Stardog Docs

Introduction

Stardog is a **fast** (/docs/performance), lightweight, commercial RDF database for mission-critical apps: it uses **SPARQL** (http://www.w3.org/TR/rdf-sparql-query/) [1] (#note-1); HTTP and the SNARL protocol for remote access and control; **RDF** (http://www.w3.org/RDF/) as a data model; and **OWL 2** (http://www.w3.org/TR/owl2-overview/) for inference and data analytics.

Acquiring Stardog & Support

Download (http://stardog.com/) Stardog to get started. The **Stardog support forum** (https://groups.google.com/a/clarkparsia.com/group/stardog/about), **stardog@clarkparsia.com** (mailto:stardog@clarkparsia.com), is the place to report bugs, ask questions, etc.

Acknowledgments

Thanks to all the Stardog testers, especially Robert Butler, Al Baker, Marko A. Rodriguez, Brian Sletten, Alin Dreghiciu, Rob Vesse, Stephane Fallah, John "New Model Army" Goodwin, José Devezas, Chris Halaschek-Wiener, Gavin Carothers, Brian Panulla, Ryan Kohl, Morton Swimmer, Quentin Reul, Paul Dlug, James Leigh, Alex Tucker, Ron Zettlemoyer, Jim Rhyne. Andrea Westerinen, Huy Phan.

The Essential Documentation

This is documentation for Stardog **1.0** (**19 June 2012**). Check out the **release notes** (/docs/RELEASE_NOTES.txt).

Quick Start Guide (/docs/quick-start)

Instructions to get up-and-running very quickly with Stardog. Includes a screencast that goes into more detail.

Terminology (/docs/term/)

A glossary (/docs/term/) of technical terms used throughout the Stardog docs.

Stardog Compatibility Policies: The Future of Queries, Data, and Programs (/docs/compatibility/)

A statement of the policies we will pursue in evolving Stardog beyond the 1.0 release.

Introduction (/docs/intro/)

1. What is Stardog? (/docs/intro/#stardog)
2. Installation (/docs/intro/#installation)
3. Support and Maintenance (/docs/intro/#support)
4. Report a Bug (/docs/intro/#report-bug)

Administration (/docs/admin/)

1. CLI Overview (/docs/admin/#cli)
2. Administering a Stardog Server (/docs/admin/#admin-server)
3. Administering a Stardog Database (/docs/admin/#admin-db)
4. Configuring Security (/docs/admin/#security)
5. Managing Search Indexes (/docs/admin/#search)
6. Optimizing Bulk Data Loading (/docs/admin/#optimizing-bulk-load)
7. Resource Requirements for Stardog (/docs/admin/#resource-requirements)
8. Using Stardog on Windows (/docs/admin/#using-windows)

Using Stardog (/docs/using/)

1. Using Stardog (/docs/using/#using-stardog)
2. Querying a Database (/docs/using/#query)
3. Adding and Removing (/docs/using/#add-remove)
4. Exporting a Database (/docs/using/#export)
5. Searching (/docs/using/#search)

Security (/docs/security)

1. The User & Security Model (/docs/security/#model)

2. Command-line Interface (/docs/security/#cli)
3. Programmatic Access (/docs/security/#api)
4. Deploying Stardog Securely (/docs/security/#deployment)

Integrity Constraint Validation (/docs/sdp)

1. Background & Terminology (/docs/sdp/#intro)
2. Validating RDF with Closed World Integrity Constraints (/docs/sdp/#validation)

OWL 2 Reasoning (/docs/owl2)

1. Using Stardog's Reasoning Capabilities (/docs/owl2/#reasoning)
2. Not Getting Expected Answers? (/docs/owl2/#trouble)
3. Known Issues (/docs/owl2/#issues)
4. Guidelines (/docs/owl2/#guidelines)
5. Technical Background (/docs/owl2/#query)

Programming with Java (/docs/java/)

1. Introduction (/docs/java/#intro)
2. Creating and Administering Databases (/docs/java/#admin)
3. Creating a Connection String (/docs/java/#connection)
4. Security in Stardog (/docs/java/#security)
5. Using SNARL (/docs/java/#snarl)
6. Using Sesame (/docs/java/#sesame)
7. Using Jena (/docs/java/#jena)
8. Client-Server Stardog (/docs/java/#client-server)
9. Embedded Stardog (/docs/java/#embed)
10. Connection Pooling (/docs/java/#pool)
11. Deprecation & Backward Compatibility (/docs/java/#deprecation)

Network Programming (/docs/network/)

1. SPARQL Protocol: HTTP (/docs/network/#http)
2. Extended HTTP Protocol (/docs/network/#extended-http)
3. SNARL: The Native Stardog Remote API (/docs/network/#snarl)

Programming with Spring (/docs/spring)

1. **Introduction** (/docs/spring/#intro)
2. **Building Spring for Stardog** (/docs/spring/#building)
3. **Overview** (/docs/spring/#overview)
4. **Use** (/docs/spring/#use)
5. **Examples** (/docs/spring/#examples)

Known Issues

As of **1.0**, the known issues include:

1. When creating a database through Stardog CLI with one or more input files, the operation will succeed even if some of those files fail to be loaded. Stardog will load all the files it can and continue with creating the index, even if there are no triples loaded. An error message will be printed on the console for each file that failed to load.
2. If relative URIs exist in the data files passed to create, add, or remove commands, then they will be resolved using the constant base URI `http://api.stardog.com/` iff the format of the file allows base URIs. Turtle and RDF/XML formats allows base URIs but N-Triples format doesn't allow base URIs and relative URIs in N-Triples data will cause errors.

Notes

1. Stardog 1.0 supports SPARQL; Stardog 1.1 will support SPARQL 1.1. ↩ (#r1)

Stardog Introduction ⌂ (/docs/)

What is Stardog?

Stardog is a **fast** (/docs/performance), lightweight, commercial RDF database for mission-critical apps: it uses **SPARQL** (http://www.w3.org/TR/rdf-sparql-query/) [1] (#note-1); HTTP and the SNARL protocol for remote access and control; **RDF** (http://www.w3.org/RDF/) as a data model; and **OWL 2** (http://www.w3.org/TR/owl2-overview/) for inference and data analytics.

Compatibility & Safety

Stardog is currently at **1.0**, and since 1.0, we provide a **compatibility statement** (/docs/compatibility) regarding our policies for the various parts of the Stardog platform.

Licensing

Evaluation

Your use of Stardog **1.0** for evaluation is governed by the license terms distributed with the software; those terms basically require you to refrain from distributing Stardog to anyone outside your organization.

Stardog 1.0 Editions

Stardog 1.0 is available in Developer, and Enterprise editions. We are also planning on a Community edition which will be a feature subset of Enterprise, but will be available for use without license fees.

Learn **more** (/docs/dotcom/) about the differences in these editions.

Installation

Required Software

See the **Quick Start Guide** (/docs/quick-start) for an installation recipe.

Stardog is bundled with all required 3rd party jars, no other downloads are required. When using Stardog in your IDE, be sure to include **ALL** of the jars provided in the download.

Testing the Installation

To test that your Stardog installation is functioning properly, you can use the **command-line client** (/docs/admin/#cli) as described in the Admin chapter.

Support and Maintenance

Stardog is currently supported via the **Stardog support forum** (https://groups.google.com/a/clarkparsia.com/group/stardog/about) , **stardog@clarkparsia.com** (mailto:stardog@clarkparsia.com) , which is available to all users. All bug reports, feature requests and general feedback is welcome in this forum.

Commercial maintenance and support contracts are available for Stardog Developer and Enterprise editions as of the 1.0 release. Please **email** (mailto:sales@clarkparsia.com) if you're interested in commercial support.

Report a Bug

Please use the **Stardog support forum** (https://groups.google.com/a/clarkparsia.com/group/stardog/about) to report bugs or other issues.

Notes

1. Stardog 1.0 supports SPARQL; Stardog 1.1 will support SPARQL 1.1. ↩ (#r1)

Stardog Quick Start (/docs/)

Unix Quick Start

This guide explains the basic steps to get you started quickly with Stardog on a Unix[1] (#note-1) box.

First, tell Stardog where its home directory (where databases and other files will be stored) is:[2] (#note-2)

```
$ export STARDOG_HOME=/data/stardog
```

Second, copy the `stardog-license-key.bin`[3] (#note-3) into place:

```
$ cp stardog-license-key.bin $STARDOG_HOME
```

[4] (#note-4)

Third, start the Stardog server. By default the server will expose SNARL and HTTP interfaces—on ports 5820 and 5822, respectively.

```
$ ./stardog-admin server start
```

Fourth, create a database with an input file; use the `--server` parameter to specify which server:

```
$ ./stardog-admin create -n myDB -t D -u admin -p admin --server snarl://localhost:5820/ examples/data/University0_0.owl
```

Fifth, optionally, admire the pure RDF bulk loading power...woof!

Sixth, query the database:

```
$ ./stardog query -c http://localhost:5822/myDB -q "SELECT DISTINCT ?s WHERE { ?s ?p ?o } LIMIT 10"
```

Notes

1. We test every Stardog release extensively with various flavors of Linux and OS X. Please report a bug if you find issues on other platforms. ↩ (#r1)
2. If you're using some weird Unix shell that doesn't create environment variables in this way, adjust accordingly. Stardog requires `STARDOG_HOME` to be defined. ↩ (#r2)
3. You'll get this either with an evaluation copy of Stardog or with a licensed copy. ↩ (#r3)
4. Of course `stardog-license-key.bin` has to be readable by the Stardog process. ↩ (#r4)

Stardog Terminology (/docs/)

In the Stardog documentation, the following terms have a specific technical meaning.

Terminology

Stardog Database Management System, aka Stardog Server, aka Stardog
: An instance of Stardog: only one Stardog Server may run per JVM. A computer may run multiple Stardog Servers by running one per multiple JVMs.

Stardog Home, aka `STARDOG_HOME`
: A directory in a filesystem in which Stardog stores files and other information; established either in a Stardog configuration file or by environment variable. Only one Stardog Server may run simultaneously from a `STARDOG_HOME`.

Stardog Network Home
: A URL (HTTP or SNARL) which identifies a Stardog Server running on the network.

Database
: A Stardog database is a graph of RDF data under management of a Stardog Server. It may contain zero or more RDF Named Graphs. A Stardog Server may manage more than one Database; there is no hard limit, and the practical limit is disk space.

Database Short Name, aka Database Name
: An identifier used to name a database, provided as input when a database is created.

Database Network Name
: A Database Short Name is part of the URI of a Database addressed over some network protocol.

Index
: The unit of persistence for a Database. We sometimes (sloppily) use Database and Index interchangeably in the Stardog docs.

Memory Database
: A Database may be stored in-memory or on disk; a Memory Database is read entirely into system memory but can be (optionally) persisted to disk.

Disk Database

A Disk Database is only paged into system memory as needed and is persisted using one or more indexes.

Connection String

An identifier (a restricted subset of legal URLs, actually) that is used to connect to a Stardog database to send queries or perform other operations.

Named Graph

A Named Graph is an explicitly named unit of data within a Database. Named Graphs are queries explicitly by specifying them in SPARQL queries. There is no practical limit on the number of Named Graphs in a Database.

Default Graph

The Default Graph in a Database is the context into which RDF triples are stored when a Named Graph is not explicitly specified. A SPARQL query executed by Stardog that does not contain any Named Graph statements is executed against the data in the Default Graph only.

Security Realm

A Security Realm defines the users and their permissions for each Database in an Stardog Server. There is only one Security Realm per Stardog Server.

Stardog Admin (/docs/)

In this chapter we discuss how to administer a Stardog Server and Stardog databases, chiefly by way of discussing how to use the various Stardog command-line programs. An important part of Stardog administration is security, which is discussed in a separate, dedicated **chapter (/docs/security)** .

CLI Overview

Stardog's command-line interface (CLI) comes in two parts:

- `stardog-admin`: admininstrative client (uses SNARL Protocol only)
- `stardog`: a user's client (uses HTTP or SNARL)

The admin and user's tools operate on local or remote databases, using either HTTP or SNARL Protocols. Both of these CLI tools are Unix-only, are self-documenting, and the help output of these tools is their canonical documentation. The documentation of these tools in this chapter goes into more detail, offers background, etc. But if there is a conflict between this documentation and the output of the CLI tools' "help" command, the CLI tools' output is correct.

To use the Stardog CLI tools, you can start by asking them to display help:

```
$ stardog help
```

Or:

```
$ stardog-admin help
```

The rationale for separating functionality into two CLI programs is largely based on security, since `stardog-admin` will need to have considerably tighter access restrictions than `stardog`.

> **SECURITY NOTICE**: For usability, Stardog 1.0 provides a default user "admin" and password "admin" in `stardog-admin` commands if no user or password are given. This is obviously **not secure**; before any serious use of Stardog is contemplated, read the **security (/docs/security)** chapter at least twice, and then—minimally—change the administrative password to something we haven't published on the interwebs!

How to Make a Connection String

Some CLI subcommands require a Stardog connection string as an argument to identify the server and database upon which operations are to be performed. Connection strings are URLs and may either be local to the machine where the CLI is run or they may be on some other remote machine. There are two URL schemes recognized by Stardog: `http://` and `snarl://`. The former uses Stardog's (extended) version of SPARQL Protocol; the latter uses Stardog's native data access protocol, called SNARL, which is based on Google's Protocol Buffers.

Note: `stardog-admin` uses SNARL Protocol only; it will not work with or recognize HTTP connection strings. `stardog` user's client works with HTTP or SNARL Protocol, interchangeably.[1] (#note-1)

Example Connection Strings

To make a connection string, you need to know the URL scheme; the machine name and port the Stardog Server is running on; any (optional)

URL path to the database (it's very unlikely you'll need this); and the name of the database:

```
{scheme}{machineName}:{port}/{databaseName};{connectionOptions}
```

```
snarl://server/billion-triples-punk
```

```
http://localhost:5000/myDatabase
```

```
http://169.175.100.5:1111/myOtherDatabase;reasoning=QL
```

```
snarl://stardog:8888/the_database
```

```
snarl://localhost:1024/db1;reasoning=NONE
```

Using the default ports for SNARL and HTTP protocols simplifies connection strings. `connectionOptions` are a series of `;` delimited key-value pairs which themselves are `=` delimited. key names must be **lowercase** and their values are case-sensitive.

Administering a Stardog Server

Stardog Server is multiprotocoled: it supports both SNARL and HTTP protocols. The default port for SNARL is **5820**; the default port for HTTP is **5822**. **All administrative functions work over SNARL protocol only—creating or dropping databases; setting databases to online or offline status; creating modifying users or roles or permissions work over SNARL, not HTTP.**

To use any of these commands against a remote server, pass a `--server` argument with a SNARL URL.

Note: If you are running `stardog-admin` on the same machine where Stardog Server is running, and you're using the default protocol ports, then you can omit the `--server` argument and simply pass a database name via `-n` option. Most of the following commands assume this for simplicity.

Starting and Stopping the Server

Note: Unlike the other `stardog-admin` subcommands, starting and stopping the server may only be run locally, i.e., on the same machine as Stardog Server will run on.

The simplest way to start the server—running on the default ports, detaching to run as a daemon, and writing `stardog.pid` and `stardog.log` to the current working directory— is

```
$ stardog-admin server start
```

To specify parameters:

```
$ stardog-admin --pidfile=stardog.pid --logfile=stardog.log server start --http=80
```

To shut down the server:

```
$ stardog-admin server stop
```

To stop a server and pass a specific PID file:

```
$ stardog-admin --pidfile=stardog.pid server stop
```

Note: ports can be specified using the properties `--snarl` and `--http`. The HTTP interface can be disabled by using the flag `--no-http`; the SNARL interface may not be disabled.[2] (#note-2)

Locking Stardog Home

Stardog Server will lock `STARDOG_HOME` when it starts to prevent synchronization errors and other nasties if you start more than one Stardog Server with the same `STARDOG_HOME`. If you need to run more than one Stardog Server instance, choose a different `STARDOG_HOME` or pass a different value to `--home`.[3] (#note-3)

Configuring Stardog Server Behavior

Stardog Server's behavior can be configured via the JVM `stardog.home`, which sets Stardog Home, overriding the value of `STARDOG_HOME` set as an environment variable.

Stardog Server's behavior can also be configured via a `stardog.properties`—which is a Java Properties file—file in `STARDOG_HOME`. To change the behavior of a running Stardog Server, it is necessary to shut it down and restart it.

The following properties are available in `stardog.properties`:

1. `strict.parsing`: Controls whether Stardog parses RDF strictly (`TRUE`, the default) or laxly (`FALSE`)
2. `query.all.graphs`: Controls what Stardog Server queries over; if `true`, it will query over the default graph and the union of all named graphs; if `false` (the default), it will query only over the default graph.

Administering a Database

Configuring a Database

To administer a Stardog database, some config options must be set at creation time; others may be changed subsequently, while yet may never be changed. All of the config options have sensible defaults (except, obviously, for database name), so you don't have to twiddle any of the knobs till you really need to.

Configuration Options

The following are the legal configuration options for a Stardog database:

database.name
: A legal database name.

database.online
: The status of the database: online or offline. It may be set so that the database is created initially in online or offline status; subsequently, it can't be set directly but only by using the relevant admin commands.

icv.active.graphs
: Specifies which part of the database, in terms of named graphs, is checked with IC validation. to validate all the named graphs in the database.

icv.enabled
: Determines whether ICV is active for the database; if true, all database mutations are subject to IC validation (i.e., "guard mode").

icv.reasoning-type
: Determines what "reasoning level" is used during IC validation.

index.differential.enable.limit
: Sets the minimum size of the Stardog database before differential indexes are used.

index.differential.merge.limit
: Sets the size in number of RDF triples before the differential indexes are merged to the main indexes.

index.literals.canonical
: Enables RDF literal canonicalization. See **literal canonicalization** (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#CANONICAL_LITERALS) for details.

index.named.graphs
: Enables optimized index support for named graphs; speeds SPARQL query evaluation with named graphs at the cost of some overhead for database loading and index maintenance.

index.persist
: Enables persistent indexes.

index.persist.sync
: Enables whether memory indexes are synchronously or asynchronously persisted to disk with respect to a transaction.

index.statistics.update.automatic
: Sets whether statistics are maintained automatically.

index.type

Sets the index type (memory or disk).

reasoning.consistency.automatic

Enables automatic consistency checking with respect to a transaction.

reasoning.punning.enabled

Enables punning.

reasoning.schema.graphs

Determines which, if any, named graph or graphs contains the "tbox", i.e., the schema part of the data.

search.enabled

Enables semantic search on the database.

search.reindex.mode

Sets how search indexes are maintained.

transactions.durable

Enables durable transactions.

A Note About Database Status

As of Stardog 1.0, a database must be set to `offline` status before most configuration parameters may be changed. So the routine is to set the database offline, change the parameters programmatically, and the set the database to online. All of these operations may be done programmatically from CLI tools, such that they can be scripted in advance to minimize downtime.[4] (#note-4)

Summary of Configuration Options

The following table summarizes the options:

Config Option	Mutability	Default	API
database.name	false	{NO DEFAULT}	DatabaseOptions.NAME (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#NAME)
database.online	false[5] (#note-5)	true	DatabaseOptions.ONLINE (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#ONLINE)
icv.active.graphs	false	default	DatabaseOptions.ICV_ACTIVE_GRAPHS (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#ICV_ACTIVE_GRAPHS)
icv.enabled	true	false	DatabaseOptions.ICV_ENABLED (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#ICV_ENABLED)
icv.reasoning.type	true	NONE	DatabaseOptions.ICV_REASONING_TYPE (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#ICV_REASONING_TYPE)
index.differential.enable.limit	true	1000000	IndexOptions.DIFF_INDEX_MIN_LIMIT (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#DIFF_INDEX_MIN_LIMIT)
index.differential.merge.limit	true	10000	IndexOptions.DIFF_INDEX_MAX_LIMIT (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#DIFF_INDEX_MAX_LIMIT)
index.literals.canonical	false	true	IndexOptions.CANONICAL_LITERALS (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#CANONICAL_LITERALS)
index.named.graphs	false	true	IndexOptions.INDEX_NAMED_GRAPHS (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#INDEX_NAMED_GRAPHS)
index.persist	true	false	IndexOptions.PERSIST (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#PERSIST)
index.persist.sync	true	true	IndexOptions.SYNC (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#SYNC)

Config Option	Mutability	Default	API
index.statistics.update.automatic	true	true	IndexOptions.AUTO_STATS_UPDATE (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#AUTO_STATS_UPDATE)
index.type	false	Disk	IndexOptions.INDEX_TYPE (/docs/java/snarl/com/clarkparsia/stardog/index/IndexOptions.html#INDEX_TYPE)
reasoning.consistency.automatic	true	false	DatabaseOptions.CONSISTENCY_AUTOMATIC (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#CONSISTENCY_AUTOMATIC)
reasoning.punning.enabled	false	false	DatabaseOptions.PUNNING_ENABLED (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#PUNNING_ENABLED)
reasoning.schema.graphs	true	default	DatabaseOptions.SCHEMA_GRAPHS (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#SCHEMA_GRAPHS)
search.enabled	false	false	DatabaseOptions.SEARCHABLE (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#SEARCHABLE)
search.reindex.mode	false	wait	DatabaseOptions.SEARCH_REINDEX_MODE (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#SEARCH_REINDEX_MODE)
transactions.durable	true	false	DatabaseOptions.TRANSACTIONS_DURABLE (/docs/java/snarl/com/clarkparsia/stardog/DatabaseOptions.html#TRANSACTIONS_DURABLE)

Legal Values of Configuration Options

The following options take a boolean value: `database.online`, `icv.enabled`, `index.literals.canonical`, `index.named.graphs`, `index.persist`, `index.sync`, `index.statistics.update.automatic`, `reasoning.consistency.automatic`, `reasoning.punning.enabled`, `search.enabled`, `transactions.durable`.

The legal value of `database.name` is given by the regular expression `[A-Za-z]{1}[A-Za-z0-9_-]`.

