. • _string[_length *=> _integer]. Binding: bf, bb. • _string[_tokenize(_string) *=> _list]. Bindings: bbf, bbb. Breaks the object-string into a list of tokens at every occurrence of any delimiter character given in the argument to the method (this argument is a string of delimeter characters). The delimiter characters are not retained as part of the tokens. For instance, 'xyzw.foo'[_tokenize ('y.') -> [x,zw,foo]]. • _string[_toUpper *=> _string]. Bindings: bf, bb. The value of the method is the object string with all characters converted to the upper case. • _string[_toLower *=> _string]. Bindings: bf, bb. The value of the method is the object string with all characters converted to the lower case. • _string[_charAt(_integer) *=> _string]. Bindings: bbf, bbb. The value of the method is the character that occurs in the object string at the position given by the argument to

	the method. Positions start with 0. • _string[_diffAt(_string) *=> _number_integer]. Bindings: bbf, bbb. The result of the method is the first position where the object string and the method argument differ. Positions start with 0. The the truth value is false if the two strings are equal. • _string[*=> _startsWith(_string)]. Binding: bb. • _string[*=> _endsWith(_string)]. Binding: bb. • _string[_substring(_integer,_integer) *=> _string]. Binding: bbbf.and: bbbb Returns a substring of the object string, where the starting and the ending position of the substring are given by the arguments of the method. -1 in argument 2 means the end of the string. • _string[_trim(_string) *=> _string]. Bindings: bbf, bbb. Deletes from the object string all the leading and trailing characters specified in the argument to the method. For instance, ' .ab c,\n'[_trim('\n\r .,') -> 'ab c']. • _string[_overlappedBy(_string) *=> _integer]. Bindings: bbf, bbb. True if a suffix of the object string is also a prefix of the string given in the method's argument. The result of the method is the number of characters in the maximal overlap.
Examples	"abc"^^_string 'abc' "a string\n"^^_String 'a string\n' 'a\tstring\b' 'string with a \'quoted\' substring'

	_PlainLiteral
Explanation	XML Schema string type with additional language tag. NOTE: this data type is currently under discussion at W3C, so the specifics might change. This is like _string, but the same string with different language tags are considered different. _text is a subtype of _string, so all of the methods of _string are applicable to _text. We don't support this subtype property, because we could not decide what type to generate as result. If needed, _text has to be converted to string explicitly. The language tag can be also ISO-639-1 (or ISO-639-2) to specify encodings.
Normal form	"value@language"^^_text Escaping conventions are the same as for _string.
Class	_text
Synonyms	http://www.w3.org/2007/rif#text
Class methods	See _string
Other methods	See _string. _text[_language *=> _string]. Binding bf, bb. Get the language of the string. _text[_textValue *=> _string]. Binding: bf, bb. Gets the text without the language tag.
Examples	"abc@en"^^_text "a string\n@en"^^_Text

	_list
Explanation	The Prolog type of list.
Normal form	[elt1, ..., eltn], [elt1,...,eltn|List] This is the usual type for list terms.
Class	_list
Class methods	• _list[=> _isTypeOf(_object)] • _list[_toType(_object,_list) => _list] This method is here for completeless only. It defined simply as _list[_toType(obj,list) => [obj|list], for all objects obj and lists list.
Other methods	• _list[_toString *=> _string] • _list[*=> _lessThan(_object)] • _list[*=> _equals(_object)]. Binding: bb. • _list[_typeName *=> _string] • _list[*=> _contains(_list)]. Binding: bb. Tells if a list object contains the method's argument as a sublist. • _list[*=> _member(_object)]. Binding: bb. The method's argument and the list-object may not be fully ground. In this case, the method succeeds, if the argument to the method unifies with a member of the list. • _list[_append(_list) *=> _list]. Bindings: fbb, bfb, bbf, bbb • _list[_indexOf(_object) *=> _long]. Bindings: bbf, bbb. The result of this method is the first position at which the argument-object unifies with the list-object. • _list[_length *=> _long]. Binding: bbbf, bbbb. Computes the length of the list. • _list[_reverse *=> _list]. Bindings: bf, fb, bb. • _list[_sort *=> _list]. • _list[_memberAt(_integer) *=> _object]. Bindings: bbf, bbb. The value of the method is the object that occurs in the list object at the position given by the argument to the method. Positions start with 0. • _list[*=> _startsWith(_list)]. Binding: bb. • _list[*=> _endsWith(_list)]. Binding: bb. • _list[*=> _subset(_list)]. Binding: bb. True if the list object contains the argument list. • _list[_overlappedBy(_list) *=> _integer]. Bindings: bbf, bbb. True if a suffix of the list object is also a prefix of the list given in the method's argument. The result of the method is the number of characters in the maximal overlap.
Examples	[a,b,c] [a,b|?X] [a,b,c|[d,e]]

	_XMLLiteral
Explanation	XMLLiteral type of RDF. An XML literal is a string that contains a well-formed fragment of XML text. _XMLLiteral is a subtype of _string, so all of the methods of _string are applicable to _text. We don't support this subtype property, because we could not decide what type to generate as result. If needed, _XMLLiteral has to be converted to string explicitly.
Normal form	"some fragment of XML"^^_XMLLiteral
Class	_XMLLiteral
Synonyms	http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral
Class methods	See _string
Component methods	
Other methods	See _string. Should add a path expression method.
Examples	"<foo>bar</foo>"^^_XMLLiteral

21.4.4.2 Miscellaneous

ap data typ

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]] equivalent to ?X=[a->1,a->2,a->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.
?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) }. translates into: ?- _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]. returns: ?X = a, ?Y = 1 ?X = a, ?Y = 2 ?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]. returns: ?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]. returns: ?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]. returns: ?X = [a->2,b->3,c->4]	Built-in _difference/2 to remove all key/ value pairs of argument1 from argurment 0
?- [b->1,a->4,a->2][_normalize->?X]. returns: ?X = [a->{2,4}, b->1]	Built-in _normalize/2 to bring map in internal order
?- _map[_toType(a,2)->?X].	Extended built-in _toType/4 to generate singleton map

ObjectLogic syntax	Description
returns: ?X = [a->2]	
?- ?M = [a->[b->[1,2], c->3], d->4],_memberByPath(?M,_path(a, b),?V). returns ?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). returns P I V --- "a.b" 0 1 "a.b" 1 2 "a.c" 0 3 "d" 0 4 ?- ?M = [a->[b->{1,2}, c->3], d->4], _map2table(?M,?P,?I,?V). returns P I V --- "a.b" 0 1 "a.b" 1 2 "a.c" 0 3 "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]. returns: 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"].
?- ?M = collectmap { ?Z
 | ((?Z=[?X,?Y], NOT ?Y:?)
      OR (?M1 = collectmap {?Z1 | ?Z1=[?X1,?Y1], ?Y[?X1->?Y1]}, ?Y:?), ?Z=[?X,?M1]),
juergen[?X->?Y] }.
```

results in:

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

21.4.4.3 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,[])))

from the copyright of the web page http://forum.projects.semwebcentral.org/forum-syntax.html

21.5 Aggregations

You can get a complete list of aggregations if you run the following query:

```
?- _aggregations(?NAME,?ARITY,?DESCRIPTION,?PARAMETERS).
```

Name	Arrity	Description	Parameters
collectPropertyEntries	3	collects property entries for later insertion in java class	["property","value","order"]
dynSvcBuildTree	5	collects property entries for later insertion in java class	["cls","id","property","value","order"]
count	1	counts items	["value"]
max	1	computes maximum value for input values	["value"]
min	1	computes minimum value for input values	["value"]
sum	1	computes sum of all input values	["value"]
avg	1	computes average of a set of values	["value"]
collectmap	2	collects key/value pairs in a map	["key","value"]
collectbag	1	collects all items in a list	["key","value"]
collectset	1	collects all distinct items in a list	["key","value"]

21.6 Connectors

You can get a complete list of connectors if you run the following query:
`?- _connectors(?NAME,?ARITY,?DESCRIPTION,?PARAMETERS).`

Name	Arrity	Description	Parameters
_reqlevel	1	returns the requirements level of the current program	["requirements level"]
_sparqlquery	5	executes sparql queries	["address","list of default graphs","list of named graphs","sparql query","map with result"]
_queryIndex	10	queries full-text index for ontology of given module	["module","options","query string","start","batch","object","total count","score","order","optional"]
_sqlexecute	3	executes SQL query	["name of table","which columns in which variables (e.g. f(\"ID\",X,\"TYPE\",Y))","resource manager key"]
_webserviceAccess	3	Calls webservice	["webservice input map","input parameter map","output map"]
_dbaccessuser	3	gets facts from database	["name of table","which columns in which variables (e.g. f(\"ID\",X,\"TYPE\",Y))","ResourceManager key"]
sparqlaccessinternal	5	executes sparql queries	["address","list of default graphs","list of named graphs","sparql query","map

			with result"]
_sparqlqueryinternal	6	executes sparql queries	["address","list of default graphs","list of named graphs","sparql query","map with result","variables for tuple cache"]
_excelaccess	7	accesses MS Excel spreadsheets	["map with result","filepath","sheet-name","start-row","end-row","start-column","end-column"]
_sparqlaccess	4	executes sparql acces	["address","list of default graphs","list of named graphs","map with result"]
_webserviceInfo	4	Retrieves information for web service	["input map","service","method","method info map"]
_echoConnector	2	echos items of list as output	["input list","output"]
_obquery	3	queries remote OntoBroker	["Object logic query","map with result","connection data source"]

21.7 Properties Settings

OntoBroker settings can be changed in OntoConfig.prp file at the OntoBroker home directory.

Property	Values	Default Value
OntologyLanguage	ObjectLogic, RDF, OWL	ObjectLogic
Storage	RAM.choose, RAM.AVL.Packed, H2	RAM.choose
EvaluationMethod	choose, DynamicFiltering, DynamicFiltering2, BottomUp, DisjunctiveMagicSet	choose
BottomUpEvaluator	choose, BottomUp2, DBBottomUp, BU3	choose
DynamicFiltering.UnfoldRules	on, off	off
Connector.Cache.Lifetime	query,unlimited. query - tuple cache is removed after the query execution. This is the default value for this option. unlimited - tuple cache is not removed after the query execution and can be used for the next query. This option is better for performance and should always be used if a database won't be updated frequently. Cache update should be done manually (see "Clear connector cache command"). The option affects all connectors.	unlimited
SPARQL Endpoint	SparqlEndpoint = on	off
Extensions.Directory	To specify multiple extensions directory just use the OntoConfig parameter Extensions.Directory as usual and separate the directories by comma.	-

21.7.1 Start-Up Settings

Parameter	Parameter	Description
-fenc	string value	File encoding
-p	number	Port (default: 2267)
-ip	<IP adress>	IP address to be used
-builtinDir	<directory name>	directory where built-ins are located
-configFile	<file name>	the place of the configuration file
-wsport	number	defines the port when starting as webservice
-sparqlEndpoint	-	If left unchanged in the settings (option HttpPort), the default port is 8267.
-X:indexArguments	<predicateName>'/'<predicateArity>	indexes all arguments of the specified predicate
-X:indexAll	<predicateName>'/'<predicateArity>	indexes all possible positions of the specified predicate
-webservice	-	starts OntoBroker server as web service
-help	-	shows the usage of the start-up settings
-h	string value	host on which you start the server e.g. "localhost"
-commitDuringImport	-	sends a commit command to the internal database during import after a fixed set of added facts
-X:memorytest	-	Starts OntoBroker, loaded ontologies, logs the memory consumption and exits
-m	-	materialization of the following file or directory
-X:performancetest	<suite name>	contains the suite name
-X:performancetest: QueryFile	<query file name>	file which contains a set of queries
-X:performancetest: OutputFile	<output file name>	contains the output data
-silent	-	starts OntoBroker without log output (such as console output)
-rdf, -rdfs,-owl, -nt, -nt3, -obl	<directory name > or <file name>	specifies the loaded files/directory according to the used data format
-collabserver	-	Enables OntoBroker collaboration server (implicitly set Indexer = on)
-collabserverNoIndexer	-	Enables OntoBroker collaboration server (leaves Indexer settings unchanged)
-project	<projectFile>	Use project file for advanced load configuration.
-webconsole	-	Enables OntoBroker web console. The query results in the webconsole can be copied to the clipboard (e.g.

21.7.2 Optimizer Settings

Property	Values	Default Value
Rewriter.ShrinkDBAccess	on, off	on
Rewriter.FLToSqlDBAccessUserRewriter	on, off	on
Rewriter.RedundantLiteralEliminator	on, off	off
Rewriter.EliminateBodies	on, off	off
Rewriter.EqualUnifyRewiter	on, off	off
Rewriter.PropagateConstraints	on, off	on
Rewriter.SchemaOptimizer	on, off	off
Rewriter.SimpleUnfolder	on, off	off
Rewriter.PropagateConstants	on,off	on
Rewriter.TransitivityRewriter	on,off	off
Rewriter.FrameOptimizer	on,off	off
Rewriter.StrongUnfolder	on,off	off

21.7.3 Tracing and Debugging Settings

Property	Values	Default Value
Trace.ExternalDatabase	on, off	off
Trace.ExternalDatabase.SQL	on, off	off
Trace.Statistics.Rule	on, off	off
Trace.Materialization	on, off	off
Trace.Evaluation	off, simple, rules, full	off
Trace.Rewriter	on, off	off
Trace.Rules	on, off	on
Trace.Queries	on, off	off
Trace.Datamodel.Changes	on, off	off