The legal value of `icv.active.graphs` is a list of named graph identifiers. See `reasoning.schema.graphs` below for syntactic sugar URIs for default graph and all named graphs.

The legal value of `icv.reasoning.type` is one of the reasoning levels (i.e, one of the following strings): `NONE`, `RDFS`, `QL`, `RL`, `EL`, `DL`.

The legal value of `index.differential.*` is an integer.

The legal value of `index.type` is the string "disk" or "memory" (case-insensitive).

The legal value of `reasoning.schema.graphs` is a list of named graph identifiers, including (optionally) the special names, `tag:stardog:api:context:default` and `tag:stardog:api:context:all`, which represent the default graph and the union of all named graphs and the default graph, respectively. In the context of database configurations only, Stardog will recognize `default` and `*` as shorter forms of those URIs, respectively.

The legal value of `search.reindex.mode` is one of the strings `sync` or `async` (case insensitive) or a legal Quartz cron expression (http://www.quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/crontrigger).

Managing Database Status

Databases are either online or offline; this allows database maintenance to be decoupled from server maintenance.

Online and Offline Strategies

Databases may be put into online or offline status via one of two strategies.

For moving from offline to online:

1. `wait`: Blocks till all database startup jobs and activities are completed; database is offline till all such jobs have completed.
2. `nowait`: Database comes online more quickly, non-essential startup and other jobs are performed asynchronously.

For moving from online to offline:

1. `wait`: No new connections to the database are allowed; pending jobs are unscheduled; open connections and running jobs are allowed to finish up to a timeout value; the default timeout value is 180 seconds.
2. `nowait`: No new connections to the database are allowed; pending jobs are unscheduled; open connections and running jobs are allowed to finish up to a timeout value: the *fixed* timeout value for `nowait` is 0 seconds.

By default, the online strategy for a database is `wait`; you can set a database from offline to online via SOMETHING.

Examples

To set a database from offline to online:

```
$ stardog-admin offline -n my_db_name
```

To set the database offline without waiting:

```
$ stardog-admin offline --nowait -n my_db_name
```

To set the database online:

```
$ stardog-admin online -n my_db_name
```

If Stardog Server is shutdown while a database is offline, the database will be offline the next time the server starts.

Creating a Database

Stardog databases may be created locally or remotely; but, of course, performance is better if the data files don't have to be transferred over a network.[6 (#note-6)] All data files, indexes, and server metadata for the new database will be stored in Stardog Home. Stardog won't create a database with the same name as an existing database.[7 (#note-7)]

Minimally, the only thing you must know to create a Stardog database is a database *name*; alternately, you may customize some other database parameters and options depending on anticipated workloads, data modeling, and other factors.

Database Creation Templates

As a boon to the overworked admin or devops peeps, Stardog Server supports a feature we call "database creation templates", which is to say that you can pass a Java Properties file with config values set and with the values (typically just the database name) that are unique to a specific database passed in CLI parameters.

Examples

To create a new database with the default options by simply providing a name and a set of initial datasets to load:

```
$ stardog-admin create -n my-database input.ttl another_file.rdf moredata.rdf.gz
```

Datasets can be loaded later as well, though bulk loading at creation time is the fastest way to load data.

To create (in this case, an empty) database from a template file:

```
$ stardog-admin create -c database.properties
```

At a minimum, the configuration file must have a value for `database.name` option.

Finally, if you only want to change only a few configuration options you can directly provide the values for these options in the CLI args as follows:

```
$ stardog-admin create -n db1 -o icv.enabled=true icv.reasoning.type=QL -- input.ttl
```

Note that '--' is used in this case when -o is the last option to delimit the value for -o from the files to be bulk loaded.

Please refer to the CLI help for more details of the `create` subcommand.

Options

Options for the Stardog `create` command.

Name	Description	Arg values	Default
`--durable, -d`	If present, sets all mutation operations to database as transactionally durable; durability increases the cost of all mutation operations.[8](#note-8)		`False`
`--guard, -g arg`	Specifies that ICV guard mode should enabled for this database. Transactional writes to database that are invalid with respect to constraints will fail.	`OFF` disables guard mode; a reasoning type[9](#note-9) enables it.	Disabled
`--type, -t`	Specifies the kind of database to be created: Memory or Disk.	`M,D`	Disk
`--searchable, -e`	Specifies this database should be searchable.		None
`--index-triples-only, -i`	Specifies this database's indexes should be optimized for RDF triples (as opposed to quads) only	[it's a flag and takes no args]	
Name	Description	Arg values	Default

Index Strategies

By default, Stardog builds extra indexes for named graphs. These additional indexes are used when SPARQL queries specify datasets using `FROM` and `FROM NAMED`. With these additional indexes, better statistics about named graphs are also computed.

Stardog may also be configured to create and to use fewer indexes, if the database is only going to be used to store RDF triples, that is, will not be used to store named graph information. In this mode, Stardog will maintain fewer indexes, which will result in faster database creation and faster updates without compromising query answering performance. In such databases, quads (that is: triples with named graphs or contexts specified) may still be added to these database at any time, but query performance may degrade in such cases.

To create a database which indexes only RDF triples, set the option `index.named.graphs` to `false` at database creation time. The CLI provides a shorthand option—`-i` or `--index-triples-only`—which is equivalent.

Please note that this option can only be set at database creation time and cannot be changed later without rebuilding the database, so use this option with caution.

Differential Indexes

While Stardog is generally biased in favor of read (i.e., query) performance, write performance is also important in many applications. In order to increase write performance, Stardog may be used, optionally, with a *differential index*.

Stardog's differential index is used to persist additions and removals separately from the main indexes, such that updates to the database can be performed faster. Query answering takes into consideration all the data stored in the main indexes and the differential index; hence, query answers are computed as if all the data is stored in the main indexes.

There is a slight overhead for query answering with differential indexes if the differential index size gets too large. For this reason, the differential index is merged into the main indexes when its size reaches the `DIFF_INDEX_MAX_LIMIT`. There is no benefit of differential indexes if the main index itself is small. For this reason, the differential index is not used until the main index size reaches

`DIFF_INDEX_MAX_LIMIT`.

In most cases, the default value of the `DIFF_INDEX_MAX_LIMIT` parameter will work fine and doesn't need to be changed.[10] (#note-10)

Using Compressed Data

Stardog supports loading data from compressed files directly so there is no need to uncompress files before loading. Compressed input is actually typically faster to load since it minimizes disk access so the recommended way to load large input files is to load them in compressed format.

Stardog supports GZIP and ZIP compressions natively. If a file name passed to create or add commands (through CLI or API) will be interpreted to be a gzip file if the file name ends with '.gz'. The RDF format of the file is determined by the extension that comes before '.gz'. So, if a file named 'test.ttl.gz' is used as input, Stardog will perform gzip decompression during loading and parse the file with Turtle parser. All the formats supported by Stardog (RDF/XML, Turtle, Trig, etc.) can be used with gzip compression.

The zip support works differently since zipped files can contain multiple files inside. When an input file name ends with '.zip', Stardog performs zip decompression and tries to load all the files inside the zip file. The RDF format of the files inside the zip file is determined by their file names as usual. If there is an unrecognized file extension (e.g. '.txt') that file will be skipped.

Dropping a Database

This command removes a database and all associated files and metadata. As with `create`, this command may work against a database directly or via a running Stardog server. `drop` operates without regard to pending writes or queries. Only use `drop` when you're certain!

It takes as its only argument a valid database name. For example,

```
$ stardog-admin drop -n my_db_name
```

Using Integrity Constraint Validation

Stardog supports integrity constraint validation as a data quality mechanism via closed world reasoning. Constraints can be specified in OWL, SWRL, and SPARQL.

Please see the **ICV chapter** (/docs/sdp) for more about using ICV in Stardog.

The `icv` subcommand can be used to add, delete, or drop all constraints from an existing database. It may also be used to validate an existing database with constraints that are passed into the `icv` subcommand; that is, using different constraints than the ones already associated with the database.

For ICV in transacted mutations of Stardog databases, see the database creation section above.

Migrating a Database

The `migrate` subcommand migrates an older Stardog database to the latest 1.0 of Stardog. Its only argument is the name of the database to migrate. `migrate` won't necessarily work between arbitrary Stardog version, so before upgrading check the release notes for a new version carefully to see whether migration is required or possible.

```
$ stardog-admin migrate -n myDatabase
```

will update `myDatabase` to the latest database format.

Getting Database Information

You can get some information about a database (online/offline status, creation time, last modification time, etc.) by running the following command:

```
$ stardog-admin get -n my_db_name
```

This will return all the metadata stored about the database, including the values of configuration options used for this database instance. If you want to get the value for a specific option then you can run the following command:

```
$ stardog-admin get -o index.named.graphs -n my_db_name
```

Configuring Security

See the **Security Chapter** (/docs/security) for information about Stardog's security system, secure deployment patterns, and more.

Managing Search Indexes

Maintaining Search Indexes

Stardog's search service is described in the **Using Stardog** (/docs/using) chapter.

But managing the reindexing of search indexes is an administrative task.

There are three modes for reindexing indexes:

1. `sync`: Recompute the search index synchronously with a transacted write.
2. `async`: Recompute the search index asynchronously as soon as possible with respect to a transacted write.
3. Scheduled: Use a **cron expression** (http://www.quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/crontrigger) to specify when the search index should be updated.

This is specified when creating a database by setting the property `search.reindex.mode` to "sync", "async", or to a valid cron expression. The default is "sync".

Optimizing Bulk Data Loading

Stardog tries hard to do bulk loading at database creation time in the most efficient and scalable way possible. But data loading time can vary widely, depending on factors in the data to be loaded, including the number of unique resources, etc. We'll continue to improve bulk loading as we move to the 1.0 release; for now, here are some tuning tips that may work for you:

1. Load **compressed data files** (#compressed) since compression minimizes disk access
2. Use a multicore machine since bulk loading is highly parallelized and every index is built concurrently.
3. Load multiple files together at creation time since different files will be parsed and processed concurrently improving the load speed
4. Turn off `strict.parsing` (#jvm-arg).
5. If you are not using named graphs, use **triples only indexing strategy** (#index-strategies)

Resource Requirements for Stardog

The three system resources used by Stardog are CPU, memory, and disk. In what follows we primarily discuss memory and disk. Stardog will take advantage of multiple CPUs, cores, and core-based threads in data loading and in throughput-heavy or multi-user loads. And obviously Stardog performance is influenced by the speed of CPUs and cores. But some workloads are bound by main memory or by disk I/O (or both) more than by CPU. In general, use the fastest CPUs you can afford with the largest secondary caches and the most number of cores and core-based threads of execution, especially in multi-user workloads.

The following subsections provides more detailed guidance for the memory and disk resource requirements of Stardog.

Memory usage

Stardog uses system memory aggressively and the total system memory available to Stardog is typically the most important factor in performance. Stardog uses both JVM memory (known as heap memory) and also the operating system memory outside the JVM (known as non-heap memory). Having more system memory available is always good; however, increasing JVM memory too close to total system memory is not usually prudent as it will tend to increase Garbage Collection (GC) time in the JVM.

The following table shows minimum recommended JVM memory and system memory requirements for Stardog:

Recommended memory resources for a Stardog database.

Number of triples	JVM memory	System memory
10 million	2GB	4GB
100 million	3GB	8GB
1 billion	4GB	16GB
10 billion	8GB	64GB

By default, Stardog CLI sets the maximum JVM memory to 2GB. This setting works fine for most small to medium database sizes (up to 100 million triples). As the database size increases, we recommend increasing JVM memory. You can increase the JVM memory for Stardog by setting the system property `STARDOG_JAVA_ARGS` using the standard JVM options. For example, you can set this property to `"-Xms4g -Xmx4g"` to increase the JVM memory to 4GB. We recommend setting the minimum heap size (`-Xms` option) as close to the max heap size (`-Xmx` option) as possible.

Disk usage

Stardog stores data on disk in a compressed format. The disk space needed for a database depends on many factors besides the number of triples, including the number of unique resources and literals in the data, average length of resource identifiers and literals, and how much the data is compressed. The following table shows average disk space used by a Stardog database:

Average amount of disk space used by a Stardog database.

Number of triples	Disk space used
10 million	700MB - 1GB
100 million	7GB - 10GB
1 billion	70GB - 100GB
10 billion	700GB - 1TB

These numbers are given for information purposes and the actual disk usage for a database may be significantly different in practice. Also it is important to note that the amount of disk space needed at creation time for bulk loading data is higher as several temporary files will be created. The additional disk space needed at bulk loading time can be 40% to 70% of the final database size.

Disk space used by a database is non-trivially smaller if **triples only indexing strategy** (#index-strategies) is used. Triples-only indexing reduces overall disk space used by 25% in average; however, please note the tradeoff: SPARQL queries involving named graphs perform significantly better with quads indexing.

The disk space used by Stardog is additive for multiple databases and there is very little disk space used outside the databases. To calculate the total disk space needed for multiple databases, one can simply sum up the disk space needed by each database.

Using Stardog on Windows

Stardog provides .bat files for use on the Windows platform which provide the nearly the same set of functionality provided by the Bash scripts which are used on *nix systems. There are however, a few small differences between the two. You cannot change the logfile with the global `--logfile` option using the .bat scripts, it will always just log to the console. Additionally, when you start a server with `server start`, this does not detach to the background, it will run in the current console. To shut down the server correctly, you should either issue a `server stop` command from another window, or press CTRL-C (and then response 'Y' when asked to "Terminate batch job"). Do not under any circumstance close the window without shutting down the server. This will simply kill the process without shutting down Stardog which could cause your database to be corrupted. In the future, we will provide support for running the Stardog server as a Windows Service.

The .bat scripts for windows support our standard `STARDOG_HOME` and `STARDOG_JAVA_ARGS` environment variables which can be used to control where Stardog's database is stored and usually, how much memory is given to Stardog's JVM when it starts. By default, the script will use the JVM that is available in the directory from which Stardog is run via the `JAVA_HOME` environment variable. If this is not set, it will simply execute `java` from within that directory.

Notes

1. SNARL is faster than HTTP in cases where payloads to and from the server are relatively *small*; for payloads that are large, raw transfer time dominates and there isn't much or any difference in performance. ↩ (#r1)
2. We'll make it possible to disable the SNARL interface by the 1.0 release. ↩ (#r2)
3. We will modify this restriction so that multiple Stardog Servers can run from a single `STARDOG_HOME` by 1.0. ↩ (#r3)
4. In a future version, we will allow some properties to be set while the database remains online. ↩ (#r4)
5. This may be set at creation time so that the database is created in offline or online status; but it may not be subsequently changed directly, except by using the admin tools to move the database on- or offline. ↩ (#r5)
6. See the section below about loading compressed data. ↩ (#r6)
7. Stardog database names must conform to the regular expression, `[A-Za-z]{1}[A-Za-z0-9_-]` . ↩ (#r7)
8. In a future release, we will allow durability to be overriden or enabled in Connection Strings, rather than set once-for-all at database creation time. ↩ (#r8)
9. One of "QL", "EL", "RL", "RDFS", or "NONE". ↩ (#r9)
10. The corrollary of this claim is that you shouldn't change this value in a production system till you've tested the effects of a change in a non-production system. ↩ (#r10)

Stardog Using (/docs/)

Using Stardog

While Stardog is a full-featured RDF database, its primary purpose is to execute queries against RDF data which it has under direct management.[1] (#note-1) Stardog supports the **SPARQL** (http://www.cambridgesemantics.com/2008/09/sparql-by-example) query language, a W3C standard.

Querying a Database

Stardog currently supports all of **SPARQL** (http://www.w3.org/TR/rdf-sparql-query/); support for SPARQL 1.1 is scheduled for Stardog 1.1. To execute a SPARQL query against a Stardog database, use the `query` subcommand.

Options for the Stardog `query` subcommand.

Name	Description	Arg values	Default
`--format, -f`	Controls the format type of the query results, either RDF or variable bindings, depending on the SPARQL query form used.	RDF: `N-TRIPLES, RDFXML, TURTLE`; variable bindings: SPARQL[2] (#note-2), `JSON, TEXT`	For RDF, `??`; for variable bindings, SPARQL
`--query,-q`	An appropriately-escaped SPARQL query.	A valid SPARQL query that's appropriately escaped	None
Name	Description	Arg values	Default

For example,

```
$ ./stardog query -c "http://localhost:8989/sp2b_10k" -q "SELECT DISTINCT ?s { ?s ?p ?o } LIMIT 10"
```

That query will return up to 10 unique resource identifiers from the `sp2b_10k` database.

Known Limitations

See **Known Issues** (/docs/#bugs) for the complete list of known issues or limits in Stardog. With respect to SPARQL query evaluation:

- queries with `FROM NAMED` with a named graph that is *not* in Stardog will **not** cause Stardog to believe that it is, in fact, Maven, i.e., to automagically download the data from an arbitrary HTTP URL and include it in the query. Stardog will *only* evaluate queries over data that has been loaded into it.
- Regular expressions in SPARQL queries aren't optimized; we recommend using the **Search** (#search) service described below.
- SPARQL queries without a context or named graph are executed against the default, unnamed graph. In Stardog, the default graph is *not* the union of all the named graphs and the default graph.

Adding and Removing

The most efficient way to load data into Stardog is at database creation time. See the **Creating a Database (/docs/admin/#create)** section for bulk loading data at database creation time. To add data to an existing Stardog database, use the `add` command:

```
$ stardog add -c snarl://server/myDatabase 1.rdf 2.rdf 3.rdf
```

The optional arguments are `-f` (or `--format`) to specify the RDF serialization type of the files to be loaded; if you specify the wrong type, `add` will fail. If you don't specify a type, Stardog will try to determine the type on its own based on the file extension. For example, the files that have names ending with '.ttl' will be parsed with Turtle syntax. If you specify a type, then all the files being loaded must of that same type.

If you want to add data to a named graph, specify it via the `--graph-uri` or `-g` options.

To remove data from a Stardog database, `remove` is used by specifying—

1. one Named Graph, OR
2. one or more files containing RDF (in some recognized serialization format, i.e., RDF/XML, Turtle, Trig), OR
3. one Named Graph and one or more RDF files.

For example,

```
$ stardog remove -c snarl://server/myDatabase -g http://foo
```

will remove the named graph `http://foo` and all its triples from `myDatabase`.

```
$ stardog remove -c snarl://server/myDatabase 1.rdf
```

will remove the triples in `1.rdf` from (the default graph of) `myDatabase`.

```
$ stardog remove -c snarl://server/myDatabase -g http://foo -f TURTLE 2.rdf 3.rdf
```

will remove the triples in the Turtle files `2.rdf` and `3.rdf` from the named graph `http://foo` of `myDatabase`.

Strict or loose parsing may be set for the input payload by using `--strict-parsing=TRUE|FALSE`.

How Stardog Handles RDF Parsing

RDF parsing in Stardog is strict: it requires typed RDF literals to match their explicit datatypes, URIs to be well-formed, etc. In some cases, strict parsing isn't ideal, so it may be disabled using the `--strict-parsing=FALSE` to disable it.

However, even with strict parsing disabled, Stardog's RDF parser may encounter parse errors from which it cannot recover. And loading data in lax mode may lead to unexpected SPARQL query results. For example, malformed literals (`"2.5"^^xsd:int`) used in filter evaluation may lead to undesired results.

Exporting a Database

To export data from a Stardog database back to RDF, `export` is used by specifying—

1. the connection string of the database to export
2. the export format: `N-TRIPLES, RDFXML, TURTLE, TRIG`. default is 'N-TRIPLES'— 'TRIG' must be used when exporting the entire database if the database contains triples inside named graphs.
3. optionally, the URI of the named graph to export if you wish to export a single named graph only.
4. the file to export to

For example,

```
$ stardog export -c myDatabase --format TURTLE myDatabase_output.ttl
```

```
$ stardog export -c myDatabase --graph-uri http://example.org/context myDatabase_output.nt
```

Searching

Stardog includes an RDF-aware semantic search capability: it will index RDF literals and supports information retrieval-style queries over indexed data.

Indexing Strategy

The indexing strategy creates a "search document" per RDF literal. Each document consists of the following fields: literal ID; literal value; and contexts.

Search in SPARQL

We use the predicate `http://jena.hpl.hp.com/ARQ/property#textMatch` to access the search index in a SPARQL query.

For example,

```
SELECT DISTINCT ?s ?score
WHERE {
?s ?p ?l.
( ?l ?score ) <http://jena.hpl.hp.com/ARQ/property#textMatch> ( 'mac' 0.5 50 ).
}
```

This query selects the top 50 literals, and their scores, which match 'mac' and whose scores are above a threshold of 0.5. These literals are then joined with the generic BGP `?s ?p ?l` to get the resources (?s) that have those literals. Alternatively, you could use `?s rdf:type ex:Book` if you only wanted to select the books which reference the search criteria; you can include as many other BGP's as you like to enhance your initial search results.

Searching with the Command Line

First, check out the CLI help for the search subcommand...I'll wait.

```
$ stardog help search
```

Okay, now let's do a search over the O'Reilly book catalog in RDF for everything mentioning "html":

```
$ stardog search -c "snarl://127.0.0.1:5820/catalog" -q "html" -l 10
```

The results?

```
Index    Score    Hit
====================
0    6.422    urn:x-domain:oreilly.com:product:9780596527402.IP
1    6.422    urn:x-domain:oreilly.com:product:9780596003166.IP
2    6.422    urn:x-domain:oreilly.com:product:9781565924949.IP
3    6.422    urn:x-domain:oreilly.com:product:9780596002251.IP
4    6.422    urn:x-domain:oreilly.com:product:9780596101978.IP
5    6.422    urn:x-domain:oreilly.com:product:9780596154066.IP
6    6.422    urn:x-domain:oreilly.com:product:9780596157616.IP
7    6.422    urn:x-domain:oreilly.com:product:9780596805876.IP
8    6.422    urn:x-domain:oreilly.com:product:9780596527273.IP
9    6.422    urn:x-domain:oreilly.com:product:9780596002961.IP
```

Query Syntax

Stardog search is based on Lucene 3.4.0: we support all of the **search modifiers** (http://lucene.apache.org/java/3_4_0/queryparsersyntax.html) that Lucene supports, with the exception of fields.