Trace.Datamodel.Requests	on, off	off

21.7.4 Performance-Related Settings

Property	Values	Default Value
ConceptNamesGround	on, off	off
AttributeNamesGround	on, off	off
ModuleNamesGround	on, off	off
WellfoundedEvaluation	on, off	off
EliminateDuplicates	on, off	on
BodyOrdering	auto, off	auto
BodyOrderingDataFile	= bodyordering.data	-

21.7.5 Security (Authentication & Authorization)

Property	Values	Default Value
Security.LoginRequired	on, off	on
Security.AccessControl	on, off	on

21.7.6 Parallelization Settings

Property	Values	Default Value
Parallelize.ServerCommands	on, off	on
Parallelize.ServerCommands.MaxThreads	number, auto	auto
Parallelize.Loader	on,off	on
Parallelize.Loader.MaxThreads	number, auto	auto
Parallelize.Reasoner	on,off	on
Parallelize.Reasoner.MaxThreads	number, auto	auto
Parallelize.Reasoner.Connectors.MaxThreads	number, auto	auto

21.7.7 Materialization Settings

Property	Values	Default Value
Materialize.ObjectLogic.AttributeInheritance	on,off	off
Materialize.ObjectLogic.Attributes	on,off	off
Materialize.ObjectLogic.Concepts	on, off	off
Materialize.ObjectLogic.Relations	on,off	off
Materialize.ObjectLogic.RootConcepts	on,off	off
Materialize.ObjectLogic.SubclassTransitivity	on,off	off
Materialize.ObjectLogic.SubsetRelationship	on,off	off
Materialize.ObjectLogic. StructuralInheritanceForInstances	on,off	off
Materialize.ObjectLogic. PropertySubsetRelationship	on,off	off
Materialize.ObjectLogic.SubpropertyTransitivity	on,off	off
Materialize.RDF.Helper.Subclass	on,off	off
Materialize.RDF.Helper.Subproperty	on,off	off
Materialize.RDF.Helper.Generic	on,off	off
Materialize.RDF.Helper.TypeTemp	on,off	off
Materialize.RDF.Helper.Type	on,off	off
Materialize.RDF.PropertyDef.ByPredicate	on,off	off
Materialize.RDF.PropertyDef.ByDomain	on,off	off
Materialize.RDF.PropertyDef.ByRange	on,off	off
Materialize.RDF.PropertyDef. BySubpropertySubject	on,off	off
Materialize.RDF.PropertyDef. BySubpropertyObject	on,off	off
Materialize.RDF.ClassDef.ByType	on,off	off
Materialize.RDF.ClassDef.ByDomain	on,off	off
Materialize.RDF.ClassDef.ByRange	on,off	off
Materialize.RDF.ClassDef.BySubclassSubject	on,off	off
Materialize.RDF.ClassDef.BySubclassObject	on,off	off
Materialize.RDF.Entailment.Domain	on,off	off
Materialize.RDF.Entailment.PropertyInheritance	on,off	off
Materialize.RDF.Entailment.Range	on,off	off
Materialize.RDF.Entailment.PropertyTransitivity	on,off	off
Materialize.RDF.Entailment.ClassInheritance	on,off	off
Materialize.RDF.Entailment.ClassTransitivity	on,off	off

21.7.8 Logging-Related Settings

Property	Values	Default Value
LogStratificationProblemDetails	on, off	off
VisualizeStratificationProblemDetails	on, off	off
LogPerformanceWarnings	on, off	on

21.7.9 Server Settings

Property	Values/Description	Default Value/Example
CommandPort	any number between 1 and 65000	2267
HttpPort	any number between 1 and 65000	8267
FrontSideCache	on, off	off
AJP.Port	any number between 1 and 65000	8009
CollaborationServer	on, off	off
RemoteAPI	on, off	off
WebServices	on, off	off
WebConsole	on, off	off
AJP	on, off	off
Session.TimeOut	Specifies the timeout for an OntoBroker session in milliseconds.	Session.TimeOut = 1800000
ExternalDB.MSSQLServer. CollationOverride	The values of these parameters are used on creating VARCHAR and NVARCHAR columns in temporary tables, respectively.	ExternalDB.MSSQLServer. CollationOverride = Latin1_General_CI_AI
ExternalDB.MSSQLServer. CollationOverrideN		

21.7.10 H2 Storage Options

Property	Values	Default Value	Example
H2.Url	URL (only local)	-	jdbc:h2:file:data/edb
H2.User	ASCII string	sa	
H2.Password	ASCII string		
H2.TermCacheSize	number	100000	

21.7.11 Fulltext Indexer Settings

Property	Values	Default Value
FullTextIndex	on, off enables the full text index engine. With default configuration, full text indexes for all ontologies are automatically generated and can be used with the _queryIndex/10 builtin.	on
FullTextIndex.ChangeLogInRam	on, off Advanced configuration option to disable persistent change log if a persistent datamodel (i.e. H2) is used. Has no meaning when using a RAM datamodel	on
FullTextIndex.Directory	Directory to use for storing the full text index files.	index
FullTextIndex.NoIndexBuilding	on, off By default it is off (and OntoBroker's behaviour is unchanged). If it is turned on, OntoBroker does not build or updates any indexes, but can still use the FullTextIndex infrastructure (inclusive searching indices)	off
FullTextIndex.WaitOnStartup	on, off If this switch is turned on, the server waits for the end of the indexing on startup, before any queries and commands are executed. Please note, that this does not apply to direct access to the KAON2 API via the OntologyManager.	off
FullTextIndex.ReuseOnRestart	on, off If this switch is on and the OntoBroker runs with a RAM datamodel (Storage = RAM.Choose, etc), then the index file are reused on OntoBroker restart if the ontology files are the same as during the creation of the index.	off

The indexer can now be customized with additional fields. Values for a defined set of attributes can be added to additional Lucene index fields and then used with the _queryIndex/10 built-in or for autocompletion.

OntoBroker automatically builds an index with several predefined fields (e.g. id, name_de, name_en, all, ...) for all ontology objects if the OntoConfig.prp parameter "Indexer" is set to "on".To customize this index, you have to adapt the configuration file "indexer-context.xml" located in the configuration directory (defaults to ./conf relative to the working directory). There you can define custom index fields for the bean with the id "ObjectLogicDocumentBuilder".

21.7.11.1 Examples and Scenarios

Scenario for "FullTextIndex.WaitOnStartup":

The full text indexes are always generated and updated asynchronously in the background. If it is important that the full text index are complete directly after the start of OntoBroker, turn FullTextIndex. WaitOnStartup=on. In this case the OntoBroker waits for completion of index generation on startup before it executes any command or queries.

Scenario for "FullTextIndex.ReuseOnRestart":

If OntoBroker runs in a read-only mode, e.g. OntoBroker only loads some ontologies on startup and then the ontologies are not changed anymore (at least without saving them), this option can be used to improve startup time. If the full text indexes have already been generated, OntoBroker will them reuse them on a restart without rebuilding them.

Configuration example

```
   ...

   <bean id="ObjectLogicDocumentBuilder" class="com.ontoprise.indexer.flogic.
ObjectLogicStandardObjectDocumentBuilder">
     ...
     <property name="customIndexFields">
        <list>
            <bean class="com.ontoprise.indexer.objectlogic.CustomIndexField">
             <property name="fieldName" value="xname" />
             <property name="stored" value="true" />
             <property name="attributeNames">
               <set>
                 <value>&lt;http://my.name.space#secondary%20name&gt;</value>
                 <value>&lt;http://my.name.space#Label&gt;</value>
               </set>
             </property>
           </bean>
        </list>
     </property>
   </bean>
   ...
```

In this example, one custom index field with the name "xname" is defined. The values are stored in the index for retrieval (this is only needed if it is used for autocompletion or if you want to use the return() option of the _queryIndex built-in). All values of the attributes <http://my.name.space#secondary%20name> and <http://my.name.space#Label> are indexed for any ontology object containing such an attribute.

Search Example

To search in this field with _queryIndex/10, use the Lucene syntax for fields (here "xname:") in front of your search text.

```
?- _queryIndex(<http://your.company.com#ontology>, [return(xname)], "xname:Transp*",
0, 10, ?OBJ, ?TC, ?SCORE, ?ORDER, ?OPT).
```

Autocomplete Example

For autocomplete you have to set the search field in the AutoComplete

```
AutocompleteHelper helper = ...
Ontology ontology = ...
AutocompleteHelper.Type type = ...
AutocompleteHelper.Options options = new AutocompleteHelper.Options();

options.setSearchField("xname");