- wildcards: `?` and `*`
- fuzzy: ~ and ~ with similarity weights (e.g. `foo~0.8`)

- proximities: `"semantic web"~5`
- term boosting
- booleans: `OR`, `AND`, `+`, `NOT`, and `-`.
- grouping

For a more detailed discussion, see the **Lucene docs** (http://lucene.apache.org/java/3_3_0/queryparsersyntax.html) .

Notes

1. This implies that Stardog will not retrieve data from the Web or from any other network via HTTP URLs in order to query that data. If you want to query data using Stardog, you must add that data to a new or existing Stardog database. Note: A future version of Stardog will support **SDQ** (http://weblog.clarkparsia.com/2011/03/07/sdq-information-integration-i n-the-real-world/) , a distributed query system, that will lift this restriction. ↩ (#r1)

2. This is the XML results format. ↩ (#r2)

Stardog Security (/docs/)

Stardog seeks a balance between security and usability: data in Stardog must be secured from unauthorized access, and using Stardog should be fun and make people more productive. In this chapter we explain how security works in Stardog, including how to securely deploy it in production environments.

The User & Security Model

Stardog security and user model is standard role-based access control. Stardog uses **Apache Shiro** (http://shiro.apache.org/) for authentication, authorization, and session management; for cryptography, Stardog uses jBCrypt.

Resources

A resource is some Stardog entity or service to which access is to be controlled; in Stardog 1.0, the resource types are `user`, `role`, `database`, and `database metadata`.[1] (#note-1) A resource must have an identifier[2] (#note-2) that is unique among resources of that type.[3] (#note-3) Stardog frequently names a resource using both its type and identifier separated by a colon to avoid ambiguity: for example, `user:admin` and `db:test`. Stardog also has a notion of a "wildcard" resource; i.e., a resource that identifies all resources of a specified type. This is syntactic sugar that will make the database admin's life easier: `user:*` refers to all users, which allows, say, the granting of permissions to all users.

In order to change permissions to a given resource (i.e., grant or revoke permissions), the user must have a permission to the `grant` action to that resource. Effectively, having `grant` permission makes the user an owner of that resource; after all, if a user is missing any permissions to that resource, they always can be granted.

In general, if a resource is created in the system, its creator receives the *standard* set of permissions: `read`, `write`, `delete`, `grant`.[4] (#note-4)

Permissions

A permission is a relation between a user or a role, an action (e.g., write), and a resource (e.g., database `foo`).

Roles

Sometimes called "groups" in other systems, a role is an abstraction over a set of users; rather than granting permissions to user accounts directly, most permissions are granted to some role, from which users (which have that role) inherit the permission.

The actions on role resources are as follows:

`read`
: Permits reading all of the role's properties; for example, all the users that belong to it.

`write`
: Permits changing the role's properties.

`create`
: Permits creation of a new role.

`list`
: Permits listing all of the existing roles.

`grant`
: Permits granting or revoking permissions to perform actions on a role.

Assigning a user to a role requires `write` permission to the user and to the role.

Users

A user is an account in Stardog, which typically (but not always) is tied to a person. Users belong to roles and own resources. Users can belong to multiple roles and inherit all the permissions assigned to the roles. Permission inheritance is additive only: if a role grants a specific permission, and a user (say, `Bob` has that role, there is no way to deny that permission to `Bob`.

Users have state: they can be disabled or enabled, for example. Note: the system will not allow you to disable the last superuser in the system. A user account's name is the identifier of the user resource: so a user `Bob` is referred to as `user:Bob`. Once a user account is created, it is not possible to change the user identifier.

The actions on user resources are as follows:

`read`
: Permits reading the properties and other info of the user: is user a superuser, is it enabled or disabled, what roles does it belong to, when was it created, etc.

`write`
: Permits modifying the *modifiable* properties of the user: enabing or disabling it, changing its roles, etc.

`create`
: Permits creating a new user resource.

`delete`
: Permits removing a user resource.

`list`
: Permits retrieving a list of all user resources in the system.

`grant`
: Permits granting or revoking permissions to any of the preceding actions on a user resource to a user resource.

Some restritions apply: only a superuser can create a new user as a superuser; it's not possible to change the superuser property of a user. Changing a user's password is also subject to special rules: a user can always change its own password, and a superuser can change another users' password.

A user account initially has no password: it has to be set explicitly. Until the password is set, it is not possible to log into the user account.

Finally, assigning a user to a role requires `write` permission to the user account and to the role.

Superuser

A user account may be marked as a superuser account, in which case that user has permissions to perform all actions on any and all resources. A user account may neither be demoted from nor promoted to superuser status at any time except when the user account is created.

Database & Database Metadata

A database resource is simply a Stardog database; for each database resource, there is a corresponding database metadata resource which controls some aspects of the database and its services.

The actions on database resources are as follows:

`read`
: Permits reading data from the database, including querying the database.

`write`
: Permits modifying the contents of the database in any way, including deletions.

`create`
: Permits creating a new database in the system.

`delete`
: Permits deleting (or "dropping") the database from the system.

`list`
: Permits retrieving a list of all the databases in the system.

`grant`
: Permits granting or revoking permissions to perform any of these actions on the database.

The database metadata resource represents all of the options or metadata that can be set on a database itself, as opposed to the data contained in the database. When a database is created—called, in this example, `myDatabase`—Stardog creates a database metadata resource, which contains three resources:

- `metadata:myDatabase:options`—a container for modifiable database options; see **Foo Chapter 0** for more information about the modifiable database options.
- `metadata:myDatabase:icv-constraints`—the set of Integrity Constraint Validation constraints for the database
- `metadata:myDatabase:icv-guard`—whether or not ICV Guard Mode can be enabled or disabled and which reasoning level guard is used with

The first resource controls whether a user or role can modify the value of the modifiable database options; the second controls access to the ICV constraints for `myDatabase`; the third controls whether ICV guard mode can be enabled or disabled.

Actions

A fixed set of operations that may be performed upon a resource; in Stardog 1.0, the actions are `read, write, create, delete, list, grant`.

Default Security Configuration

Out of the box, the Stardog security setup is minimal:

- `user:admin` with password set to "admin" is a superuser.

- `user:anonymous` with password "anonymous" has the "reader" role.
- `role:reader` allows reading from all resources.

The rationale for `user:anonymous` is to support, out of the box, the use case of a public SPARQL endpoint that allows access to multiple databases, while not allowing the user to have excessive privileges.

Authentication and Authorization

To be documented: how to extend Stardog to allow 3rd party authentication services.

Password Policies

To be documented: how to implement custom password policies in Stardog.

Command Line Interface

Most of security-related commands are in the `stardog-admin` command, since most users don't need to manage security. In general, all options should accept `-u username` option (the user performing the operation) and either `-p password` (to pass the password on the command line; not secure but useful for scripting) or `-P` to force the CLI to prompt for the password.

If no username is passed, then user `admin` with password `admin` is assumed.

Similarly, all admin options accept the `--server` parameter to specify the URL of the Stardog server.

In the examples below, we generally don't show these options and parameters to avoid clutter, unless a full command is shown.

Note: the name of the user performing the operation should not be confused with a user that may be the object of the operation (e.g., when changing a password of a different user).

User Management

To create a new user:

```
$ stardog-admin user add -n newusername -u admin -p admin --server "snarl://server/"
```

To make the new user a superuser, pass the `-s` flag.

To assign a password to the user `bob`:

```
$ stardog-admin passwd -n username -u admin -p admin --server "snarl://server/"
```

To remove a user, `frank`:

```
$ stardog-admin user drop -n frank -u admin -p admin--server "snarl://server/"
```

To list users:

```
$ stardog-admin user list -u admin -p admin --server "snarl://server/"
```

To list more information about users,[5] (#note-5) pass the `-a` flag:

```
$ stardog-admin user list -a -u admin -p admin --server "snarl://server/"
```

To add roles (`role1` and `role2`) to a user, `dahlia`:

```
$ stardog-admin user edit -R "role1,role2" -n dahlia -u admin -p admin --server "snarl://server/"
```

To replace dahlia's roles with new roles (or to remove a user's roles, pass `--replace -R ""`):

```
$ stardog-admin user edit --replace -R "role3,role4" -n dahlia -u admin -p admin --server "snarl://server/"
```

To disable a user's account (note: you can never disable (or remove) the last superuser account):

```
$ stardog-admin user edit -n dahlia --enabled false -u admin -p admin --server "snarl://server"
```

Role Management

To add a new role, `spyboy`, to Stardog:

```
$ stardog-admin role add -n spyboy -u admin -p admin --server "snarl://server"
```

To delete a role from Stardog:

```
$ stardog-admin role drop -n spyboy -u admin -p admin --server "snarl://server"
```

To grant a permission (modifying the ICV Constraints resource of database `test2`) to a role, `spyboy`:

```
$ stardog-admin role grant -n spyboy -a write -o "metadata:test2:icv-constraints" -u admin -p admin --server "snarl://serve
```

Dropping a role will fail if there are users who belong to that role. Pass `-f` or `--force`, which unassigns users from the role before removing it.

To list roles (short version):

```
$ stardog-admin role list -u admin -p admin --server "snarl://server"
```

And the long version, which includes users who belong to the role:

```
$ stardog-admin role list -a -u admin -p admin --server "snarl://server"
```

Permissions Management

You can list a user's assigned and inherited permissions; or you can list the permissions of a role. The syntax is the same for both; they accept `-n` to specify the user or role, respectively, for which the permissions should be listed.

The output is written in tabular form, grouping permissions to the same resource in a single line; for example, for user admin:

```
$  stardog-admin user permission -n admin -u admin -p admin --server "snarl://server"
```

Which returns:

```
anonymous permissions
+---------------+----------------------+-------------+-------------+
| Resource Type |    Resource Name     | Permissions |   Source    |
+---------------+----------------------+-------------+-------------+
| *             | *                    | -R----      | reader      |
| db            | *                    | C-----      | [anonymous] |
| db            | lubm                 | -RWD-G      | lubmadmins  |
| metadata      | lubm:icv-constraints | -RWD-G      | [anonymous] |
| metadata      | lubm:icv-guard       | -RWD-G      | [anonymous] |
| metadata      | lubm:options         | -RWD-G      | [anonymous] |
| user          | *                    | C---L-      | useradmins  |
+---------------+----------------------+-------------+-------------+

useradmins permissions

+---------------+---------------+-------------+
| Resource Type | Resource Name | Permissions |
+---------------+---------------+-------------+
| user          | *             | C---L-      |
+---------------+---------------+-------------+
```

The permissions are compactly displayed, one letter for permission (CRWDLG);[6] (#note-6) if a user does not have a permission, a `-` is displayed in that position. For the `anonymous` user who belongs to the `reader` group, the output is `-R----`.

Granting Permissions

There are four subcommands to modify permissions: `user grant`, `user revoke`, `role grant`, and `role revoke`.

They accept the `-n` parameter to specify the user or role to which the permission will be modified. They also accept an `-a` parameter to specify the action (e.g., `-a "write"`) and an `-o` parameter for the resource to which the permission applies (e.g., "db:foo").

To grant user `jim` permission to write to database `test2`:

```
$  stardog-admin user grant -n jim -a write -o "db:test2" -u admin -p admin --server "snarl://server"
```

To revoke permission to read from database `test3` from role `spyboy`:

```
$  stardog-admin role revoke -n spyboy -a read -o "db:test3" -u admin -p admin --server "snarl://server"
```

Programmatic Access

See the **Java Programming** () chapter.

Deploying Stardog Securely

Securing Stardog in a Networked Environment

To ensure that Stardog's RBAC access control implementation will be effective, all non-administrator access to Stardog databases should occur over network (i.e., non-native) database connections.[7] (#note-7) To ensure the confidentiality of user authentication credentials when using remote connections, the Stardog server should only accept connections that are secured with SSL. This section describes how Stardog can be configured to use SSL for data confidentiality and server authentication. It does not address using SSL for client authentication.[8] (#note-8)

Configuring Stardog to use SSL

Stardog's HTTP server does not include support for SSL; it must be deployed with other components to provide SSL support. The two primary ways to accomplish such a deployment are both described below: HTTPS reverse proxying; and SSL-enabled application server.

HTTPS Reverse Proxying

An HTTPS reverse proxy[9] (#note-9) may be used to secure Stardog client-server connections if the Stardog server is run using the command-line tool or deployed as a servlet. In the following two sections, we describe how to use **Apache** () and **lighttpd** () as HTTPS reverse proxies for Stardog. These configurations can be used for new reverse proxy deployments or can be modified to augment existing reverse proxies with SSL.[10] (#note-10)

In this deployment approach, the network connection between Stardog clients and the proxy server is secured using SSL. But the connection between the proxy server and Stardog server is insecure; thus, *care should be taken to ensure that proxy-Stardog connections only occur over trusted networks.* Note also that non-SSL connections to the Stardog server from hosts other than the proxy server should be prohibited in order to prevent network exposure of user credentials and data.[11] (#note-11)

HTTPS reverse proxying depends on having a certificate and private key on the proxy server. A cheap and easy deployment strategy is to use a self-signed certificate. Creating such a certificate is documented elsewhere adn not repeated here.[12] (#note-12) Alternatively, an SSL cert can be obtained from a commercial certificate authority.

Reverse Proxy with lighttpd

lighttpd (http://lighttpd.net) can be configured to provide an SSL layer for remote connections. The following lighttpd configuration file is a complete example that lets clients use HTTPS connections with the lighttpd proxy to communicate with a Stardog HTTP server listening on port `12345` of the lighttpd host.

```
server.port = 443
ssl.engine = "enable"
ssl.pemfile = "server.pem"
server.modules = ( "mod_proxy" )
proxy.server = ( "" =>
        ( ( "host" => "127.0.0.1" , "port" => "12345" ) )
        )
server.document-root = "/dev/null"
```

This configuration directs lighttpd to use the certificate and private key in server.pem for SSL connections.[13] (#note-13)

Reverse Proxy with Apache 2

Apache httpd can be configured to provide an SSL layer for remote connections. The following partial configuration file[14] (#note-14) allows clients to use HTTPS connections with the Apache proxy, which communicates with a Stardog HTTP server listening on port `12345` of the Apache host.

```
SSLEngine       On
SSLCertificateFile      server.pem
<Directory />
        SSLRequireSSL
</Directory>
ProxyPass       /       http://127.0.0.1:12345/
```

This configuration depends on the SSL certificate and private key being located in the `server.pem` file in the Apache server root. It also depends on `mod_ssl`, `mod_proxy`, and `mod_proxy_http` modules being compiled into the httpd binary or loaded via directives elsewhere in the configuration file.

SSL-Enabled App Server

Of course Stardog may also be deployed as a servlet in a servlet container or app server that can provide SSL support. For example, if Stardog is deployed into a default Resin Server,[15] (#note-15) then the following configuration would enable SSL on the server using the certificate and private key stored in the Java KeyStore at `server-keystore.jks`.

```
<http address="*" port="443">
        <jsse-ssl>
                <key-store-file>server-keystore.jks</key-store-file>
                <password>********</password>
        </jsse-ssl>
</http>
```

Other Java app servers support SSL including GlassFish, Tomcat, and JBoss. The configuration of SSL for each application server is implementation specific, so users should consult the relevant server's documentation.

Configuring Stardog Client to use SSL

The Stardog HTTP client driver directly supports SSL when the `https:` scheme is used in the database connection string. For example, the following invocation of the Stardog command line utility will initiate an SSL connection to a remote database

```
$ stardog status -c https://stardog.example.org/sp2b_10k
```

If the client is unable to authenticate the server, then the connection will fail and an error message like the following will be generated.

```
Error during connect.  Cause was SSLPeerUnverifiedException: peer not authenticated
```

The most common cause of this error is that the server presented a certificate that was not issued by an authority that the client trusts. The Stardog HTTP client driver uses standard Java security components to access a store of trusted certificates. By default, it trusts a list of certificates installed with the Java runtime environment, but it can be configured to use a custom trust store.[16] (#note-16)

The client driver can be directed to use a specific Java KeyStore file as a trust store by setting the `javax.net.ssl.trustStore` system property. To address the authentication error above, that trust store should contain the issuer of the server's certificate.Standard Java tools can create such a file. The following invocation of the `keytool` utility creates a new trust store named `my-truststore.jks` and initializes it with the certificate in `my-trusted-server.crt`. The tool will prompt for a passphrase to associate with the trust store. This is not used to encrypt its contents, but can be used to ensure its integrity.[17] (#note-17)

```
$ keytool -importcert  -keystore my-truststore.jks \
       -alias stardog-server -file my-trusted-server.crt
```

The following Stardog command line invocation uses the newly created truststore.

```
$ STARDOG_JAVA_ARGS="-Djavax.net.ssl.trustStore=my-truststore.jks" \
       stardog status -c https://stardog.example.org/sp2b_10k
```

For custom Java applications that use the Stardog HTTP client driver, the system property can be set programmatically or when the JVM is initialized.

The most common deployment approach requiring a custom trust store is when a self-signed certificate is presented by the Stardog server. For connections to succeed, the Stardog client must trust the self-signed certificate. To accomplish this with the examples given above, the self-signed certificate should be in the my-trusted-server.crt file in the keytool invocation.

A client may also fail to authenticate the server if the hostname in the Stardog database connection string does not match a name contained in the server certificate.[18] (#note-18)

This will cause an error message like the following

```
Error during connect.  Cause was SSLException: hostname in certificate didn't match
```

The client driver does not support connecting despite a mismatch, therefore the only solutions are to replace the server's certificate or modify the connection string to use an alias for the same server that matches the certificate.

Securing Stardog on Linux

This section describes one approach to installing Stardog on Linux—or another Unix-like operating system—with the goal of restricting unauthorized access to Stardog data. The approach detailed below is not the only effective way to secure a Stardog installation. Stardog administrators should customize their installation for the requirements of their environment.

In what follows, you'll see snippets of shell code. For each snippet $ represents the shell prompt and \ is the line continuation character. Some of the shell snippets make use of relative paths and are intended to be run from within directory where Stardog release file was unzipped. Many of the snippets will need to be run with elevated permissions.

Warning

Make sure that you know what you're doing before you copy any of the snippets of shell code or configuration syntax into a real Linux system. *You've been warned*.

Creating A Basic Stardog Environment

The Stardog library files should be copied from the distribution directory into a permanent location in the host system. The snippet below chooses a common location and uses a versioning string to facilitate parallel installs of different Stardog releases. An administrator may choose any location.

```
$ export STARDOG_VERSION=1.0
$ export STARDOG_LIBDIR=/opt/stardog-${STARDOG_VERSION}/lib
$ install -d ${STARDOG_LIBDIR}
$ cp -r lib/* ${STARDOG_LIBDIR}
$ chown -R root:root ${STARDOG_LIBDIR}
$ chmod -R a+r ${STARDOG_LIBDIR}
```

The Stardog command line tools should be copied from the distribution directory into a location that places them into most users' execution PATH. Execution permissions to the tools are *not* limited because access to the data directory will be *strictly limited*.

```
$ install -c -m a=rx \
./stardog /usr/bin/stardog
$ install -c -m a=rx \
./stardog-admin /usr/sbin/stardog-admin
```

The Stardog data directory stores both user data and system configuration data, including access control information. The location selected for the data directory should be reliable, large enough to meet data requirements, and secured from unauthorized access.

```
$ export STARDOG_HOME=/var/db/stardog
```

Access to the data directory is limited to the `stardog` group and the only member of that group is the `stardog` user.[19] (#note-19) The snippet below creates that user and group.

```
$ groupadd stardog
```

```
$ useradd \
-d ${STARDOG_HOME} \
-g stardog \
-s /sbin/nologin \
stardog
```

This snippet creates the data directory and limits its access to the newly created group.

```
$ install -d -o stardog -g stardog -m ug=rwx,o= \
${STARDOG_HOME}
```

Note that if a Stardog server is used to allow network access to remote Stardog clients, then the approach described here requires the server to run as a user in the `stardog` group.

An administrator can accomplish this by running the server as the `stardog` user or by adding the relevant user to the `stardog` group. For example, the following snippet adding the `tomcat` user to the `stardog` group may be needed in an environment where a Stardog HTTP server is run as an application within Tomcat.

```
$ usermod --add-to-group stardog tomcat
```

Of course the ideal deployment of the Stardog server depends on the particulars of the deployment environment, the preferences of the administrator, and the anticipated user load. The group-based permission approach provides flexibility to satisfy many alternatives.