options.set...
```

```
CompletionResults results = helper.getCompletion(ontology, options, type, "Transp",
0, 10);
...
```

21.7.11.2 Custom Index Fields

The indexer can now be customized with additional fields. Values for a defined set of attributes can be added to additional Lucene index fields and then used with the _queryIndex/10 built-in or for autocompletion.

Configuration

If the full-text index engine is enabled, i.e. the parameter "FullTextIndex = on" is set in the OntoConfig.prp, OntoBroker creates an full-text index for every ontology.The fields and values can be configured with two files in the conf directory:

- `fulltextindex-config.xml`

  Contains the general options and the definition of the index fields.
- `fulltextindex-schema.xml`

  Contains the index schema definition, i.e. field types and analyzers to be used for indexing and querying.
  See also Apache Solr documentation wiki.apache.org/solr/SchemaXml
  In contrast to Apache Solr, the index fields are not defined in the schema definition, but in the `fulltextindex-config.xml`.

Also note that the full text index engine only works with extensional data, i.e. facts stored in the OntoBroker datamodel by import from ontology files, API insert/delete, materialization.

General Options

property	Description
schema	path to schema definition, relative to conf directoy
excludedOntologies	set of ontologies, identified by ontologyURI, which should not be full-text indexed
batchSize	number of objects processed in one batch during index creation
flushSize	number of objects processed before flushing to disk
waitTime	waiting time before starting processing invalidated objects
languages	languages to consider for representations, synonyms and custom fields with language pattern
booleanQueryMaxClauseCount	maximum number of Lucene boolean clauses
defaultQueryOperator	default query operator if no operator is specified. Default is OR, alternative is AND, but this has a major impact on the overall behavior of the search
defaultStringMetric	string metric used for fuzzy search if not explicitly specified. Possible values are e.g. Jaro, Levenstein, JaroScaled, JaroWinkler, DamerauLevenshtein, DamerauLevenshteinScaled, MaxJaroDamerauLevenshteinScaled, Jaccard, Soundex, SmithWatermna

Standard index fields

With the default configuration, there are the following index fields available for every ontology:

Field name	Description
id	object id
lid	local name of object id
type	Object type: c = Concept, i = Instance, a = Attribute, r = Relation, u = Rule, q = Query, t = Constraint
assertedisa	all concept ids, this object id (instance) has an assertedisa fact, i.e. ?- $assertedisa(obj,?V). i.e. extensional ?- obj:?V.
axiomtext	text of rules, queries, and constraints
syn	local name, all representations and all synonyms (restricted to values of language property)
all	all text values (values from fields id, lid, axiomtext, syn)
repr_de, repr_en, ...	language-dependent representation, i.e. extensional value of ?- obj[_representation(lang)->?V]
docu_de, docu_en, ...	language-dependent documentation, i.e. extensional value of ?- obj[_documentation(lang)->?V]
syn_de, syn_en, ...	language-dependent synonyms, plus all language-independent synonyms i.e. extensional value of ?- obj[_synonym(lang)->?V] OR obj[_synonym->?V]
name_de, name_en, ...	language-dependent names: representation (or localname as fall back if representation is not defined), synonyms

Custom index fields

You can add additional fields by defining a bean of class com.ontoprise.indexer.FullTextIndexField. FullTextIndexField has the following properties:

property name	Description
fieldName	name of the index field. Attribute 'languageAware' (boolean): If true, the fieldName is expanded if it contains the languagePattern.
fieldType	name of field type as defined in fulltextindex-schema.xml
ontobrokerInternal	false, default=false: Flag to mark field to be important for various OntoBroker functionalities (SearchHelper, AutocompleteHelper)
attributeNames	set of property names whose value should be included in this field. If an attributeName contains the value of the property languagePattern, it is replaced by a concrete language code.
languagePattern	pattern to be replaced by the language values (see general options above). In this case this field definition is translated into multiple fields.
restrictedToInstancesOf	set with concept ids. The field is only created for an object if it is an asserted instance of one of the given concept ids
indexed	false: Should field be indexed? (overwrites value from field type)
tokenized	false: Should values be tokenized? (overwrites value from field type)
stored	false: Should value be stored in index,i.e. values can be retrieved on search? (overwrites value from field type)
binary	false: is this a binary field? (overwrites value from field type)
compressed	false: should stored text field be compressed? (overwrites value from field type)
omitNorms	false: advanced option: omit norms associated with this field (changes ranking). (overwrites value from field type)
omitTermFreqAndPositions	false: omit term freq, positions and payloads from postings for this field? (overwrites value from field type)
termVectors	false: expert field option (see Lucene documentation for details)
termPositions	false: expert field option (see Lucene documentation for details)
termOffsets	false: expert field option (see Lucene documentation for details)

Configuration example

```
  ...

      <bean class="com.ontoprise.indexer.FullTextIndexField">
          <property name="languagePattern" value="$LANGUAGE$" />
          <property name="fieldName" value="xname_$LANGUAGE$" />
          <property name="fieldType" value="otext" />
          <property name="stored" value="true" />
          <property name="attributeNames">
             <set>
                 <value>&lt;http://my.namespace#name&gt;("$LANGUAGE$")</value>
             </set>
          </property>
      </bean>
  ...
```

In this example, one custom index field definition with a language pattern is defined. This means that multiple index fields are created, one for each language defined in the property languages (see global options above). Assuming that the defined languages are "de" and "en", the effective index fields are "xname_de" containing the values of the attribute <http://my.namespace#name>("de"), and "xname_en" containing the values of the attribute <http://my.namespace#name>("en"). The field type used here is "otext" , this is a standard field type used by many standard index fields of OntoBroker. The values are stored

in the index for retrieval (this is only needed if it is used for autocompletion or if you want to use the return() option of the _queryIndex built-in)

Search example

To search in this field with _queryIndex/10, use the Lucene syntax for fields (here "xname_en:") in front of your search text.

```
?- _queryIndex(<http://your.company.com#ontology>, [return(xname_en)], "xname_en:
Transp*", 0, 10, ?OBJ, ?TC, ?SCORE, ?ORDER, ?OPT).
```

Autocomplete example

For autocomplete you have to set the search field in the AutoComplete.

```
AutocompleteHelper helper = ...
Ontology ontology = ...
AutocompleteHelper.Type type = ...
AutocompleteHelper.Options options = new AutocompleteHelper.Options();

options.setSearchField("xname_en");

options.set...