Notes

1. We will extend the resource types to include named graph in Stardog 1.1. (#r1)
2. The lexical space of legal identifiers for users, roles, and databases is given by the regular expression, `[A-Za-z_]{1}[A-Za-z0-9_]*`. (#r2)
3. Resources of different types may have the same identifier, of course. (#r3)
4. `create` and `list` are not granted because they apply to resource types, not to resources per se. (#r4)
5. User names, superuser info, enabled-disabled state, roles. (#r5)
6. For create, read, write, delete, list, and grant. (#r6)
7. In other words, embedded or native Stardog access is inherently *insecure* and should be used accordingly. (#r7)
8. Stardog 1.0 does not support client authentication using X.509 certificates instead of passphrases. (#r8)
9. Reverse proxying may be useful beyond SSL layering—it may be used to distribute load across multiple Stardog servers. For general documentation of reverse proxying with lighttpd, see **the fine documentation** (http://redmine.lighttpd.net/wiki/lighttpd/Docs:ModProxy) ; likewise for **Apache** (http://httpd.apache.org/docs/2.2/mod/mod_proxy.html#forwardreverse) . (#r9)
10. Of course other solutions may be used; these are illustrative of the general technique and approach. (#r10)
11. Stardog's default HTTP server listens on all host interfaces and accepts all connections. If it is used, then a host-based firewall is necessary to prohibit connections from servers other than the proxy server. (#r11)
12. For example, see **the example** (http://docs.oracle.com/javaee/1.4/tutorial/doc/Security6.html) creating a certificate with the Java keytool; or **an example** (http://www.openssl.org/docs/apps/req.html) generating a self signed root certificate using the openssl req tool. (#r12)
13. lighttpd can be configured to present chaining certificates with the server certificate. This may be necessary if the server certificate is not directly signed by a trusted authority, but chains to a trusted authority. For details on this configuration see **the docs** (http://redmine.lighttpd.net/wiki/lighttpd/Docs:SSL) (the ssl.ca-file option). (#r13)
14. A complete configuration file is not provided because the minimal configuration file required by Apache is more detailed than the configuration file required by lighttpd.The configuration directives shown are those necessary to enable SSL and reverse proxying. (#r14)
15. See **Resin** (http://caucho.com/resin/) for more info; it supports SSL using JSSE in the open source version and using OpenSSL in the professional version. Resin's SSL support is **well documented** (http://www.caucho.com/resin-4.0/admin/security-ssl.xtp) . (#r15)
16. The Stardog HTTP client driver uses an X509TrustManager. The details of how a trust store is selected to initialize the trust manager are **documented** (http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager) . (#r16)
17. See the `javax.net.ssl.trustStorePassword` system property **documentation** (http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager) . (#r17)
18. The matching algorithm used is **described** (http://hc.apache.org/httpcomponents-client-ga/tutorial/html/connmgmt.html) in the Apache docs about `BrowserCompatHostnameVerifier`. (#r18)
19. Granting access permissions to members of the `stardog` group is more flexible than limiting access to a single user. For example, it may allow a Stardog network server to run as a user other than `stardog`; or it may facilitate other processes other than Stardog, i.e., data backup. (#r19)

Stardog ICV (/docs/)

Background & Terminology

Using OWL as if it were a schema or constraint language for RDF and Linked Data has several advantages:

- Unifying the domain model with data quality rules
- Aligning the domain model and data quality rules with the integration model and language (i.e., RDF)
- Being able to query the domain model, data quality rules, integration model, mapping rules, etc with SPARQL
- Being able to use automated reasoning about all of these things to insure logical consistency, explain errors and problems, etc

But the OWL 2 standard from W3C isn't designed for this usage; rather, it's primarily designed for *inferencing*.

In Stardog, you can use OWL as both a schema language for RDF and as a modeling or inference language. This chapter explains how.

If you are also interested in the theoretical background, please see the **ICV specification** (icv-specification.html), which has all the formal details.

Terminology

In the sections below, we explain the operational usage of Integrity Constraint Validation (ICV) in Stardog: it may be easier for you to understand if you read this section on terminology first.

Schema, TBox
: A schema (or "terminology box" a.k.a., TBox) is a set of statements that define the relationships between data elements, including property and class names, their relationships, etc. In practical terms, schema statements for a Stardog database are RDF Schema and OWL 2 terms, axioms, and definitions.

Data, ABox
: All of the triples in a Stardog database that aren't part of the schema are part of the data (or "assertional box" a.k.a. ABox).

Integrity Constraint
: A declarative expression of some rule or constraint which data must conform to in order to be valid. Integrity Constraints are typically domain and application specific. They can be expressed in OWL 2 (any legal syntax), SWRL rules, or (a restricted form of) SPARQL queries.

Constraints
: Constraints that have been associated with a Stardog database and which are used to validate the data it contains. Each Stardog may optionally have one and only one set of constraints associated with it.

ICV, Integrity Constraint Validation
: The process of checking whether some Stardog database is valid with respect to some integrity constraints. The result of ICV is a boolean value (true if valid, false if invalid) and, optionally, an *explanation of constraint violations*.

Closed World Assumption, Closed World Reasoning
: Stardog ICV assumes a closed world with respect to data and constraints: that is, it assumes that all relevant data is known to it and included in a database to be validated. It interprets the meaning of Integrity Constraints in light of this assumption; if a constraint says a value *must* be present, the absence of that value is interpreted as a constraint violation and, hence, as invalid data.

Open World Assumption, Open World Reasoning
: A legal OWL 2 inference may violate or satisfy an Integrity Constraint in Stardog. In other words, you get to have your cake (OWL as a constraint language) and eat it, too (OWL as modeling or inference language). This means that constraints are applied to a Stardog database *with respect to an OWL 2 profile*.

Monotonicity

OWL is a monotonic language: that means you can't ever add anything to a Stardog database that causes there to be fewer legal inferences. Or, put another way, the only way to decrease the number of legal inferences is to *delete* something. Monotonicity interacts with ICV in the following ways:

1. Adding data to or removing it from a Stardog database may make it invalid.
2. Adding schema statements to or removing them from a Stardog database may make it invalid.
3. Adding new constraints to a Stardog database may make it invalid.
4. Deleting constraints from a Stardog database *cannot make it invalid*.

Validating RDF with Closed World Integrity Constraints

The use of high-level languages (OWL 2, SWRL, and SPARQL) to validate RDF data using closed world semantics is one of Stardog's unique capabilities.

ICV and OWL 2 Reasoning

An integrity constraint may be satisfied or violated in either of two ways: by an explicit statement in a Stardog database or by statement that's been legally inferred by Stardog. This means that when ICV is enabled for a Stardog database, it has to be enabled relative to a reasoning type. The valid choices of reasoning type are any type or kind of reasoning supported by Stardog. As of 1.0, those types include

- no reasoning
- OWL 2 QL
- OWL 2 EL
- OWL 2 RL

Future releases of Stardog will include RDF Schema and OWL 2 DL reasoning types.

The important implication here is that Integrity Constraint Validation is performed with three inputs: a Stardog database, a set of constraints, and a reasoning type (which may be, of course, no reasoning). This is the case because domain modelers, ontology developers, or integrity constraint authors must consider the interactions between explicit and inferred statements and how these are accounted for in integrity constraints.

Security Implications

There is a security implication in this design choice that may not be obvious. Changing the reasoning type associated with a database and integrity constraint validation has serious security implications with respect to a Stardog database and may only be performed by a user role with sufficient privileges for that action.

How to Use ICV in Stardog

This section will describe how to use Stardog ICV via the SNARL APIs. For more information on using SNARL in general, please refer to the section on **programming with Stardog in Java** (/docs/java) .

There is command-line interface support for many of the operations necessary to using ICV with a Stardog database; please see the **Administration** (/docs/admin) chapter for that documentation.

To use ICV in Stardog, one must:

1. create some constraints
2. associate those constraints with a Stardog database

Creating Constraints

`Constraints` (/docs/java/snarl/com/clarkparsia/stardog/icv/Constraint.html) can be created using the `ConstraintFactory` (/docs/java/snarl/com/clarkparsia/stardog/icv/ConstraintFactory.html) which provides methods for creating integrity constraints from OWL axioms or from SPARQL select queries. `ConstraintFactory` expects your constraints, if they are defined as OWL axioms, as an RDF triple (or graph). To aid in authoring constraints in OWL, `ExpressionFactory` (/docs/java/snarl/com/clarkparsia/openrdf/util/ExpressionFactory.html) is provided for building the **RDF equivalent** (http://www.w3.org/TR/owl2-mapping-to-rdf/) of the OWL axioms of your constraint.

You can also write your constraints in OWL in your favorite editor, such as Protege, and load them into the database from your OWL file.

We recommend defining your constraints as OWL axioms, but you are free to define them using SPARQL SELECT queries. If you choose to define a

constraint using a SPARQL select query, please keep in mind that if your query returns results, those are interpreted as the violations of the integrity constraint.

```
URI Product = ValueFactoryImpl.getInstance().createURI("urn:Product");
URI Manufacturer = ValueFactoryImpl.getInstance().createURI("urn:Manufacturer");
URI manufacturedBy = ValueFactoryImpl.getInstance().createURI("urn:manufacturedBy");

// we want to say that a product should be manufactured by a Manufacturer:
Constraint aConstraint = ConstraintFactory.constraint(subClassOf(Product,
                                                                 some(manufacturedBy, Manufacturer)));
```

This Gist view raw (https://gist.github.com/raw/1333767/ad6cee01d02aef6153b8c1ee8e4a633e0f521fb7/CreateConstraint.java) (https://gist.github.com/1333767) CreateConstraint.java (https://gist.github.com/1333767#file_create_constraint.java) brought to you by GitHub (http://github.com).

An example of creating a simple constraint using the ExpressionFactory.

Adding Constraints to Stardog

The ICVConnection (/docs/java/snarl/com/clarkparsia/stardog/icv/api/ICVConnection.html) interface provides programmatic access to the ICV support in Stardog. It provides support for adding, removing and clearing integrity constraints in your database as well as methods for checking whether or not the data is valid; and when it's not, retrieving the list of violations.

```
// We'll start out by creating a validator from our SNARL Connection
ICVConnection aValidator = aConn.as(ICVConnection.class);

// add add a constraint, which must be done in a transaction.
aValidator.addConstraint(aConstraint);
```

This Gist view raw (https://gist.github.com/raw/1333767/fbb3982a83ff07d34c19014fda4f76ce0744d846/AddConstraint.java) (https://gist.github.com/1333767) brought AddConstraint.java (https://gist.github.com/1333767#file_add_constraint.java) to you by GitHub (http://github.com).

This example shows how to add an integrity constraint to a Stardog database.

```
// We'll start out by creating a validator from our SNARL Connection
ICVConnection aValidator = aConn.as(ICVConnection.class);

// add add a constraint
aValidator.addConstraints()
        .format(RDFFormat.RDFXML)
        .file(new File("myConstraints.owl"));
```

This Gist view raw (https://gist.github.com/raw/1333767/47528396db0df5681761658d92e77fc4da651323/AddConstraint2.java) (https://gist.github.com/1333767) brought AddConstraint2.java (https://gist.github.com/1333767#file_add_constraint2.java) to you by GitHub (http://github.com).

Here we show how to add a set of constraints as defined in a local OWL ontology.

IC Validation

Checking whether or not the contents of a database are valid is easy. Once you have an ICVConnection (/docs/java/snarl/com/clarkparsia/stardog/icv/api/ICVConnection.html) you can simply call its isValid() (/docs/java/snarl/com/clarkparsia/stardog/icv/api/ICVConnection.html#isValid()) method which will return whether or not the contents of the database are valid with respect to the constraints associated with that database. Similarly, you can provide some constraints (/docs/java/snarl/com/clarkparsia/stardog/icv/Constraint.html) to the `isValid()` method to see if the data in the database is invalid for those **specific** constraints; which can be a subset of the constraints associated with the database, or they can be new constraints you are working on.

If the data is invalid for some constraints—either the explicit constraints in your database or a new set of constraints you have authored—you can get some information about what the violation was from the SNARL IC Connection. ICVConnection.getViolationBindings() (/docs/java/snarl/com/clarkparsia/stardog/icv/api/ICVConnection.html#getViolationBindings()) will return the constraints which are violated, and for each constraint, you can get the violations as the set of bindings that satisfied the constraint query. You can turn the bindings into the individuals which are in the violation using ICV.asIndividuals (/docs/java/snarl/com/clarkparsia/stardog/icv/ICV.html#asIndividuals()).

ICV and Transactions

In addition to using the ICConnection a data oracle to tell whether or not your data is valid with respect to some constraints, you can also use Stardog's

ICV support to protect your database from invalid data by using ICV as a guard within transactions.

When guard mode for ICV is enabled in Stardog, each commit is inspected to ensure that the contents of the database are valid for the set of constraints that have been associated with it. Should someone attempt to commit data which violates one or more of the constraints defined for the database, the commit will fail and the data will not be added/removed from your database.

By default, reasoning is not used when you enable guard mode, however you are free to specify any of the reasoning types supported by Stardog when enabling guard mode. If you have provided a specific reasoning type for guard mode it will be used during validation of the integrity constraints. This means you can author your constraints with the expectation of inference results satisfying a constraint.

```
StardogDBMS dbms = StardogDBMS.login("admin", "admin".toCharArray());

dbms.disk("icvWithGuard")                                          // disk db named 'icvWithGuard'
    .set(DatabaseOptions.ICVEnabled, true)                         // enable icv guard mode
    .set(DatabaseOptions.ICVReasoningType, ReasoningType.QL)    // specify the reasoning level icv guard should use
    .create(new File("data/sp2b_10k.n3"));                         // create the db, bulk loading the file(s) to sta

dbms.logout();
```

This Gist view raw (https://gist.github.com/raw/1333782/de8854ffc20c44e712ecf5375c50766be94f4b40/CreateDiskAndICV.java) (https://gist.github.com/1333782) CreateDiskAndICV.java brought to you by GitHub (http://github.com) . (https://gist.github.com/1333782#file_create_disk_and_icv.java)

This illustrates how to create a persistent disk database with ICV guard mode enabled at the QL reasoning type. Guard mode can also be enabled when the database is created on the **command line** (/docs/admin)

ICV Examples

Rather than discuss the **formal semantics** (icv-specification.html) of ICV here, we will look at some examples. The following examples use OWL 2's Manchester syntax; and they assume a simple data schema, which is available as an **OWL ontology** (company.owl) and as a **UML diagram** (ClassDiagram.png) . The examples also assume that the default namespace is `<http://example.com/company.owl#>` and that `xsd:` is bound to the standard, `<http://www.w3.org/2001/XMLSchema#>`.

We **provide reference code** (https://gist.github.com/1333767) for each of the different following examples. This is also included in the examples directory in the Stardog distribution.

Subsumption Constraints

This kind of constraint guarantees certain subclass and superclass (i.e., subsumption) relationships exist between instances.

Managers must be employees

Constraint	`Class: Manager` `  SubClassOf: Employee`	
A	`Individual: Alice` `  Types: Manager`	Invalid
B	`Individual: Alice` `  Types: Manager, Employee`	Valid

This constraint says that if an RDF individual is an instance of *Manager*, then it must also be an instance of *Employee*. In ontology A, the only instance of *Manager*, namely *Alice*, is not an instance of *Employee*; therefore, A is invalid. In B, *Alice* is an instance of both *Manager* and *Employee*; therefore, B is valid.

Domain-Range Constraints

These constraints control the types of domain and range instances for properties.

Only project leaders can be responsible for projects.

Constraint	`ObjectProperty: is_responsible_for`
	`  Domain: Project_Leader`
	`  Range: Project`	

A	`Individual: Alice`
	`  Facts: is_responsible_for MyProject`
	
	`Individual: MyProject`
	`  Types: Project`
		Invalid

B	`Individual: Alice`
	`  Types: Project_Leader`
	`  Facts: is_responsible_for MyProject`
	
	`Individual: MyProject`
		Invalid

C	`Individual: Alice`
	`  Types: Project_Leader`
	`  Facts: is_responsible_for MyProject`
	
	`Individual: MyProject`
	`  Types: Project`
		Valid

This constraint says that if an RDF instance *i* is related to an RDF instance *j* via the property *is_responsible_for*, then *i* must be an instance of *Project_Leader* and *j* must be an instance of *Project*. In ontology A, there is only one pair of individuals related via *is_responsible_for*, namely *(Alice, MyProject)*, and *MyProject* is an instance of *Project*; but *Alice* is not an instance of *Project_Leader*. Therefore, A is invalid. In B, *Alice* is an instance of *Project_Leader*, but *MyProject* is not an instance of *Project*; therefore, B is not valid. In C, *Alice* is an instance of *Project_Leader*, and *MyProject* is an instance of *Project*; therefore, C is valid.

Only employees can have an SSN.

Constraint	`DataProperty: SSN` `  Domain: Employee`	
A	`Individual: Bob` `  Facts: SSN "123-45-6789"`	Invalid

B	`Individual: Bob`
	`  Types: Employee`
	`  Facts: SSN "123-45-6789"`
		Valid

This constraint says that if an RDF instance *i* has a data assertion via the the property *SSN*, then *i* must be an instance of *Employee*. In ontology A, *Bob* is not known to be an instance of *Employee* but has *SSN*, therefore, A is invalid. In B, *Bob* is defined to be an instance of *Employee*, therefore the ontology is valid.

Each date of birth must be a date.

Constraint	`DataProperty: DOB` `  Range: xsd:date`	
A	`Individual: Bob` `  Facts: DOB "1970-01-01"`	Invalid
B	`Individual: Bob` `  Facts: DOB "1970-01-01"^^xsd:date`	Valid

This constraint says that if an RDF instance *i* is related to a literal *l* via the data property *DOB*, then *l* must have the XML Schema type *xsd:date*. In ontology A, *Bob* is related to the untyped literal *"1970-01-01"* via *DOB* so A is invalid. In B, the literal *"1970-01-01"* is properly typed so the ontology is

valid.

Participation Constraints

These constraints control whether (or not) an RDF instance participates in some specified relationship.

Each supervisor must supervise at least one employee.

Constraint	Class: Supervisor SubClassOf: supervises some Employee	
A	Individual: Alice	Valid
B	Individual: Alice Types: Supervisor	Invalid
C	Individual: Alice Types: Supervisor Facts: supervises Bob Individual: Bob	Invalid
D	Individual: Alice Types: Supervisor Facts: supervises Bob Individual: Bob Types: Employee	Valid

This constraint says that if an RDF instance *i* is of type *Supervisor*, then *i* must be related to an individual *j* via the property *supervises*, and *j* must be an instance of *Employee*. In ontology A, *Supervisor* has no instances; therefore, A is vacuously valid. In B, the only instance of *Supervisor*, namely *Alice*, is related to no individual; therefore, B is invalid. In C, *Alice* is related to *Bob* via *supervises*, but *Bob* is not an instance of *Supervisor*; therefore, C is invalid. In D, *Alice* is related to *Bob* via *supervises*, and *Bob* is an instance of *Supervisor*; therefore, D is valid.

Each project must have a valid project number.

Constraint	Class: Project SubClassOf: number some integer[> 0, < 5000]	
A	Individual: MyProject	Valid
B	Individual: MyProject Types: Project	Invalid
C	Individual: MyProject Types: Project Facts: number "23"	Invalid
D	Individual: MyProject Types: Project Facts: number "6000"^^integer	Invalid
E	Individual: MyProject Types: Project Facts: number "23"^^integer	Valid

This constraint says that if an RDF instance *i* is of type *Project*, then *i* must be related via the property *number* to an integer between *0* and *5000* (inclusive). In ontology A, the individual *MyProject* is not known to be an instance of *Project* so the constraint does not apply and the ontology is valid. In

B, *MyProject* is an instance of *Project* but is not known to have any data assertions via *number* so A is invalid. In C, *MyProject* does have a data property assertion via *number* but the literal *"23"* is untyped (not an integer) therefore the ontology is invalid. In D, *MyProject* is related to an integer via *number* but it is out of the range so the ontology is invalid. Finally, in E, *MyProject* is related to the integer *23* which is in the range of *[0,5000]* so this ontology is valid.

Cardinality Constraints

These constraints control the number of various relationships or property values.

Employees mustn't work on more than 3 projects

Constraint	`Class: Employee` `SubClassOf: works_on max 3 Project`	
A	`Individual: Bob`	Valid

B	`Individual: Bob`
	`Types: Employee`
	`Facts: works_on MyProject`
	
	`Individual: MyProject`
	`Types: Project`
		Valid

C	`Individual: Bob`
	`Types: Employee`
	`Facts: works_on MyProject, works_on MyProjectFoo, works_on MyProjectBar, works_on MyProjectBaz`
	
	`Individual: MyProject`
	`Types: Project`
	
	`Individual: MyProjectFoo`
	`Types: Project`
	
	`Individual: MyProjectBar`
	`Types: Project`
	
	`Individual: MyProjectBaz`
	`Types: Project`
		Invalid

This constraint says that if an RDF instance *i* is an *Employee*, then *i* must not be related via the property *works_on* to more than 3 named individuals of class *Project*. In ontology A, *Bob* is not known to be an instance of *Employee* so the constraint does not apply and the ontology is valid. In B, *Bob* is an instance of *Employee* but is known to work on only a single project, namely *MyProject*, so the ontology is valid. In C, *Bob* is related to 4 named individuals of class *Project* (namely, *MyProject*, *MyProjectFoo*, *MyProjectBar*, and *MyProjectBaz*) via *works_on*. Due to the weak UNA these individuals are considered distinct so the ontology is invalid.

Departments must have at least 2 employees.

Constraint	`Class: Department` `SubClassOf: inverse(works_in) min 2 Employee`	
A	`Individual: MyDepartment`	Valid

B	`Individual: MyDepartment`
	`Types: Department`
	
	`Individual: Bob`
	`Types: Employee`
	`Facts: works_in MyDepartment`
		Invalid

	`Individual: MyDepartment` `Types: Department`	

C	Individual: Bob Types: Employee Facts: works_in MyDepartment Individual: Alice Types: Employee Facts: works_in MyDepartment	Valid

This constraint says that if an RDF instance *i* is a *Department*, then there should exist at least 2 instances *j* and *k* of class *Employee* which are related to *i* via the property *works_in* (or, equivalently, *i* should be related to them via the inverse of *works_in*). In ontology A, the individual *MyDepartment* is not known to be an instance of *Department* so the constraint does not apply and the ontology is valid. In B, *MyDepartment* is an instance of *Department* but only one instance of *Employee*, namely *Bob*, is known to work in it, so the ontology is invalid. In C, *MyDepartment* is related to the individuals *Bob* and *Alice*, which are both instances of *Employee* and (due to the weak Unique Name Assumption that Stardog adopts for ICV), are distinct, so the ontology is valid.

Managers must manage exactly 1 department.

Constraint	Class: Manager SubClassOf: manages exactly 1 Department	
A	Individual: Isabella	Valid
B	Individual: Isabella Types: Manager	Invalid
C	Individual: Isabella Types: Manager Facts: manages MyDepartment	Invalid
D	Individual: Isabella Types: Manager Facts: manages MyDepartment Individual: MyDepartment Types: Department	Valid
E	Individual: Isabella Types: Manager Facts: manages MyDepartment, MyDepartment1 Individual: MyDepartment Types: Department Individual: MyDepartment1 Types: Department	Invalid

This constraint says that if an RDF instance *i* is a *Manager*, then it must be related to exactly 1 instance of *Department* via the property *manages*. In ontology A, the individual *Isabella* is not known to be an instance of *Manager* so the constraint does not apply and the ontology is valid. In B, *Isabella* is an instance of *Manager* but is not related to any instances of *Department*, so the ontology is invalid. In C, *Isabella* is related to the individual *MyDepartment* via the property *manages* but *MyDepartment* is not known to be an instance of *Department*, so the ontology is invalid. In D, *Isabella* is related to exactly one instance of *Department*, namely *MyDepartment*, so the ontology is valid. Finally, in E, *Isabella* is related to 2 distinct (again, because of weak UNA) instances of *Department*, namely *MyDepartment* and *MyDepartment1*, so the ontology is invalid.

Entities must not have more than one name.

Constraint	DataProperty: name Characteristics: Functional	
A	Individual: MyDepartment	Valid

B	`Individual: MyDepartment` `  Facts: name "Human Resources"`	Valid
C	`Individual: MyDepartment` `  Facts: name "Human Resources", name "Legal"`	Invalid

This constraint says that no RDF instance *i* can have more than 1 assertion via the data property *name*. In ontology A, the individual *MyDepartment* does not have any data property assertions so A is valid. In B, *MyDepartment* has a single assertion via *name*, so the ontology is also invalid. In C, *MyDepartment* is related to 2 literals, namely *"Human Resources"* and *"Legal"*, via *name*, so the ontology is invalid.

Property Constraints

These constraints control how instances are related to one another via properties.

The manager of a department must work in that department.

Constraint	`ObjectProperty: manages` `  SubPropertyOf: works_in`	
A	`Individual: Bob` `  Facts: manages MyDepartment`	Invalid
B	`Individual: Bob` `  Facts: manages MyDepartment, works_in MyDepartment`	Valid

This constraint says that if an RDF instance *i* is related to *j* via the property *manages*, then *i* must also be related to *j* va the property *works_in*. In ontology A, *Bob* is related to *MyDepartment* via *manages*, but not via *works_in*, so the ontology is invalid. In B, *Bob* is related to *MyDepartment* via both *manages* and *works_in*, so the ontology is valid.

Department managers must supervise all the department's employees.

Constraint	`ObjectProperty: is_supervisor_of` `  SubPropertyChain: manages o inverse(works_in)`	

A	`Individual: Jose`
	`  Facts: manages MyDepartment, is_supervisor_of Maria`
	
	`Individual: Maria`
	`  Facts: works_in MyDepartment`
	
	`Individual: Diego`
	`  Facts: works_in MyDepartment`
		Invalid

B	`Individual: Jose`
	`  Facts: manages MyDepartment, is_supervisor_of Maria, is_supervisor_of Diego`
	
	`Individual: Maria`
	`  Facts: works_in MyDepartment`
	
	`Individual: Diego`
	`  Facts: works_in MyDepartment`
		Valid

This constraint says that if an RDF instance *i* is related to *j* via the property *manages* and *k* is related to *j* via the property *works_in*, then *i* must be related to *k* via the property *is_supervisor_of*. In ontology A, *Jose* is related to *MyDepartment* via *manages*, *Diego* is related to *MyDepartment* via *works_in*, but *Jose* is not related to *Diego* via any property, so the ontology is invalid. In B, *Jose* is related to *Maria* and *Diego*, who both are related to *MyDepartment* via *works_in*, via the property *is_supervisor_of*, so the ontology is valid.

Complex Constraints

These constraints are more complex, often including multiple conditions, etc.

Employee Constraints

Each employee either works on at least one project, supervises at least one employee that works on at least one department.

Constraint	Class: Employee SubClassOf: works_on some Project or supervises some (Employee and works_on some Project) or manages some Department	
A	Individual: Esteban Types: Employee	Invalid
B	Individual: Esteban Types: Employee Facts: supervises Lucinda Individual: Lucinda Types: Employee	Invalid
C	Individual: Esteban Types: Employee Facts: supervises Lucinda Individual: Lucinda Types: Employee Facts: works_on MyProject Individual: MyProject Types: Project	Valid
D	Individual: Esteban Types: Employee Facts: manages MyDepartment Individual: MyDepartment Types: Department	Valid
E	Individual: Esteban Facts: manages MyDepartment, works_on MyProject Individual: MyDepartment Types: Department Individual: MyProject Types: Project	Valid

This constraint says that if an individual *i* is an instance of *Employee*, then at least one of three conditions must be met: First, it is related to an instance of *Project* via the property *works_on*. Second, it is related to an instance *j* via the property *supervises* and *j* is an instance of *Employee* and also related to some instance of *Project* via the property *works_on*. Third, it is related to an instance of *Department* via the property *manages*.

Ontologies A and B are invalid because none of the conditions are met. C meets the second condition: *Esteban* (who is an *Employee*) is related to *Lucinda* via the property *supervises* whereas *Lucinda* is both an *Employee* and related to *MyProject*, which is a *Project*, via the property *works_on*. D meets the third condition: *Esteban* is related to an instance of *Department*, namely *MyDepartment*, via the property *manages*. Finally, E meets the first and the third conditions because in addition to managing a department *Esteban* is also related an instance of *Project*, namely *MyProject*, via the property *works_on*

Employees and US government funding

Only employees who are American citizens can work on a project that receives funds from a US government agency.

Constraint	Class: Project and receives_funds_from some US_Government_Agency SubClassOf: inverse(works_on) only (Employee and nationality value "US")	

A	Individual: MyProject Types: Project Facts: receives_funds_from NASA Individual: NASA Types: US_Government_Agency	Valid
B	Individual: MyProject Types: Project Facts: receives_funds_from NASA Individual: NASA Types: US_Government_Agency Individual: Andy Types: Employee Facts: works_on MyProject	Invalid
C	Individual: MyProject Types: Project Facts: receives_funds_from NASA Individual: NASA Types: US_Government_Agency Individual: Andy Types: Employee Facts: works_on MyProject, nationality "US"	Invalid
D	Individual: MyProject Types: Project Facts: receives_funds_from NASA Individual: NASA Types: US_Government_Agency Individual: Andy Types: Employee Facts: works_on MyProject, nationality "US" Individual: Heidi Types: Supervisor Facts: works_on MyProject, nationality "US"	Invalid
E	Individual: MyProject Types: Project Facts: receives_funds_from NASA Individual: NASA Types: US_Government_Agency Individual: Andy Types: Employee Facts: works_on MyProject, nationality "US" Individual: Heidi Types: Supervisor Facts: works_on MyProject, nationality "US" Class: Supervisor SubClassOf: Employee	Valid

This constraint says that if an individual *i* is an instance of *Project* and is related to an instance of *US_Government_Agency* via the property *receives_funds_from*, then any individual *j* which is related to *i* via the property *works_on* must satisfy two conditions: First, it must be an instance of *Employee*. Second, it must not be related to any literal other than *"US"* via the data property *nationality*.

Ontology A is valid because there is no individual related to *MyProject* via *works_on*, so the constraint is vacuously satisfied. Ontology B is invalid since *Andy* is related to *MyProject* via *works_on*, *MyProject* is an instance of *Project* and is related to an instance of *US_Government_Agency*, that is, *NASA*, via *receives_funds_from*, but *Andy* does not have any data property assertions. C is valid because both conditions are met. D is not valid because *Heidi* violated the first condition: she is related to *MyProject* via *works_on* but is not known to be an instance of *Employee*. Finally, this is fixed in the ontology E which states that every instance of *Supervisor* is an instance of *Employee*, so *Heidi* is inferred to be an instance of *Employee* and, consequently, the

ontology is valid.

Stardog OWL 2 (/docs/)

Stardog may perform OWL reasoning during SPARQL query answering. It supports the **OWL 2 profiles** (http://www.w3.org/TR/owl2-profiles/) and **OWL 2 DL** (http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/) via an internal integration of **Pellet 3** (http://clarkparsia.com/pellet/). Stardog supports OWL 2 QL, EL and RL profile reasoning for data; it supports schema-only reasoning for OWL 2 DL.

This chapter explains the OWL 2 reasoning support in Stardog in detail, including relevant background information and reasoning terminology.

In this section we give a brief overview of Stardog's approach to query answering. Based on the specific characteristics of this approach, we derive a set of **guidelines** (#guidelines) that contribute towards efficient query answering. If you are not familiar with the terminology, you can read our section on **background and terminology** (#background).

Using Stardog's Reasoning Capabilities

Schema extraction

In order to do any reasoning, Stardog requires the Schema (or TBox) to be present in the database. Since these are all serialized as RDF, they are loaded into a Stardog database in the same way that any RDF is loaded into a database. The point here to note is, however, that they must be present to be taken into account in the reasoning process.

Note that that Stardog will *not* follow `owl:imports` statements automatically; any imported OWL ontologies that are required for reasoning must be loaded into a Stardog database in the normal way, per the preceding paragraph.

We can specify from where we want the Schema to be extracted by setting the `reasoning.schema.graphs` property to a set of named graphs URIs. If we want to take the default graph into account we can use the built-in URI `tag:stardog:api:context:default`, and if we want to use all named graphs, we can use `tag:stardog:api:context:all`. The default value for this property is to use the default graph only.

Command Line

In order to pose queries over Stardog using reasoning via the command line, one simply specifies the desired reasoning level in the **connection string** (http://stardog.com/docs/admin/):

```
$ ./stardog query -c "http://localhost:8989/myDB;reasoning=QL" -q "SELECT ?s { ?s a :C } LIMIT 10"
```

ReasoningConnection API

Currently, this API only has two methods:

- isConsistent(), which can be used to check if the current KB is consistent with respect to the current reasoning level.
- isSatisfiable(URI theURIClass), which can be used to check if the given class if satisfiable with respect to the current KB and reasoning level.

Explaining Reasoning Results

Stardog provides the functionality to explain the inferences it produces. An explanation of an implicit inference is the minimum set of statements explicitly stored in the database. Explanations are useful for debugging and understanding especially when large number of statements interact

with each other to infer new statements

Explanations can be retrieved using the CLI command 'explain inference' by providing an input file that contains the inference to be explained as in:

```
$ stardog explain inference -c snarl://localhost/myDB inference_to_explain.ttl
```

The output is displayed in a concise syntax designed to make long OWL axioms readable but it can be rendered in any one of supported RDF syntaxes if desired. Explanations are also accessible through the **extended HTTP protocol** (/docs/network/#extended-http) and SNARL API. See the examples included in the distribution for more details about retrieving explanations programmatically.

Note that, there might be more than one distinct explanation for an inference but Stardog returns a single explanation. We are planning to add support for computing multiple explanations in future versions of Stardog.

Not Getting Correct Answers?

Here's a few things that you might want to try:

- **Do you know what to expect?** The **OWL 2 primer** (http://www.w3.org/TR/owl2-primer/) is always a good place to start.
- **Is the TBox where you think it is?** Stardog might be extracting the wrong TBox. You have to tell it where it is. Check **TBox extraction** (#tbox_extraction) for details.
- **Are you using the right reasoning level?** Perhaps some of the modeling constructs (a.k.a. axioms) in your database are being ignored. You can find out which axioms are being ignored due to the reasoning level used by simply including the following line in the logging.properties file in `STARDOG_HOME`:

  ```
  com.clarkparsia.blackout.level = ALL
  ```

- **Are you using DL?** Stardog supports schema-only reasoning for OWL 2 DL, which effectively means that only TBox queries—queries that contain **TBox BGPs** (#query_types) only—should be expected to be complete with respect to query results.

Known Issues

Version **1.0** of Stardog does not support the following features:

- Recursive EQs. Query results will be sound (no wrong answers) but incomplete (some correct answers not returned) with respect to the requested reasoning type (i.e., either OWL 2 RL or EL).
- Transitivity. Axioms asserting that a property is transitive will be ignored.
- Equality reasoning. Only explicit `owl:sameAs` and `owl:differentFrom` Data assertions will be taken into account for query answering.
- Datatype reasoning and user-defined datatypes.

Guidelines for Efficient Query Answering

The query rewriting approach implemented by Stardog suggests some guidelines that might contribute to more efficient query answering.

Hierarchies and Queries

Avoid unnecessarily deep class/property hierarchies. If you do not need to model several different types of a given class or property in your Schema, then do not. The reason shallow hierarchies are desirable is that the maximal hierarchy depth in the ontology partly determines the maximal size of the EQs produced by Stardog. The larger the EQ, the more difficult is to evaluate it over the Data.

For example, suppose we add to MyDB_1 a very thorough and detailed set of subclasses of the class `:Employee`:

```
:Manager rdfs:subClassOf :Employee
:SeniorManager rdfs:subClassOf :Manager
...

:Supervisor rdfs:subClassOf :Employee
:DepartmentSupervisor rdfs:subClassOf :Supervisor
...
```

```
:Secretary rdfs:subClassOf :Employee
...
```

If we wanted to retrieve the set of all employees as before, Blackout would produce an EQ containing a query of the following form for every subclass $:C_i$ of :Employee:

```
SELECT ?employee WHERE { ?employee rdf:type :C_i }
```

At this point, it is easy to see that **the more specific the query, the better** as general queries—that is, queries that contain concepts high up in the class hierarchy defined by the Schema—as the one above, will typically yield larger EQs.

Domains and Ranges

Specify domain and range of the properties in the Schema. These types of axiom can help reduce the size of the EQs significantly due to an optimization technique implemented in Blackout called *query subsumption*. In order to grasp the intuition behind it, let us consider the following query asking for people and the employees they manage:

```
SELECT ?manager ?employee WHERE { ?manager :manages ?employee. ?employee rdf:type :Employee }
```

We know that this query would cause a large EQ given the deep hierarchy of :Employee in $MyDB_1$. However, if we added the following single range axiom:

```
:manages rdfs:range :Employee
```

then the EQ would collapse to:

```
SELECT ?manager ?employee WHERE { ?manager :manages ?employee }
```

which is considerably less difficult to evaluate.

Technical Background

Query answering with reasoning in Stardog is based on *query rewriting*. Intuitively, the idea is to expand or rewrite the original query with respect to the knowledge represented in the Schema, and then executing the resulting expanded query (EQ) over the Data only.

Blackout—Stardog's internal reasoner—implements a highly optimized version of the **query rewriting algorithm of Pérez-Urbina et al** (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.786). As can be seen in Figure 2, the rewriting process involves 5 different phases. For the sake of simplicity we do not treat them separately here; interested readers can refer to the provided paper for a detailed presentation.

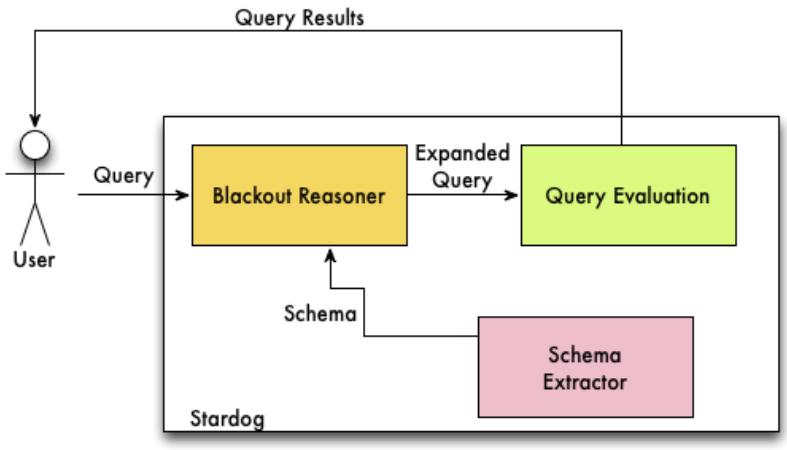


Figure 1. Query Answering

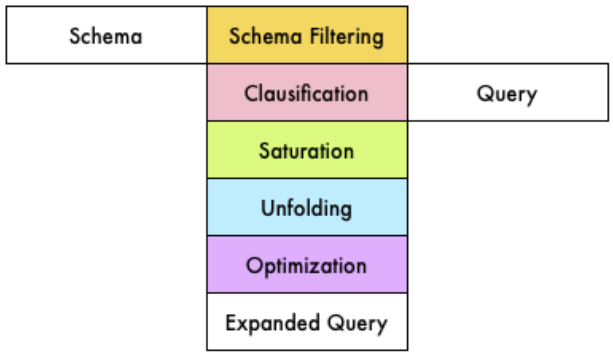


Figure 2. Query Rewriting

We illustrate the query answering process by means of an example. Consider an OWL 2 EL DB $MyDB_1$ containing the following Schema axioms:

```
:SeniorManager rdfs:subClassOf :manages some :Manager
:manages some :Employee rdfs:subClassOf :Manager
:Manager rdfs:subClassOf :Employee
```

stating that a senior manager manages at least one manager, that every person that manages an employee is a manager, and that every manager is also an employee. Moreover, let us assume $MyDB_1$ also contains the following Data assertions:

```
:Bill rdf:type :SeniorManager
:Robert rdf:type :Manager
:Ana :manages :Lucy
:Lucy rdf:type :Employee
```

Finally, let us assume that we want to retrieve the set of all employees. We do this by posing the following query over $MyDB_1$:

```
SELECT ?employee WHERE { ?employee rdf:type :Employee }
```

Given the knowledge captured in the Schema, we expect all individuals occurring in the Data to be part of the answer.

In order to answer this query, Stardog first **rewrites** it using the relevant knowledge in the Schema. In this case, it can be shown that the EQ produced by Stardog contains the following queries:

```
SELECT ?employee WHERE { ?employee rdf:type :Employee }
SELECT ?employee WHERE { ?employee rdf:type :Manager }
SELECT ?employee WHERE { ?employee rdf:type :SeniorManager }
SELECT ?employee WHERE { ?employee :manages ?x. ?x rdf:type :Employee }
```

The second and final step consists of executing the EQ over the Data by computing the union of the results of the produced queries.

The form of the EQ depends on the OWL 2 profile in which the DB is expressed. If the DB is within OWL 2 QL, then every EQ produced by Stardog *is guaranteed to be expanded into a set of queries*. If the DB is within OWL 2 RL or EL, then the EQ *might* include a recursive rule; it is important to understand, however, that this is not always the case as demonstrated by the above example.

Why Query Rewriting?

As explained previously, the query rewriting approach deals with query answering in two separate phases: first, the query is rewritten with respect to the Schema in order to get an EQ, and then the EQ is evaluated over the Data. Notably, given this separation, it can be shown that the EQ is *independent from the Data*. This independence is important on several levels:

1. The time it takes to compute the EQ depends on the size of the Schema and the original query only, and not on the size of the Data. Typically, the size of the Schema and the query are much smaller than that of the Data.
2. Given the form of the EQ, it can be evaluated over secondary storage. That is, Data does not need to be loaded into memory, which makes the query rewriting approach to query answering a suitable reasoning technique for scenarios where the Data is too big to fit in memory.
3. The same EQ can be evaluated over different instances of Data without having to be recomputed.

An alternative is a technique known as materialization. In this approach, it is the Data that gets expanded with respect to the Schema, not the query. That is, the axioms in the Schema are used as rules to generate new triples. For example we could use the axiom `:manages some :Employee rdfs:subClassOf :Manager` and the Data assertions

```
:Ana :manages :Lucy
:Lucy rdf:type :Employee
```

to derive the *new* triple:

```
:Ana rdf:type :Manager
```

which was only implicit before.

The materialization phase typically consists in applying all the axioms in the Schema to the original Data and the inferred triples until no more triples can be generated. After materialization, we can evaluate queries over the new Data disregarding the Schema altogether. In our example, it can be shown that materializing the Data with respect to the Schema would eventually produce the following triples:

```
:Bill rdf:type :Employee
:Robert rdf:type :Employee
:Ana rdf:type :Employee
:Lucy rdf:type :Employee
```

Given the fact that query answering over a materialized database does not require to take the Schema into account, it is typically faster than query answering via query rewriting. However, materialization introduces several issues:

- **Data freshness**. Materialization has to be performed every time the Data or the Schema change. This is particularly unsuitable for applications where the data changes relatively frequently.
- **Data size**. Depending on the Schema, materialization can significantly increase the size of the Data. A relatively simple class hierarchy with a single level of subclasses can duplicate the size of the Data.
- **OWL 2 profile reasoning**. Given the fact that QL, RL, and EL, are not comparable with respect to expressive power, an application that requires reasoning with more than one profile would need to maintain different corresponding materialized versions of the Data.
- **Resources**. Depending on the size of the original Data and the complexity of the Schema, materialization can be very computationally expensive.

Terminology

Databases

A *database* (DB), a.k.a. ontology, is composed of two different parts: the Schema or *Terminological Box* (TBox) and the Data or *Assertional Box* (ABox). Analogus to relational databases, the TBox can be thought of as the schema, and the ABox as the data. In other words, the TBox is a set

of *axioms*, whereas the ABox is a set of *assertions*.

As we explain in **Section OWL 2 Profiles (#profiles)**, the kinds of assertion and axiom that one might use for a particular database are determined by the fragment of OWL 2 to which one would like to adhere. In general, you should choose the OWL 2 profile that most closely fits the data modeling needs of your application.

The most common data assertions are class and property assertions. Class assertions are used to state that a particular individual is an instance of a given class. Property assertions are used to state that two particular individuals (or an individual and a literal) are related via a given property. For example, suppose we have a DB MyDB_2 that contains the following data assertions:[1] (#note-1)

```
:clark_and_parsia rdf:type :Company
:clark_and_parsia :maintains :Stardog
```

stating that `:clark_and_parsia` is a company, and that `:clark_and_parsia` maintains `:Stardog`.

The most common schema axioms are subclass axioms. Subclass axioms are used to state that every instance of a particular class is also an instance of another class. For example, suppose that MyDB_2 contains the following TBox axiom:

```
:Company rdfs:subClassOf :Organization
```

stating that companies are a type of organization.

Queries

When reasoning is enabled, Stardog executes **SPARQL queries (/docs/query/)** (simply queries from now on) depending on the type of Basic Graph Patterns they contain.