CompletionResults results = helper.getCompletion(ontology, options, type, "Transp",
0, 10);
..
```

21.7.12 Connector Cache Options

Connector.Database.Cache.Lifetime	query | unlimited
Connector.SPARQL.Cache.Lifetime	query | unlimited
Connector.OntoBroker.Cache.Lifetime	query | unlimited
Connector.Excel.Cache.Lifetime	query | unlimited
Connector.WebService.Cache.Lifetime	query | unlimited

`query` - tuple cache is removed after the query execution. This is the default value for this option for all connectors in OntoStudio and for the WebService connector in Ontobroker.

`unlimited` - tuple cache is not removed after the query execution and can be used for the next query. This option is better for performance and should always be used if a database won't be updated frequently. The cache update should be done manually (see "Clear connector cache command"). This is the default value for this option in OntoBroker for all connectors except the WebService connector.

Logging Configuration 279

21.8 Auditing

OntoBroker includes an audit feature which allows to track changes to the ontologies in an OntoBroker server was well as user actions. These operations are logged to a file which has an ObjectLogic format and can be read by OntoBroker. Auditing is not intended to be used directly with OntoStudio, i.e. with non Collab Server projects. This means that auditing must be turned off in the OntoStudio OntoConfig.prp (which is the default).

The audit feature can be configured by these OntoConfig options:

```
Auditing = off|on|detailed
```

If 'off' is choosen, the audit feature is disabled. 'on' and 'detailed' will turn auditing on. In this case the value is also the default value for all operations which are designated for auditing. Single operations can be configured by

```
Auditing.EventType.<EventTypeName> = off|on|detailed
```

and

```
Auditing.UserAction.<UserActionName> = off|on|detailed
```

Available event type and user action names are mentioned in the default OntoConfig.prp. For their meaning refer to the ObjectLogic ontology audit.obl, which is contained in <OntoBroker installation directory>/config/auditing. This ontology also contains the ObjectLogic schema which is used to log the operations to the audit file. The audit file name can be choosen by

```
Auditing.File = <filename>
```

If the file size exceeds a limit, which can be configured by

```
Auditing.MaxFileSize = <limit>
```

, then the audit is rolled over, i.e. the current audit file is renamed to a backup file and the auditing continues with a new (empty) audit file named by Auditing.File.

A maximum amount of backup files can be specified by

```
Auditing.MaxBackupIndex = <max-index>
```

The audit can be manually rolled over by using the OntoBroker command

```
auditRollOver
```

, which has no further parameters.

21.8.1 Logged Information

OntoBroker notifies about changes to its datamodel by events. This events and additional user actions can be logged. Logged objects include information about user, host and session id (if available, which holds if OntoBroker is configured to require logins). Each logged object has a timestamp including time zone information.

Events

This is a list of logged events:

Event	Included information
Ontology manager opened I.e. OntoBroker server startup.	• OntoBroker version This event doesn't include user, host or session related information.
Ontology manager closed I.e. OntoBroker server shutdown.	- This event doesn't include user, host or session related information.
Ontology created	• ontology URI
Ontology opened	• ontology URI • The axioms of the opened ontology iff the audit level for this event type is 'detailed'
Ontology content imported I.e. the content of an ontology is added to a given ontology	• ontology URI (into which the content is imported) • imported ontology URI • The axioms of the imported ontology iff the audit level for this event type is 'detailed'
Ontology closed	• ontology URI
Ontology deleted	• ontology URI • The axioms of the deleted ontology iff the audit level for this event type is 'detailed'
Ontology renamed	• (old) ontology URI • new ontology URI • All old axioms of the renamed ontology iff the audit level for this event type is 'detailed'

	• All new axioms of the renamed ontology iff the audit level for this event type is 'detailed' Axioms contain the information to which ontology they belong to. By changing the URI of an ontology, all contained axioms are changed implicitly.
Ontology import relationship changed	• ontology URI (of importing ontology) • ontology URI of imported ontology • change type: add/remove
Term replaced	• ontology URI • replaced term • new term • All changed axioms iff the audit level for this event type is 'detailed'
Axioms updated	• ontology URI • list of changed axioms For each axiom in the list: • change type: add/remove • axiom text • predicate, arguments, module, if the axiom is a literal or F-atom
Bulk axioms updated I.e. axioms are updated by patterns.	• ontology URI • list of bulk changes For each bulk change in the list: • change type: add/remove • axiom text for query and update formula • predicate, arguments, module for query and update formula, if the respective formula is a literal or F-atom • All added/removed axioms iff the audit level for this event type is 'detailed'
All axioms removed	• ontology URI • All axioms previously contained in the ontology iff the audit level for this event type is 'detailed'
Namespace prefix changed	• ontology URI • prefix • change type: add/remove • namespace, if change type is 'add' • old namespace value, if any
Materialized rules	• materialized rules/standard axioms • flag indicating if the materialized rules have been removed from the ontology • All added facts thru the materialization as well as all removed rules, if any, iff the audit level for this event type is 'detailed'.

User Actions

This is a list of logged user actions.

User action	Included information
Opened session	-
Login attempt	- Indicates a login attempt which failed. Doesn't include session information, since there is none.
Closed session	-
Executed command	• command text

OntoStudio

Auditing is an OntoBroker feature. If using OntoStudio to access an OntoBroker server, the following points should be considered.

An OntoStudio operation like deleting a class A may trigger multiple OntoBroker events, e.g. "removed all instances of class A", "removed all sub classes of A", "removed all super classes of A". I.e. an OntoStudio operation does not generate an atomic log entry but may be reflected by several OntoBroker change events.

If auditing is enabled, the ObjectLogic Source Editor of OntoStudio gets deactivated.

21.8.2 Step by step guide

This chapter helps you to set up the logging functionality of OntoBroker. Follow the steps described below.

Note:

For OntoBroker startup use a clean config directory (this means a new folder containing an ontoconfig file without any subdirectories).

1. Browse to the OntoBroker installation directory [ONTOBROKER -> conf -> auditing] and copy the contents of the "log4j.properties.obl.mixin" file.
2. Browse to the OntoBroker configuration directory [ONTOBROKER _CONFIG -> conf] and paste the contents of the "log4j.properties.obl.mixin" file into the "log4j.properties". If there is no log4j.properties in the configuration directory, copy it from the installation directory.
3. Comment the following lines:

```
# Enable event logging.
log4j.category.[EVENT]=DEBUG, auditing
log4j.additivity.[EVENT]=false

# To log user actions uncomment the following lines.
log4j.category.[USER_ACTION]=DEBUG, auditing
log4j.additivity.[USER_ACTION]=false

# Set up renderers.
log4j.renderer.org.semanticweb.kaon2.api.event.DatamodelEvent=com.ontoprise.util.
logging.ObjectLogicDatamodelEventRenderer
log4j.renderer.com.ontoprise.util.logging.useraction.UserAction=com.ontoprise.util.
logging.ObjectLogicUserActionRenderer

# Log to the file log/auditing.obl
log4j.appender.auditing=org.apache.log4j.RollingFileAppender
log4j.appender.auditing.MaxFileSize=5MB
log4j.appender.auditing.MaxBackupIndex=20
log4j.appender.auditing.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.auditing.layout.ConversionPattern=%m%n
log4j.appender.auditing.File=log/auditing.obl
```

4. Start the OntoBroker Admin Console in start-up mode "Collaboration Server" (the same user as the OS Collaboration user. The default is "Administrator").
5. Start OntoStudio and chose "Collaboration Server" as the storage type, set up a new project.
6. Execute the ping command in the OntoBroker Admin Console.
7. Create a new ontology (e.g. http://www.example.com/auditing) in your already existing project in OntoStudio (see step 5.).
8. Create a class "Person", set a property (name, range = string), and create an instance (e.g. value = Robert).
9. Delete the class "Person"
10. Create another new ontology (e.g. http://www.example.com/importedOntology)
11. Import the ontology from step 10. to the ontology created in step 7.
12. Create a class with a typo (e.g. Cra in place of Car) in http://www.example.com/importedOntology.
13. Rename the class "Cra" to "Car"
14. Delete the imported ontology in the http://www.example.com/auditing ontology.
15. Completely remove the http://www.example.com/importedOntology ontology (also from the repository).
16. Remove the whole project from OntoStudio (do not remove the ontologies from the repository in this case!).
17. Shut down OntoBroker
18. Create a new file-based project (e.g. auditing).
19. Import the log/auditing.obl ontology from the configuration directory to the project created in step 18.
20. Import the "event-schema.obl" and "user-actions-schema.obl" ontologies from [ONTOBROKER -> conf -> auditing]
21. Import the "defaultModule" into the " event-schema.obl" ontology
22. Run the query "getDatamodelEvents"

Note:
All of the logged actions can be found as instances in the "defaultModule" ontology!

Note:
All logged actions can be found as instances in the "defaultModule" ontology!

21.9 Changelog

For relevant changes in OntoBroker 6.3 please take a look at the README.TXT file in the Ontobroker installation directory.

Index

- Q -

- R -

- S -

- T -

- W -

Legal Notice

semafora systems GmbH
Wilhelm-Leuschner-Str. 7
64625 Bensheim
Germany

Disclaimer

All of the data and settings in this program have been checked and tested extensively. Despite taking great care and making extensive technical checks, we cannot guarantee absolute accuracy, or completely correct contents. No responsibility will be taken for technical errors and incorrect information or for their consequences, e.g. effects on other programs. We are grateful to be informed of any errors at any time.
The information in this document reflects the level of information available at the time of going to press. Any necessary corrections will be covered by subsequent versions.
semafora systems GmbH does not accept any responsibility or liability for changes caused by circumstances for which they are not responsible.
We will accept no liability for problems with the Application Time Tracking caused by incorrect usage or for any complications caused by third-party software.

Copyright

Bensheim, May 2020