A BGP is said to be an ABox BGP if it is of one of the following forms:

- **term_1** rdf:type **uri**
- **term_1 uri term_2**
- **term_1** owl:differentFrom **term_2**
- **term_1** owl:sameAs **term_2**

A BGP is said to be a TBox BGP if it is of one of the following forms:

- **term_1** rdfs:subClassOf **term_2**
- **term_1** owl:disjointWith **term_2**
- **term_1** owl:equivalentClass **term_2**
- **term_1** rdfs:subPropertyOf **term_2**
- **term_1** owl:equivalentProperty **term_2**
- **term_1** owl:inverseOf **term_2**
- **term_1** owl:propertyDisjointWith **term_2**
- **term_1** rdfs:domain **term_2**
- **term_1** rdfs:range **term_2**

A BGP is said to be a Hybrid BGP if it is of one of the following forms:

- **term_1** rdf:type **?var**
- **term_1 ?var term_2**

where **term** (possibly with subscripts) is either an URI or variable; **uri** is a URI; and **?var** is a variable.

When executing a query, ABox BGPs are handled by Blackout, TBox BGPs are executed by Pellet, and Hybrid BGPs by a combination of both.

Reasoning

Intuitively, reasoning with a DB means to make implicit knowledge explicit. There are two main use cases for reasoning: infer implicit knowledge and discover modeling errors.

With respect to the first use case, recall that $MyDB_2$ contains the following assertion and axiom:

```
:clark_and_parsia rdf:type :Company
:Company rdfs:subClassOf :Organization
```

From this DB, we can use Stardog in order to *infer* that `:clark_and_parsia` is an organization:

```
:clark_and_parsia rdf:type :Organization
```

Using reasoning in order to infer implicit knowledge in the context of an enterprise application can lead to simpler queries. Let us suppose, for example, that $MyDB_2$ contains a complex class hierarchy including several types of organization (including company). Let us further suppose that our application requires to use Stardog in order to get the list of all considered organizations. If Stardog were used **with reasoning**, then we would need only issue the following simple query:

```
SELECT ?org WHERE { ?org rdf:type :Organization}
```

In contrast, if we were using Stardog **with no reasoning**, then we would have to issue the following considerably more complex query that considers all possible types of organization:

```
SELECT ?org WHERE { { ?org rdf:type :Organization } UNION
                    { ?org rdf:type :Company } UNION
                    ...
                  }
```

Stardog can also be used in order to discover modeling errors in a DB. The most common modeling errors are *unsatisfiable* classes and *inconsistent* DBs.

An unsatisfiable class is simply a class that cannot have any instances. Say, for example, that we added the following axioms to $MyDB_2$:

```
:Company owl:disjointWith :Organization
:LLC owl:equivalentClass :Company and :Organization
```

stating that companies cannot be organizations and vice versa, and that an LLC is a company and an organization. The disjointness axiom causes the class `:LLC` to be unsatisfiable because, for the DB to be contradiction-free, there can be no instances of `:LLC`.

Asserting (or inferring) that an unsatisfiable class has an instance, causes the DB to be *inconsistent*. In the particular case of $MyDB_2$, we know that `:clark_and_parsia` is a company AND an organization (see above); therefore, we also know that it is an instance of `:LLC`, and as `:LLC` is known to be unsatisfiable, we have that $MyDB_2$ is inconsistent.

Using reasoning in order to discover modeling errors in the context of an enterprise application is useful in order to maintain a correct contradiction-free model of the domain. In our example, we discovered that `:LLC` is unsatisfiable and $MyDB_2$ is inconsistent, which leads us to believe that there is a modeling error in our DB. In this case, it is easy to see that the problem is the disjointness axiom between `:Company` and `:Organization`.

OWL 2 Profiles

As explained in the **OWL 2 Web Ontology Language Profiles Specification** (http://www.w3.org/TR/owl2-profiles/) of the W3C, an OWL 2 profile is a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning. There are three OWL 2 profiles, each of which achieves efficiency differently.

- **OWL 2 QL** (http://www.w3.org/TR/owl2-profiles/#OWL_2_QL) is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. The expressive power of the profile is necessarily limited, however it includes most of the main features of conceptual models such as UML class diagrams and ER diagrams.
- **OWL 2 EL** (http://www.w3.org/TR/owl2-profiles/#OWL_2_EL) is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes. This profile captures the expressive power used by many such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology.
- **OWL 2 RL** (http://www.w3.org/TR/owl2-profiles/#OWL_2_RL) is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity.

Each profile restricts the kinds of axiom and assertion that can be used in a DB. Intuitively, QL is the least expressive of the profiles, followed by RL and EL; however, strictly speaking, no profile is more expressive than any other as they provide incomparable sets of constructs.

Stardog supports the three profiles of OWL 2 by making use of Blackout and Pellet. Notably, since TBox BGPs are handled completely by Pellet, Stardog supports reasoning for the whole of OWL 2 for queries containing TBox BGPs only.

Notes

1. We use the usual standard prefixes for RDF(S) and OWL. ↩ (#r1)

Stardog Java ⌂ (/docs/)

Introduction

In the **Network Programming (/docs/network/)** chapter, we looked at how to interact with Stardog over a network via HTTP and SNARL protocol.

In this chapter we describe how to program Stardog from Java using SNARL ("Stardog Native API for the RDF Language"), Sesame, and Jena. We prefer SNARL to Sesame to Jena and recommend, all other things being equal, them in that order.

If you're a Spring developer, see the **Programming with Spring (/docs/spring)** chapter.

Examples

The *best* way to learn to program Stardog with Java is to study the examples:

1. **SNARL (https://gist.github.com/1045573)**
2. **Sesame bindings (https://gist.github.com/1045568)**
3. **Jena bindings (https://gist.github.com/1045572)**
4. **SNARL and OWL 2 reasoning (https://gist.github.com/1045578)**
5. **SNARL and Connection Pooling (https://gist.github.com/1070230)**
6. **SNARL and Searching (https://gist.github.com/1085116)**

We offer some commentary on the interesting parts of these examples below.

Creating and Administering Databases

`StardogDBMS` provides simple programmatic access to all administrative functions available in Stardog.

Creating a Database

You can create a basic temporary memory database with Stardog with one line of code:

```
StardogDBMS dbms = StardogDBMS.login("admin", "admin".toCharArray());

dbms.createMemory("memDb");

// you must always log out of the dbms.
dbms.logout();
```

.github.com/raw/1333782/4fa582138ae478688695bb5c859e3804fbad6c4b/CreateTempMemDb.java)
CreateTempMemDb.java (https://gist.github.com/1333782#file_create_temp_mem_db.java)
This Gist (https://gist.github.com/1333782) brought to you by GitHub (http://github.com).

You can also use the mem (/docs/java/snarl/com/clarkparsia/stardog/StardogDBMS.html#mem()) and disk (/docs/java/snarl/com/clarkparsia/stardog/StardogDBMS.html#disk()) functions to configure and create a database in any way you prefer. These methods return DatabaseBuilder (/docs/java/snarl/com/clarkparsia/stardog/DatabaseBuilder.html) objects which you can use to configure the options of the database you'd like to create. Finally, the create (/docs/java/snarl/com/clarkparsia/stardog/DatabaseBuilder.html#create()) method takes the list of files to bulk load into the database when you create it. This returns a valid ConnectionConfiguration (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html) which can be used to create new Connections (/docs/java/snarl/com/clarkparsia/stardog/api/Connection.html) to your database.

It is important to note that, as shown in the example, you **must** take care to always log out of the server when you are done working with StardogDBMS.

```
StardogDBMS dbms = StardogDBMS.login("admin", "admin".toCharArray());

dbms.memory("waldoTest")
    .searchable(true)
    .create();

dbms.logout();
```

ithub.com/raw/1333782/b362ae8dd71c5022237eb11aa07a7f74fd78f1da/CreateMemSearchDb.java)
CreateMemSearchDb.java
(https://gist.github.com/1333782#file_create_mem_search_db.java)
This Gist (https://gist.github.com/1333782) brought to you by GitHub (http://github.com).

This illustrates how to create a temporary memory database named 'test' which supports full text search via **Waldo** (../using).

```
StardogDBMS dbms = StardogDBMS.login("admin", "admin".toCharArray());

dbms.disk("icvWithGuard")                                          // disk db nam
    .set(DatabaseOptions.ICVEnabled, true)                         // enable icv
    .set(DatabaseOptions.ICVReasoningType, ReasoningType.QL)    // specify the reasoni
    .create(new File("data/sp2b_10k.n3"));                         // create the
```

```
dbms.logout();
```

github.com/raw/1333782/de8854ffc20c44e712ecf5375c50766be94f4b40/CreateDiskAndICV.java)
CreateDiskAndICV.java
(https://gist.github.com/1333782#file_create_disk_and_icv.java)
This Gist (https://gist.github.com/1333782) brought to you by GitHub
(http://github.com) .

This illustrates how to create a persistent disk database with ICV guard mode enabled at the QL reasoning type. For more information on what the available options for `set` are and what they mean, see the refer to the **admin docs** (../admin/), specifically the chapter on administing a database.

Also note, Stardog database administration can be performed from the **command line** (../admin/).

Creating a Connection String

As you can see from all of the examples, the `ConnectionConfiguration` (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html) (in `com.clarkparsia.stardog.api` (/docs/java/snarl/com/clarkparsia/stardog/api/package-summary.html) package) class is where the initial action takes place:

```
Connection aConn = ConnectionConfiguration
        .to("noReasoningExampleTest")              // the name of the db to connect t
        .credentials("admin", "admin")             // credentials to use while connec
        .connect();
```

w raw
tps://gist.github.com/raw/1045578/04e8e32e3c9aafb99991d399adf5767ea47118c9/L4044.java)
This Gist L4044.java (https://gist.github.com/1045578#file_l4044.java)
(https://gist.github.com/1045578) brought to you by GitHub (http://github.com) .

The `to()` (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html#to()) method takes a Database Name (as a string); and then `connect()` (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html#connect()) actually connects to the database using all specified properties on the configuration.

This class and its constructor methods are used for *all* of Stardog's Java APIs: SNARL (native Stardog API), Sesame, Jena, as well as HTTP and SNARL protocol. In the latter cases, you must also call `url()` (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html#url(java.lang.String)) and pass it a valid URL to the Stardog server using the HTTP or SNARL protocols.

Without the call to `url()`, your ConnectionConfiguration will attempt to connect to a local, embedded version of the Stardog server. The Connection still operates in the standard client-server mode, the only difference is that the server is running in the *same* JVM as your application. You can use the convenience methods on StardogDBMS to start and stop the embedded server.

Note: Whether using SNARL, Sesame, or Jena, most (perhaps all) Stardog Java code will use `ConnectionConfiguration` to get a handle on a Stardog database—whether embedded or remote—and, after getting that handle, can use the API that makes the most sense for the use cases and requirements at hand.

See the `ConnectionConfiguration` (/docs/java/snarl/com/clarkparsia/stardog/api/ConnectionConfiguration.html) API docs or the **administration section** (/docs/admin/) for more information on connection strings.

Security in Stardog

We extensively discuss the security sytem in Stardog in the **security** (/docs/security) section.

When logged into a **StardogDBMS** (/docs/java/snarl/com/clarkparsia/stardog/StardogDBMS.html) you can access all security related features detailed in the security section using any of the core security interfaces for **managing users** (/docs/java/snarl/com/clarkparsia/stardog/security/UserManager.html), **roles** (/docs/java/snarl/com/clarkparsia/stardog/security/RoleManager.html) and **permissions** (/docs/java/snarl/com/clarkparsia/stardog/security/PermissionManager.html)

Shiro (http://shiro.apache.org) is used internally as the core of the security framework, but unlike previous versions, you do not need to configure Shiro directly. All management can be done via the command-line or via the security API provided by StardogDBMS

Using SNARL

In the examples (1) and (4) above, you can see how to use SNARL in Java to interact with Stardog. The SNARL API will give the best performance overall and is the native Stardog API. It uses some Sesame domain classes but is otherwise a clean-sheet API and set of implementations.

The SNARL API is fluent with the aim of making code written for Stardog easier to write and easier to maintain. Most objects are easily re-used to make basic tasks with SNARL as simple as possible. We are always interested in feedback on the API, so if you have suggestions or comments, please send them to the mailing list.

Let's take a closer look at some of the interesting parts of SNARL.

Adding Data

```
// auto commit is off by default for native connections
// lets start a transaction
aConn.begin();

// lets add some data from a file on disk
aConn.add().io()
        .format(RDFFormat.N3)
        .stream(new FileInputStream("data/sp2b_10k.n3"));
```

```
Graph aGraph = new GraphImpl();
Resource aContext = ValueFactoryImpl.getInstance().createURI("urn:test:context");

// of if you have a graph with some data, you can easily add that to the database as w
// you can also easily specify the context to add the data
aConn.add().graph(aGraph, aContext);

// we can commit this data to the db now
aConn.commit();
```

ist.github.com/raw/1045573/eb597ec47fee47b9a3a98425ad55c7197328cafa/SNARLAddData.java)
This SNARLAddData.java (https://gist.github.com/1045573#file_snarl_add_data.java)
Gist
(https://gist.github.com/1045573) brought to you by GitHub (http://github.com).

As this snippet shows, Stardog has autocommiting transactions disabled; if you need autocommit, turn it on. Otherwise, you must always surround changes to a database with a transaction begin and commit. Changes are kept locally until the transaction is committed, or you try and perform a query operation to inspect the state of the database within the transaction.

By default, RDF added will go into the default context unless specified otherwise. As shown, you can use **Adder** (/docs/java/snarl/com/clarkparsia/stardog/api/Adder.html) directly to add statements and graphs to the database, and if you want to add data from a file or input stream, you use an `io()` (/docs/java/snarl/com/clarkparsia/stardog/api/IO.html), `format()`, and `stream()` chain of method invocations.

See the **SNARL API** (/docs/java/snarl) Javadocs for all the gory details.

Getter Interface

```
// The previous query was just getting the statements which the value of aURI is the s
// which we can easily do via the getter interface
Iteration<Statement, StardogException> aIter = aConn.get().subject(aURI).iterator();

System.out.println("\nOr you can use a getter to do the same thing...");

while (aIter.hasNext()) {
        System.out.println(aIter.next());
}

// always close your iterations as well...
aIter.close();

// Getter objects are parameterizable like queries, and can be reused

Getter aGetter = aConn.get();

aGetter.predicate(RDF.TYPE);

// calling iterator() on this getter will return all statements which have RDF.TYPE as
// or we can bind the subject and get a specific type statement...
```

```
aGetter.subject(aURI);

// this will return the type triple for aURI as an Iteration
aIter = aGetter.iterator();

System.out.println("\nJust a single statement now...");

while (aIter.hasNext()) {
        System.out.println(aIter.next());
}

aIter.close();

// revert having set the predicate on the getter
aGetter.predicate(null);

// we can also get the results as a graph:
aGraph = aGetter.graph();

System.out.println("\nFinally, the same results as earlier, but as a graph...");
for (Statement aStmt : aGraph) {
        System.out.println(aStmt);
}
```

gist.github.com/raw/1045573/944a42a7859aff360144353d954f6495380e5624/SNARLGetter.java)
This Gist SNARLGetter.java (https://gist.github.com/1045573#file_snarl_getter.java)
(https://gist.github.com/1045573) brought to you by GitHub (http://github.com).

SNARL also supports some sugar for the classic statement-level (getSPO() scars, anyone?) interactions. We ask in the first line of the snippet above for an iterator over the Stardog connection, based on `aURI` in the subject position. Then a while-loop, as one might expect...

You can also parameterize `Getters` by binding different positions of the `Getter` (which acts like a kind of RDF statement filter)—and the iterating as usual.

Note the `aIter.close()` which is important for Stardog databases to avoid memory leaks.

If you need to materialize the iterator as a graph, you can do that by calling `graph()`.

The snippet doesn't show `object()` or `context()` parameters on a `Getter`, but those work, too, in the obvious way.

Parameterized SPARQL Queries

SNARL also lets us parameterize SPARQL queries.

```
URI aURI = ValueFactoryImpl.getInstance()
                .createURI("http://localhost/publications/articles/Journal1/1940/Artic

Query aQuery = aConn.query("select * where {?s ?p ?o}");
```

```
// now we can run this query...but lets set a limit on it since otherwise that'd be th
aQuery.limit(10);

TupleQueryResult aResult = aQuery.executeSelect();

System.out.println("The first ten results...");

// and do something with the results
while (aResult.hasNext()) {
        System.out.println(aResult.next());
}

// always close your result sets
aResult.close();

// query objects are easily parameterized, we can bind the "s" variable in the previou
// with a specific value
aQuery.parameter("s", aURI);

// and remove the limit
aQuery.limit(Query.NO_LIMIT);

// now we can re-run the query
aResult = aQuery.executeSelect();

System.out.println("\nNow a particular slice...");

while (aResult.hasNext()) {
        System.out.println(aResult.next());
}

aResult.close();
```

/gist.github.com/raw/1045573/2bb175079c40b72413815142026edcc3f9da45e0/SNARLQuery.java) This Gist SNARLQuery.java (https://gist.github.com/1045573#file_snarl_query.java) (https://gist.github.com/1045573) brought to you by GitHub (http://github.com).

We can make a `Query` object by passing a SPARQL query in the constructor. Simple. Obvious.

Next, let's set a limit for the results: `aQuery.limit(10)`; or if we want no limit, `aQuery.limit(Query.NO_LIMIT)`. By default, there is no limit imposed on the query object; we'll use whatever is specified in the query. But you can use limit to override any limit specified in the query, however specifying NO_LIMIT will not remove a limit specified in a query, it will only remove any limit override you've specified restoring the state to the default of using whatever is in the query.

We can execute that query with `executeSelect()` and iterate over the results. We can also rebind the "?s" variable easily: `aQuery.parameter("s", aURI)`, which will work for all instances of "?s" in any BGP in the query, and you can specify `null` to remove the binding.

Query objects are re-useable, so you can create one from your original query string and alter bindings, limit, and offset in any way you see fit and re-execute the query to get the updated results.

It's not in the code snippet, but you can also parameterize `offset`; and we'll add support for SPARQL's

DISTINCT soon, too.[1] (#note-1)

Removing Data

```
// first start a transaction
aConn.begin();

aConn.remove().io()
        .format(RDFFormat.N3)
        .file(new File("data/remove_data.nt"));

// and commit the change
aConn.commit();
```

.github.com/raw/1045573/48e9f686b20761b18ccb96a115464d397143c446/SNARLRemoveData.java)
SNARLRemoveData.java (https://gist.github.com/1045573#file_snarl_remove_data.java)
This Gist (https://gist.github.com/1045573) brought to you by GitHub (http://github.com).

Let's look at **removing** (/docs/java/snarl/com/clarkparsia/stardog/api/Remover.html) data via SNARL; in the example above, you can see that file or stream-based removal is symmetric to file or stream-based addition, i.e., calling `remove()` in an `io()` chain with a file or stream call. See the SNARL API docs for more details about finer-grained deletes, etc.

Reasoning

AStardog supports query time OWL 2 QL, EL, and RL reasoning by using a query rewriting technique.[2] (#note-2). As we discuss below in Connection Pooling, reasoning is enabled at the `Connection` layer and then any queries executed over that connection are executed with reasoning enabled; you don't need to do anything up front when you create your database if you want to use reasoning.

```
// now create one with reasoning
Connection aReasoningConn = ConnectionConfiguration
        .to("reasoningExampleTest")
        .credentials("admin", "admin")  // credentials to use while connecting
        .reasoning(ReasoningType.QL)    // now lets specify that we want a db w/ QL re
        .connect();
```

hub.com/raw/1045578/f1a831d65b26dce1ca58f88926f42af286753a02/CreateReasoningConn.java)
CreateReasoningConn.java (https://gist.github.com/1045578#file_create_reasoning_conn.java)
This Gist (https://gist.github.com/1045578) brought to you by GitHub (http://github.com).

In this code example, you can see that it's trivial to enable reasoning for a `Connection`: simply call `reasoning()` with the appropriate constant (such as `ReasoningType.QL`) passed in. In addition to OWL2 QL, EL, and RL, Stardog supports OWL2 DL *schema* queries.

For more information on how reasoning is supported in Stardog, check out the reasoning section (/docs/owl2) .

Search

We introduced a search (/docs/using) system into Stardog **0.6.5**; it can be used from the command line or remotely over the network interface. It can also be used from Java in the following way.

```
// first we'll construct a searcher
Searcher aSearch = aConn.search()
        .limit(50)                      // as before we only want the top fifty result
        .query("mac")           // our search term
        .threshold(.5);         // Since Waldo is implemented over lucene, we can also

SearchResults<Resource> aSearchResults = aSearch.search();

System.out.println("Search found " + aSearchResults.size() + " results");

// and now we can just iterate over the search results
for (SearchResult<Resource> aHit : aSearchResults) {
        System.out.println(aHit.getHit() + " with a score of: " + aHit.getScore());
}

// we can also re-use the searcher if we want to find the next set of results...

aSearch.offset(50); // we already found the first fifty, so lets grab the next set

aSearchResults = aSearch.search();

// we can now check the next page of search results!
```

st.github.com/raw/1085116/dc1f2895dadc5da9d0dd6f94a78ff28c7f06722a/SearcherUsage.java)
This Gist SearcherUsage.java (https://gist.github.com/1085116#file_searcher_usage.java)
(https://gist.github.com/1085116) brought to you by GitHub (http://github.com) .

The fluent Java API for searching in SNARL looks a lot like the other search interfaces: We create a `Searcher` instance with a fluent constructor: `limit()` sets a limit on the results; `query()` contains the search query, and `threshold` sets a minimum threshold for the results.

Then we call the `search()` method of our `Searcher` instance and iterate over the results (i.e., `SearchResults`). Last, we can use `offset()` on an existing code›Searcher to grab another page of results.

SNARL Connection Views

As of 0.7, SNARL Connections (/docs/java/snarl/com/clarkparsia/stardog/api/Connection.html#) support obtaining a specified type of Connection. This provides the ability to extend and enhance the features available to a Connection while maintaining the standard, simple Connection API. The Connection as (/docs/java/snarl/com/clarkparsia/stardog/api/Connection.html#as()) method takes as a parameter the interface,

which must be a sub-type of a Connection, that you would like to use. as will either return the Connection as the view you've specified, or it will throw an exception if the view could not be obtained for some reason.

```
SearchConnection aSearchConn = aConn.as(SearchConnection.class);
```

ub.com/raw/1085116/81aa53ea71ceb90b8ccc5adfcccce95d4a904fee/SearchConnectionView.java)
SearchConnectionView.java
(https://gist.github.com/1085116#file_search_connection_view.java)
This Gist (https://gist.github.com/1085116) brought to you by GitHub
(http://github.com) .

An example of obtaining an instance of a SearchConnection (/docs/java/snarl/com/clarkparsia/stardog/api/search/SearchConnection.html) to use Stardog's full text support (../search)
.

SNARL API Docs

Please see SNARL API (/docs/java/snarl/) docs for more information.

Using Sesame

Stardog supports the Sesame API (http://www.openrdf.org/doc/sesame/users/ch07.html) ; thus, for the most part, using Stardog and Sesame is not much different from using Sesame with other RDF databases. There are, however, at least two differences worth pointing out.

```
// Create a Sesame Repository from a Stardog ConnectionConfiguration.  The configurati
// when creating new RepositoryConnections
Repository aRepo = new StardogRepository(ConnectionConfiguration
                                                .to("testSesame")
                                                .credentials("admin", "admin"));

// init the repo
aRepo.initialize();

// now you can use it like a normal Sesame Repository
RepositoryConnection aRepoConn = aRepo.getConnection();

// always best to turn off auto commit
aRepoConn.setAutoCommit(false);
```

ew raw
ttps://gist.github.com/raw/1045568/ee598873656c5e4b400416faf45e2158c9ea74b0/init.java)
This Gist init.java (https://gist.github.com/1045568#file_init.java)
(https://gist.github.com/1045568) brought to you by GitHub (http://github.com) .

Wrap the connection with `StardogRepository`

As you can see from the code snippet, once you've created a `ConnectionConfiguration` with all the

details for connecting to a Stardog database, you can wrap that in a `StardogRepository` which is a Stardog specific implementation of the Sesame `Repository` interface. At this point, you can use the resulting Repository like any other Sesame Repository implementation. Each time you call `Repository.getConnection`, your original ConnectionConfiguration will be used to spawn a new connection to the database.

Disable Autocommit

We also suggest disabling Sesame's autocommit since it's a bit too chatty with respect to committing transactions: if you don't disable autocommit, it will commit after every RDF statement, which, for anything non-trivial, will cause you to have a bad day.

Using Jena

Stardog supports Jena via a Sesame-Jena bridge, so it's got more overhead than Sesame or SNARL. YMMV. There two points in the Jena example to re-iterate.

Init in Jena

```
// obtain a Jena model for the specified stardog database connection.  Just creating a
// database; this is roughly equivalent to ModelFactory.createDefaultModel.
Model aModel = SDJenaFactory.createModel(aConn);
```

aw
://gist.github.com/raw/1045572/305008900508c1c33109d5f22f6fc0b578678c8a/InitJena.java)
This Gist InitJena.java (https://gist.github.com/1045572#file_init_jena.java)
(https://gist.github.com/1045572) brought to you by GitHub (http://github.com).

The initialization in Jena is a bit different from either SNARL or Sesame; you can get a Jena `Model` instance by passing the `Connection` instance (returned by `ConnectionConfiguration`) to the Stardog factory, `SDJenaFactory`.

Add in Jena

```
// start a transaction before adding the data.  This is not required,
// but it is faster to group the entire add into a single transaction rather
// than rely on the auto commit of the underlying stardog connection.
aModel.begin();

// read data into the model.  note, this will add statement at a time.
// Bulk loading needs to be performed directly with the BulkUpdateHandler provided
// by the underlying graph, or by reading in files in RDF/XML format, which uses the
// bulk loader natively.  Alternatively, you can load data into the Stardog
// database using SNARL, or via the command line client.
aModel.getReader("N3").read(aModel, new FileInputStream("data/sp2b_10k.n3"), "");
```

```
// done!
aModel.commit();
```

w
//gist.github.com/raw/1045572/12cf386718dcea92867f002c3477489c2113b293/AddToJena.java)
This Gist AddToJena.java (https://gist.github.com/1045572#file_add_to_jena.java)
(https://gist.github.com/1045572) brought to you by GitHub (http://github.com) .

Jena also wants to add data to a `Model` one statement at a time, which can be less than ideal. To work around this restriction, we recommend adding data to a `Model` in a single Stardog transaction, which is initiated with `aModel.begin()`. Then to read data into the model, we recommend using RDF/XML, since that triggers the `BulkUpdateHandler` in Jena or grab a `BulkUpdateHandler` directly from the underlying Jena graph.

The other options include using the Stardog **command-line** (/docs/admin/#cli) client to bulk load a Stardog database or to use SNARL for loading and then switch to Jena for other operations, processing, query, etc.

Client-Server Stardog

As you can see, using Stardog from Java in either embedded or client-server mode is *very similar*—the only really visible difference is the use of `url()` in a `ConnectionConfiguration`: when it's present, we're in client-server model; else, we're in embedded mode.

That's a good and a bad thing: it's good because the code is symmetric and uniform. It's bad because it can make reasoning about performance difficult, i.e., it's not entirely clear in client-server mode which operations trigger (or don't trigger) a round trip with the server and, thus, which may be more expensive than in embedded mode.

In client-server mode, *everything triggers a round trip* with these exceptions:

- closing a connection outside a transaction
- any parameterizations (or other) of a query or getter instance
- any database state mutations in a transaction that don't need to be immediately visible to the transaction; that is, changes are sent to the server only when they are required, on commit, or on any query or read operation that needs to have the accurate up-to-date state of the data within the transaction.

Stardog generally tries to be as lazy as possible; but in client-server mode, since state is maintained on the client, there are fewer chances to be lazy and more interactions with the server.

Embedded Stardog

In addition to the `url()` issue, the other key difference between client-server and embedded Stardog is, of course, Java classpath woes. As of **1.0**, there is one classpath issues to watch out for:

- if you're using Jena in embedded mode, then Jena's libraries should be on the classpath *after* Stardog's, because of conflicting Lucene JARs

Please let us know if you find any other conflicts among JARs or other classpath issues.

Connection Pooling

Stardog supports connection pools for SNARL `Connection` objects for efficiency and programmer sanity. Here's how they work:

```
StardogDBMS.startEmbeddedServer();

// first create a temporary database to use (if there is one already, drop it first)
StardogDBMS dbms = StardogDBMS.login("admin", "admin".toCharArray());
if (dbms.list().contains("testConnectionPool")) {
        dbms.drop("testConnectionPool");
}
dbms.createMemory("testConnectionPool");
dbms.logout();

// First, we need a configuration object for our connections, this is all the informat
// the database that we want to connect to.
ConnectionConfiguration aConnConfig = ConnectionConfiguration
        .to("testConnectionPool")
        .credentials("admin", "admin");

// now we want to create a pool over these objects.  see the javadoc for ConnectionPoo
// more information on the options and information on the defaults.
ConnectionPoolConfig aConfig = ConnectionPoolConfig
                        .using(aConnConfig)                                 // use my conn
                        .minPool(10)                                        // the number
                        .maxPool(1000)                                      // the maximum
                        .expiration(1, TimeUnit.HOURS)                      // Connections
                        .blockAtCapacity(1, TimeUnit.MINUTES);              // I want obta

// now i can create my actual connection pool
ConnectionPool aPool = aConfig.create();

// if I want a connection object...
Connection aConn = aPool.obtain();

// now I can feel free to use the connection object as usual...

// and when I'm done with it, instead of closing the connection, I want to return it t
aPool.release(aConn);

// and when I'm done with the pool, shut it down!
aPool.shutdown();
```

aw
://gist.github.com/raw/1070230/7fda2e2aec42e8a7a697aade3dd39965083f1719/JustCode.java)
This Gist JustCode.java (https://gist.github.com/1070230#file_just_code.java)
(https://gist.github.com/1070230) brought to you by GitHub (http://github.com).

Per standard practice, we first initialize security and grab a connection, in this case to the `test` database.

Then we setup a `ConnectionPoolConfig`, using its fluent API, which establishes the parameters of the pool:

using()
: Sets which connection we want to pool

minPool()

maxPool()
: Establishes min and max pooled objects; max pooled objects includes both leased and idled objects.

expiration()
: Sets the idle life of objects; in this case, the pool reclaims objects idled for 1 hour.

blockAtCapacity()
: Sets the max time (here: in minutes) that we'll block waiting for an object when there aren't any idle ones in the pool.

Whew! Next we can create() the pool using this `ConnectionPoolConfig` thing.

Finally, we call `obtain()` on the `ConnectionPool` when we need a new one. And when we're done with it, we return it to the pool so it can be re-used, by calling `release()`. When we're done, we `shutdown()` the pool.

Since **reasoning (/docs/owl2)** in Stardog is enabled per `Connection`, you can create two pools: one with reasoning connections, one with non-reasoning connections; and then use the one you need to have reasoning *per query*. Neato mosquito: never pay for more than you need.

Deprecation & Backward Compatibility

Methods and classes in SNARL API that are marked with the `com.google.common.annotations.Beta` are subject to change and could be removed entirely prior to the 1.0 release. We are using this annotation to denote new or experimental features, the behavior or signature of which may change or, in some cases, does not yet work.

We will otherwise attempt to keep the public APIs as stable as possible, and methods will be marked with the standard `@Deprecated` annotation for a least one full revision cycle before their removal from the SNARL API.

Anything marked `@VisibleForTesting` is likely to be removed before the 1.0 release; don't write any important code that depends on functions with this annotation.

Notes

1. On that note, we'll also add support for projection variables, group by, order by, nested queries for 1.1, etc. All in good time, peeps. All in good time! ↩ (#r1)
2. In short, when reasoning is requested, a query is automatically rewritten in *n* queries, which are then executed. ↩ (#r2)

Stardog Network ⌂ (/docs/)

Network Programming & Stardog

In the **Programming with Java (/docs/java/)** chapter, we consider interacting with Stardog programatically with Java. In the **Programming with JVM-based Languages (/docs/programming/)** chapter, we consider interacting with Stardog by way of languages other than Java. In this chapter we consider interacting with Stardog over a network. In some use cases or deployment scenarios, it may be necessary to interact with or control Stardog remotely over an IP-based network. For those purposes, Stardog supports **SPARQL Protocol (http://www.w3.org/TR/rdf-sparql-protocol/)**; an Extended HTTP Protocol; and, natively, an RPC-style protocol based on **Google Protocol Buffers (http://code.google.com/apis/protocolbuffers/)**.

SPARQL Protocol: HTTP

Stardog supports the standard SPARQL Protocol HTTP bindings in a very obvious way. Since the Extended HTTP Protocol is a superset of SPARQL Protocol, the latter is documented below alongside the former.

SPARQL 1.1 Protocol for RDF will be available for the 1.0 release.

Extended HTTP Protocol

In addition to SPARQL Protocol for RDF support, Stardog supports an Extended HTTP Protocol with additional resource representations and capabilities. [1] (#note-1)

Generating URLs

You may deploy the Stardog sever in a Java Servlet Container that is exposed via an arbitrary HTTP URL; for example,

```
http://localhost:12345/stardog
```

To form the URI of a particular Stardog Database, the Database Short Name is the first URL path segment appended to the deployment URI as configured in a Java Servlet Container. For example, for the Database called `cytwombly`, the Database Network Name might be

```
http://localhost:12345/stardog/cytwombly
```

All the resources related to this database are identified by URL path segments relative to the Database Network Name; hence:

```
http://localhost:12345/stardog/cytwombly/size
```

In what follows, we use a **URI Template (http://code.google.com/p/uri-templates/)** notation to parameterize the actual request URLs, thus: `/{db}/size`.

We also abuse notation to show the permissible HTTP request types and default MIME types, thus:

```
REQ | REQ /resource/identifier → mime_type | mime_type
```

In a few cases, we use `void` as short hand for the case where there is a response code but the response body may be empty.

Accept Header

All HTTP requests that are mutative (add or remove) must include a valid `Accept` header set to the MIME type of the request body, where "valid" is a valid MIME type for N-Triples, Trig, Trix, Turtle, NQuads, or RDF/XML. In short,

RDF/XML
: `application/rdf+xml`

Turtle
: `application/x-turtle` or `text/turtle`

N-Triples
: `text/plain`

Trig
: `application/trig`

Trix
: `application/x-trix`

NQuads
: `text/x-nquads`

Further, in other cases where `Accept` header should be set—chiefly for content negotiation—the other MIME types that Stardog recognizes include:

SPARQL XML Results Format
: `application/sparql-results+xml`

SPARQL JSON Results Format
: `application/sparql-results+json`

SPARQL Boolean Results
: `text/boolean`

SPARQL Binary Results
: `application/x-binary-rdf-results-table`

Response Codes

Stardog uses HTTP response codes in the following way:

`200`
: Indicates the operation has succeeded.

`400`
: Indicates parse errors or that the transaction identifier specified for an operation is invalid or does not correspond to a known transaction.

`403`
: Indicates that the user attempting to perform the operation does not exist, their username or password is invalid, or they do not have the proper credentials to perform the action.

`500`
: Indicates a failure in some internal operation.

In cases of error, the message body of the result will include any error information provided by the server to indicate the cause of the error.

Exposed Resources

To interact with Stardog over HTTP, use the following resource representations, HTTP response codes, and resource identifiers.

A Stardog Database

`GET /{db} → void`

Returns a representation of the database. As of **1.0**, this is merely a placeholder; in a later release, this resource will include basic information about the database, including a link to the SPARQL 1.1 service description, and other database metadata.

Database Size

`GET /{db}/size → text/plain`

Returns the number of RDF triples in the database.

Query Evaluation

`GET | POST /{db}/query → application/sparql-results+xml | text/boolean | application/rdf+xml`

The SPARQL endpoint for the database. Note that ASK queries currently only support an Accept header of text/boolean, you cannot get SPARQL-XML result formatted results for ASK queries.

Query Plan

`GET | POST /{db}/explain → text/plain`

Returns the explanation for the execution of a query, i.e., a query plan. All the same arguments as for Query Evaluation are legal here; but the only MIME type for the Query Plan resource is `text/plain`.

Transaction Begin

`POST /{db}/transaction/begin → text/plain`

Returns a transaction identifier resource as `text/plain`; this is likely to be deprecated in a future release in favor of a hypertext format in place of `text/plain`.

Transaction Security Considerations

Note: Stardog's implementation of transactions with HTTP is vulnerable to man-in-the-middle attacks, which could be used to violate Stardog's isolation guarantee (among other nasty side effects).

Stardog's transaction identifiers are 64-bit GUIDs and, thus, pretty hard to *guess*; but if you can grab a response in-flight, you can steal the transaction identifier if basic access auth or RFC 2069 digest auth is in use.

You've been warned.

In a future release, Stardog will switch to only use **RFC 2617 HTTP Digest Authentication (http://tools.ietf.org/html/rfc2617)**, which is less vulnerable to various attacks, and will never ask a client to use a different authentication type, which should lessen the likelihood of MitM attacks for properly restricted Stardog clients.[2] (#note-2)

Transaction Commit

`POST /{db}/transaction/commit/{txId} → void | text/plain`

Returns a representation of the committed transaction; `200` means the commit was successful. Otherwise a `500` error indicates the commit failed and the text returned in the result is the failure message.

As you might expect, failed commits exit cleanly, rolling back any changes that were made to the database.

Transaction Rollback

`POST /{db}/transaction/rollback/{txId} → void | text/plain`

Returns a representation of the transaction after it's been rolled back. 202 means the rollback was successful, otherwise 500 indicates the rollback failed and the text returned in the result is the failure message.

Query in Transaction

```
GET | POST /{db}/{txId}/query → application/sparql-results+xml | text/boolean | application/rdf+xml
```

Returns a representation of a query executed within the txId transaction. Queries within transactions will be slower as extra processing is required to make the changes visible to the query.

Add in Transaction

```
POST /{db}/{txId}/add → void | text/plain
```

Returns a representation of data added to the database within of the specified transaction. Accepts an optional parameter, graph-uri, which specifies the named graph the data should be added to. If a named graph is not specified, the data is added to the default (i.e., unnamed) context. The response codes are 200 for success and 500 for failure.

Remove in Transaction

```
POST /{db}/{txId}/remove → void | text/plain
```

Returns a representation of data removed from the database within the specified transaction. Also accepts graph-uri with the analogous meaning as above (Add in Transaction); response codes are the same as with Add in Transaction.

Clear Database

```
POST /{db}/{txId}/clear → void | text/plain
```

Removes all data from the database within the context of the transaction. 200 indicates success; 500 indicates an error. Also takes an optional parameter, graph-uri, which removes data from a named graph. To clear only the default graph, pass DEFAULT[3] (#note-3) as the value of graph-uri.

Explanation of inferences

```
POST /{db}/reasoning/explain        → text/turle | application/x-turtle
POST /{db}/reasoning/{txId}/explain → text/turle | application/x-turtle
```

Returns the explanation of the axiom which is in the body of the POST request. The request takes the axioms in any supported RDF format and returns the explanation for why that axiom was inferred in Turtle format.

Consistency

```
GET | POST /{db}/reasoning/consistency        → text/boolean
```

Returns whether or not the database is consistent w.r.t to the TBox.

Listing Integrity Constraints

```
GET   /{db}/icv       → RDF
```

Returns the integrity constraints for the specified database serialized in any supported RDF format.

Adding Integrity Constraints

```
POST /{db}/icv/add
```

Accepts a set of valid Integrity constraints serialized in any RDF format supported by Stardog and adds them to the database in an atomic action. 200 return code indicates the constraints were added successfully, 500 indicates that the constraints were not valid or unable to be added.

Removing Integrity Constraints

```
POST /{db}/icv/remove
```

Accepts a set of valid Integrity constraints serialized in any RDF format supported by Stardog and removes them from the database in a single atomic action. 200 indicates the constraints were successfully remove; 500 indicates an error.

Clearing Integrity Constraints

```
POST /{db}/icv/clear
```

Drops **ALL** integrity constraints for a database. 200 indicates all constraints were successfully dropped; 500 indicates an error.

SNARL: The Native Stardog Remote API (1.0)

The Stardog Native API for the RDF Language, or SNARL, protocol is an RPC protocol based on exchanging **Google Protocol Buffers** (http://code.google.com/apis/protocolbuffers/) formatted messages, which we call BARC (BigPacket Access for Remote Communications). The SNARL protocol supports the same set of client operations as the HTTP interface while also supporting the administrative functions provided by the Stardog database server.

Big Packets

Client and server are implemented via **Netty** (http://http://netty.io/) with the payloads being exchanged between them encoded via Google's Protocol Buffers in an RPC-esque manner.

Netty's built in Protobuf support allows only a single kind of `Message` to be exchanged over a `Channel`; you have to provide the archetype of the `Message` to be exchanged when you start things up. Since we want to exchange more than one type of message between the client and the server, we're using Protobuf extensions to create the Big Packets API.

BigPacket

The single message type exchanged between client and server is `BigPacket`. There is a handler prior to the Netty-Protobuf layer which enforces frame sizes. 3 bytes are reserved to encode the ensuing frame size (in bytes) allowing for frames up to 16.7M bytes. This limits the size of BARC messages to 16.7M.

The Protobuf docs don't recommend having `Messages` larger than 1M. No explicit reasoning is provided: they suggest smaller messages or using a different system. Unfortunately a 2 byte size header only gives us frames of 64k, which is rather small and well under the suggested threshold. Some care is taken to limit message size on the server by providing a max number of things we'll send in a single message; while we've got nearly 17M of space to use in the message payload, we're typically using only a fraction of that.

Each `BigPacket` has a UID which uniquely identifies the packet and can be used by a client to associated responses with the original request. BigPacket's also contain a type, as an int, which is application specific. The type describes what is in the payload, this is typically provided to a message decoder which will extract the relevant information from the packet's payload before passing the request to the appropriate handler.

The complete listing of types in the base protocol can be found **here** (/docs/java/snarl/com/clarkparsia/stardog/snarl/shared/PacketType.html) while the packet types for the admin protocol are listed **here** (/docs/java/snarl/com/clarkparsia/stardog/snarl/admin/shared/Admin/PacketType.html).

BigPacket also contains error information which is specified on response packets to indicate to the client that there was a failure which

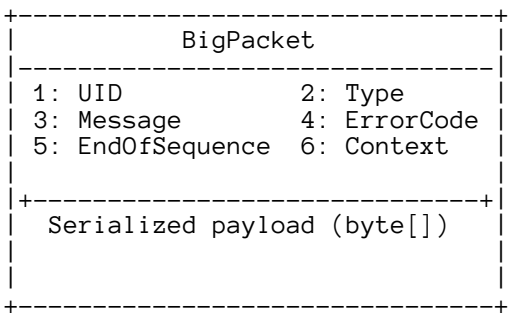
processing the original request. There is an error string containing any relevant error information along with an error code which can be used to associate an error with an appropriate application specific exception type.

While BigPacket's can be quite large, up to 16.7M, there are cases where the data for the request/response is larger than the maximum, or it simply faster to send it in smaller chunks. Thus, there is a boolean field on the BigPacket which indicates whether or not it's the last packet in a sequence of packets. When false, there will be more packets with the same UID arriving; packets arrive in the order they are sent, when there are more in the sequence, you can put them in a queue for processing until you see the terminating packet.

BigPacket also contains information about the connection it belongs to. This includes authentication credentials and database connection properties such that the connection can correctly be established on the remote end.

Lastly the BigPacket contains the payload which is simply a sequence of bytes.

BigPacket Diagram



Streaming BigPackets

We stream requests or results which are too big to fit into a single Protobuf message, both due to the size restrictions suggested by Google and due to memory constraints on the client or server.

A request or response can be sent as a stream of BigPackets each with the same ID. The entity receiving the packets is responsible for collating them into a single coherent structure that it's able to work with. The end of the stream of BigPacket is specified by the header flag `endOfSequence` being set to true.

SNARL Protocol

The best source is the source:

- **core.proto** (core.proto)
- **admin.proto** (admin.proto)
- **snarl.proto** (snarl.proto)

Errors

Errors are provided to the client in the header of the packet. The payload is always empty when the packet type is an error; the error code and message fields are populated with information describing the error condition and the error message allowing the appropriate exception to be re-created and thrown on the client.

Authentication

On connect, the client provides its credentials with an authentication request (`AuthRequest`). The server will verify the credentials specified in the request; and, if successful, it will send back a `ConnectionContext` as the payload of a `CONNECTION_CONTEXT`. The resulting `ConnectionContext` must be provided with all subsequent operations against the server or the action will not succeed because the request cannot be authenticated.

Query

Query requests are initialized with a `QueryRequest` message using a type of `QUERY_REQUEST`. There is a single endpoint on the server that is called for all query types. It dispatches to a specific query handler based on the query in the request and then returns the appropriate result(s).

Boolean queries are a single response from the server, `AskResult` with type `ASK_RESULT`.

The initial response to a select query is a `SelectResult`, with type `SELECT_RESULT`, which lists the bindings (i.e., the projection) of the executed query. The rest of the results in the stream will be a series of `Tuples` messages using the `TUPLES` type. Tuples messages contain lists of result bindings, roughly analagous to the Sesame BindingSet. These bindings are ordered within the Tuples messages, and Tuples messages arrive in order, so these can be added as they come in to a List to reproduce the correct ordering and number of results to the original query. The final Tuples message in the sequence will have an endOfSequence marker as `true` to indicate there are no more results pending.

The arrays of bindings in Tuples messages are encoded with `LiteBindingSet` messages. These simply have an array of RDF value objects and are the contents of a single result binding. The corresponding variable for each binding value is not included in the binding set for the purposes of compression. However, the bindings are always in the same order as the variables listed in the projection as provided in the initial SelectResult response from the server. So looping over those variables and grabbing the value at the corresponding index is sufficient for re-creating the full binding.

Similar to select queries, the results of a graph query return as a series of `Graph` messages. Each Graph message contains a list of `Statement` messages. The Graph message can be collected, the Statements pulled from those messages and returned to the user as part of the result to the graph query.

Compressed messages

The compressed message model is based on a lookup; `Dictionary`, being intermixed with the compressed payloads, either `CompressedGraph` or `CompressedTuples`. This lookup is a mapping from integers to strings, where the strings are the namespace URIs of the URIs returned in the subsequent RDF payload. So the flow of packets in the stream alternates between dictionary and compressed messages. The dictionary is always scoped to the subsequent compressed message, so it does not have to be kept cumulatively throughout the stream. This is done to minimize memory overhead of the receiving end of the compressed data. In the case of sending CompressedTuples messages, they are still preceded by a SelectResult message with the projection, and the `CompressedBindingSet` messages still adhere to the same ordering guidelines as their uncompressed counterparts.

Currently, compressed tuples & graphs are not used by the server and the design is not yet finalized.

Size

Requesting the size of the database currently connected to is very straightforward. A message with type `SIZE` and an empty payload is sent to the server. If successful, the server will respond with a `LongMessage` payload which is a simple protobuf wrapper around a single long value, which in this case, is the size of the database.

Transactions

Before any data upload, either an add or remove, can take place, a transaction must be initiated for a connection. The client should send a message of the type `BEGIN_TX`. The server response will contain a new `ConnectionContext` the client must use which contains the UID of the newly opened transaction. Any subsequent data operations will take place inside this transaction. Similarly, queries will query the state of the database with respect to the contexts of the transaction.

Once changes have been made to the database, the transaction can be rolled back or committed to persist the changes. A message with the type `ROLLBACK_TX` or `COMMIT_TX` should be sent to the server. Like with beginning a transaction, the server will respond with a new `ConnectionContext` for the client to use for all subsequent interactions with the server. The new context will reflect the fact that the transaction is no longer active.

Changing the contents of a database

Uploads use the same packet streaming model as graph query results returned from the server, except for the fact that the streams originate on the client rater than the server. These streams of RDF are combined with the type of the original packet, such as `ADD`, or `REMOVE`, to process the upload. As previously mentioned, a transaction **MUST** be active in order for changes to the database to succeed.

Removing data

If you want to remove data from the database you are connected to, you need to send a stream of messages, again formatted using the same type of `Graph/Statement` stream used to provide graph query results from the server. Each message should be typed with `REMOVE` and as usual, the final message in the stream should denote endOfSequence=true. Once this final message is received the server will begin processing the message stream and applying the sequence of removes to the connection. When this is completed, the server will send a boolean acknowledgement back to the client formatted as a `BoolMessage` indicating whether or not the operation was successful.

Clearing a context requires a single message of type `REMOVE_CONTEXT`. The payload should be a `Resource` corresponding to the URI of the context to be removed. The response from the server will be a single packet with a `BoolMessage` payload which contains the result of the invocation on the server, true if dropping the context was successful, false otherwise.

To clear the database, a single message of type `REMOVE_ALL` should be sent to the server, the payload should be empty. This will clear the contents of the database you are connected to. The response from the server will be a single packet with a `BoolMessage` payload which contains the result of the invocation on the server, true if the clear was successful, false otherwise.

Adding data

Adding data works **exactly** the same as removing RDF from the database except for the fact that the messages in the stream of RDF to add should be typed as `ADD` messages as opposed to `REMOVE`. The payload will be applied when the server sees the end of sequence flag, and again, a boolean acknowledgement will be sent back from the server indicating the results of the operation.

Search

Coming Soon.

Reasoning

Coming Soon.

Extending the Protocol

Coming Soon.

Notes

1. Note, as of **1.0**, none of the administrative functions are available via HTTP. They're available via **command-line** (/docs/admin/) and **programmatically** (/docs/java/); they will also be available via the SNARL protocol. ↩ (#r1)

2. In other words, a Stardog client that treats any request by a proxy server or origin server (i.e., Stardog) to use basic access auth or RFC 2069 digest auth as a MitM attack. See **RFC 2617** (http://tools.ietf.org/html/rfc2617), Section 4.8 Man in the Middle for more information. ↩ (#r2)

3. We will deprecate this identifier in favor of a proper URI or URN for the default graph in Stardog. Stay tuned. ↩ (#r3)

Stardog Spring ⌂ (/docs/)

Introduction

Spring (http://www.springsource.org/) is a platform to build and run enterprise applications in Java. Stardog's Spring support makes life easier for enterprise developers who need to work with Semantic Web technology—including RDF, SPARQL, and OWL—by way of Stardog.

The Spring for Stardog **source code** (http://github.com/clarkparsia/stardog-spring) is available on Github. A more featureful version of will be available in **Stardog Enterprise Edition** (/enterprise).

Building Spring for Stardog

To build Spring for Stardog, you need a relase of Stardog; we use **Gradle** (http://www.gradle.org/) to build Stardog for Spring. Then,

- edit `build.gradle` to point `stardogLocation` at a Stardog release directory;
- run `gradlew`, which will download and bootstrap a gradle build environment;
- then run `gradlew build`, which eventually results in a `stardog-spring.jar` in `build/libs`; finally,
- `gradlew install` does a build, generates a POM, and installs the POM in local Maven repo; alternately,
- `mvn install` will work, too:

```
mvn install:install-file
      -DgroupId=com.clarkparsia.stardog
      -DartifactId=stardog-spring
      -Dversion=0.0.2
      -Dfile=stardog-spring.jar
      -Dpackaging=jar
      -DpomFile=pom.xml
```

Overview

Spring for Stardog **0.0.3** provides a set of capabilities for rapidly building Stardog-backed applications with the Spring Framework. As with many other parts of Spring, Stardog's Spring integration uses the template design pattern for abstracting standard boilerplate away from application developers.

At the lowest level, Spring for Stardog includes

1. `DataSouce` and `DataSourceFactoryBean` for managing Stardog connections
2. `SnarlTemplate` for transaction- and connection-pool safe Stardog programming
3. `DataImporter` for easy bootstrapping of input data into Stardog

Future releases of Spring for Stardog will address other common enterprise capabilities: Spring Batch, Spring Data, etc.

Use

There are three Beans to add to a Spring application context:

- `DataSourceFactoryBean`: `com.clarkparsia.stardog.ext.spring.DataSourceFactoryBean`
- `SnarlTemplate`: `com.clarkparsia.stardog.ext.spring.SnarlTemplate`
- `DataImporter`: `com.clarkparsia.stardog.ext.spring.DataImporter`

`DataSourceFactoryBean` is a Spring `FactoryBean` that configures and produces a `DataSource`. All of the Stardog `ConnectionConfiguration` and `ConnectionPoolConfig` methods are also property names of the `DataSourceFactoryBean`—for example, "to", "url", "createIfNotPresent".

`DataSource` is a Spring for Stardog class, similar to `javax.sql.DataSource`, that can be used to retrieve a `Connection` from the `ConnectionPool`. This additional abstraction serves as place to add Spring-specific capabilities (e.g. spring-tx support in the future) without directly requiring Spring in Stardog.

`SnarlTemplate` provides a template abstraction over much of Stardog's native API, **SNARL (/docs/java)**, and follows the same approach of other Spring template, i.e., `JdbcTemplate`, `JmsTemplate`, and so on.

The key methods on `SnarlTemplate` include the following:

`query(String sparqlQuery, Map args, RowMapper)`

`query()` executes the SELECT query with provided argument list, and invokes the mapper for result rows.

`doWithAdder(AdderCallback)`

`doWithAdder()` is a transaction- and connection-pool safe adder call.

`doWithGetter(String subject, String predicate, GetterCallback)`

`doWithGetter()` is the connection pool boilerplate method for the `Getter` interface, including the programmatic filters.

`doWithRemover(RemoverCallback)`

`doWithRemover()` As above, the remover method that is transaction and pool safe.

`execute(ConnectionCallback)`

`execute()` lets you work with a connection directly; again, transaction and pool safe.

`construct(String constructSparql, Map args, GraphMapper)`

`construct()` executes a SPARQL CONSTRUCT query with provided argument list, and invokes the `GraphMapper` for the result set.

`DataImporter` is a new class that automates the loading of RDF files into Stardog at initialization time.

It uses the Spring Resource API, so files can be loaded anywhere that is resolvable by the Resource API: classpath, file, url, etc. It has a single load method for further run-time loading and can load a list of files at initialization time. The list assumes a uniform set of file formats, so if there are many different types of files to load with different RDF formats, there would be different `DataImporter` beans configured in Spring.

Here's a sample `applicationContext`:

```
<bean name="dataSource" class="com.clarkparsia.stardog.ext.spring.DataSourceFactoryBea
      <property name="to" value="testdb"/>
      <property name="createIfNotPresent" value="true"/>
</bean>

<bean name="template" class="com.clarkparsia.stardog.ext.spring.SnarlTemplate">
      <property name="dataSource" ref="dataSource"/>
</bean>

<bean name="importer" class="com.clarkparsia.stardog.ext.spring.DataImporter">
      <property name="snarlTemplate" ref="template"/>
      <property name="format" value="N3"/>
      <property name="inputFiles">
        <list>
        <value>classpath:sp2b_10k.n3</value>
        </list>
      </property>
</bean>
```


Examples

query() with SELECT queries

```
String sparql = "SELECT ?a ?b WHERE { ?a  <urn:test:b> ?b } LIMIT 5";

List<Map<String,String>> results = snarlTemplate.query(sparql, new RowMapper<Map<Strin
        @Override
        public Map<String,String> mapRow(BindingSet bindingSet) {
                Map<String,String> map = new HashMap<String,String>();
                map.put("a", bindingSet.getValue("a").stringValue());
                map.put("b", bindingSet.getValue("b").stringValue());
                return map;
        }

});
```

view raw (https://gist.github.com/raw/1115894/aa240b74fc2c7a37cc5b7fa369099f9d29ee5ef5/spring-select.java)
This Gist spring-select.java (https://gist.github.com/1115894#file_spring_select.java) (https://gist.github.com/1115894) brought to you by GitHub (http://github.com).

doWithGetter

```
List<String> results = snarlTemplate.doWithGetter(null, "urn:test:n", new GetterCallba
        @Override
        public String processStatement(Statement statement) {
                return statement.getObject().stringValue();
        }
});
```

ist.github.com/raw/1115915/0dfe80efc54beab43b40e4fffee124a0a421ea46/dowithgetter.java)
This Gist dowithgetter.java (https://gist.github.com/1115915#file_dowithgetter.java) (https://gist.github.com/1115915) brought to you by GitHub (http://github.com).

doWithAdder

```
snarlTemplate.doWithAdder(new AdderCallback<Boolean>() {
        @Override
        public Boolean add(Adder adder) throws StardogException {
                String uriA = "urn:test:j";
                String uriB = "urn:test:k";
                String litA = "hello world";
                String litB = "goodbye";

                adder.statement(new URIImpl(uriA), new URIImpl(uriB), new LiteralImpl(
                adder.statement(new URIImpl(uriA), new URIImpl(uriB), new LiteralImpl(
                return true;
        }
});
```

gist.github.com/raw/1115915/85fd633b983a5daaf639bd9db2727daf050fefa3/doWithAdder.java)
This Gist doWithAdder.java (https://gist.github.com/1115915#file_do_with_adder.java)
(https://gist.github.com/1115915) brought to you by GitHub (http://github.com).

doWithRemover

```
snarlTemplate.doWithRemover(new RemoverCallback<Boolean>() {
        @Override
        public Boolean remove(Remover remover) throws StardogException {
                remover.statements(new URIImpl("urn:test:m"), new URIImpl("urn:test:n"
                return true;
        }
});
```

st.github.com/raw/1115915/877499d75ad0d2e8430beb07c2cc30761f91ab89/doWithRemover.java)
This Gist doWithRemover.java (https://gist.github.com/1115915#file_do_with_remover.java)
(https://gist.github.com/1115915) brought to you by GitHub (http://github.com).

construct()

```
String sparql = "CONSTRUCT { ?a <urn:test:new> ?b } WHERE { ?a <urn:test:p> ?b }";
List<Map<String,String>>  results = snarlTemplate.construct(sparql, new GraphMapper<Ma
        @Override
        public Map<String, String> mapRow(Statement next) {
                Map<String,String> map = new HashMap<String,String>();
                map.put(next.getSubject().stringValue(), next.getObject().stringValue(
                return map;
        }
});
```

w
//gist.github.com/raw/1115915/5257c61451a5988cd399c2407763d3ee0ebf79fa/construct.java)
This Gist construct.java (https://gist.github.com/1115915#file_construct.java)
(https://gist.github.com/1115915) brought to you by GitHub (http://github.com).

Stardog Compatibility (/docs/)

Stardog 1.0 and the Future of Queries, Data, and Programs

Introduction

The Stardog 1.0 release ("Stardog" for short) is a major milestone in the development of the system. Stardog is a stable platform for the growth of projects and programs written for Stardog.

Stardog provides (and defines) several user-visible things:

1. SNARL API
2. BigPacket Message Format
3. Stardog Extended HTTP Protocol
4. a command-line interface

It is intended that programs—as well as SPARQL queries—written to Stardog APIs, protocols, and interfaces will continue to run correctly, unchanged, over the lifetime of Stardog.[1] (#note-1) At some indefinite point, Stardog 2.0 may be released; but, until that time, Stardog programs that work today should continue to work even as future releases of Stardog (1.1, 1.2, 1.3, etc.) occur.

APIs, protocols, and interfaces may grow, acquiring new parts and features, but not in a way that breaks existing Stardog programs.

Expectations

Although we expect that nearly all Stardog programs will maintain this compatibility over time, it is impossible to guarantee that no future change will break any program. This document sets expectations for the compatibility of Stardog programs in the future. The main, foreseeable reasons for which this compatibility may be broken in the future include:

1. **Security**: We reserve the right to break compatibility if doing so is required to address a security problem in Stardog.
2. **Unspecified behavior**: Programs that depend on unspecified[2] (#note-2) behaviors may not work in the future if those behaviors are modified.
3. **3rd Party Specification Errors**: It may become necessary to break compatibility of Stardog programs in order to address problems in some 3rd party specification.
4. **Bugs**: It will not always be possible to fix bugs found in Stardog—or in its 3rd party dependencies—while also preserving compatibility. With that proviso, we will endeavor to only break compatibility when repairing critical bugs.

It is always possible that the performance of a Stardog program may be (adversely) affected by changes in the implementation of Stardog. No guarantee can be made about the performance of a given program between releases, except to say that our expectation is that performance will generally trend in the appropriate direction.

Data Migration & Safety

Finally, we expect that data safety will always be given greater weight than any other consideration. But since Stardog stores user data differently from the form in which data is input to Stardog, we may from time to time change the way it is stored such that explicit data migration will be necessary.

Stardog provides for two data migration strategies:

1. Command-line migration tool(s)
2. Dump and reload

We expect that explicit migrations may be required from time to time between different releases of Stardog 1.0 (that is, 1.1, 1.2, etc.). We will endeavor to minimize the need for such migrations. We will only require the "dump and reload" strategy between *major* releases of Stardog (that is, from 1.x to 2.x, etc.), unless that strategy of migration is required to repair a security or other data safety bug in Stardog 1.x.

Notes

1. That is, over all releases identified by version `1.x`. ↩ (#r1)

2. The relevant specs include the Stardog-specific specifications documented on this site, but also W3C (and other) specifications of various languages, including SPARQL, RDF, RDFS, OWL 2, HTTP, Google Protocol Buffers, as well as others. ↩ (#r2)

```
1.0 Changes
-----------
* MODIFIED:     New login functions in StardogDBMS to clarify which server (embedded or remote)
we are connecting to
* FIXED:        Handling DESCRIBE queries where triples about the resource to describe occur in
named graphs
* FIXED:        Handling DESCRIBE queries with multiple resources to describe
* ADDED:  Προσθέσαμε πασχαλινά αυγά.

1.0 RC1 Changes
---------------
* ADDED:        Batch files for executing the Stardog command line in Windows
* ADDED:        directSubClassOf & directSubPropertyOf (same sparqldl namespace that is used by
Pellet 'http://pellet.owldl.com/ns/sdle#') which can be used to get
                        the explicit sub class/properties in a query that is using reasoning.
* ADDED:        New command line function to check whether or not the database is consistent.
* ADDED:        New command line function 'explain inference' to explain reasoning results
(existing 'explain' command has been renamed to 'explain plan' to avoid confusion).
* FIXED:        Resolved issue where an unbound type operator, "?s a ?o" produced incorrect
inferences for some ontologies where there is interaction b/w domain/range & hasValue axioms.
* FIXED:        Bug in the checking of user & role resource names when they're using the All
permission, eg "user:*"
* FIXED:        Consistency checks will not fail (say the db is inconsistent) when the
consistency check requires reasoning services not provided by Stardog, such as datatype
reasoning.
* FIXED:        Deadlock in bulk loading which can occur when the bulk load exceeds 5B triples.
* FIXED:        Consistency checks for functional data properties which require literal value
validations
* FIXED:        Bug in axiom filtering for the EL profile that let non-EL axioms into an EL
knowledge base
* FIXED:        Bug in reasoning in the type operator where nominal are used within an
equivalence or subclass axiom
* FIXED:        Race condition in index commits in highly multi-threaded environments
* MODIFIED:     Clarified error message when using search but the database does not have search
enabled
* MODIFIED:     Consistency checking performance improved
* MODIFIED:     Bulk loading performance improved at larger scales
* MODIFIED: Small reduction in memory overhead of query response streams

0.9.6 Changes
-------------
* FIXED:        Checking consistency of a database over HTTP.
* FIXED:        Resolved regex issue used to validate input to security where some valid inputs
could be rejected.
* FIXED:        Multi-line queries passed to CLI explain are now correctly parsed.
* FIXED:        Inconsistent usage of base URI used for parsing when one is not specified by the
user.
* FIXED         Fixed an issue with automated TBox extraction where some parts of the ABox were
incorrectly put into the TBox resulting in slower than expected
                        performance for queries involving reasoning.
* FIXED:        Bug in reasoning query evaluation where the property variable is unbound which
resulted in incorrect inferences to be returned.
* FIXED:        Password decoding in SNARL was incorrectly padding when converting to/from UTF-8
resulting in passwords sent over HTTP failing to match
                        the password in the user database which only occurred with passwords over
a certain length.
* FIXED:        Queries with reasoning where the predicate in a BGP is unbound does not correctly
return explicit statements when the schema is not present.
* FIXED:        Context variables, such as "graph ?g { ... }" are correctly used in joins and
used for join ordering.
* FIXED:        Issue with decimal inlining where negative scale values were handled incorrectly
resulting in decimal values being mangled on load.
* MODIFIED: Avoiding reclassification of TBox after commit when the TBox contents did not change.
Significant improvement in commit performance for databases with expressive TBoxes.
* MODIFIED:     Handling of add/remove operations on the server resulting in decreased memory
overhead and up to a 50% increase in the time it takes to process the change.
* MODIFIED: DL queries that involve ABox operators now cause errors at query time.
* MODIFIED: Inferences involving Top/Bottom Object/Data property are ignored.
* MODIFIED: Offlining a database will drop the database completely out of memory reducing server
overhead of having large, offline databases.
```

* MODIFIED: Increased packet size in SNARL protocol changing maximum size from 64k to 16.7M bytes. This raises the maximum size of a triple or query that can be exchanged to the protocol to nearly 17MB and should avoid any issues with loading datasets with large literals. Earlier versions of the SNARL protocol from 0.9.5 and earlier are not compatible with the new format.
* MODIFIED: Storage format of RDF concepts loaded into Stardog has been changed. This is a backwards incompatible change in format and no migration is possible.
* MODIFIED: Significant performance increase in loading throughput through better utilization of system resources to perform concurrent loading and in a more compact on disk format for some parts of the database contents.
* MODIFIED: Handling of typed literal values in query evaluation resulting in performance improvements in queries which perform math & comparison operations on datatyped literals.
* MODIFIED: Reduced overhead of query plan generation resulting in performance increases for smaller queries where execution time does not dominate planning time.

0.9.5 Changes
---
* FIXED: Name of index copy target is validated so you cannot copy a database and give it an invalid name.
* FIXED: Printing result of ASK queries in the CLI.
* FIXED: Format parsing for construct queries on the CLI where unrecognized format types were resulting in NullPointerExceptions.
* FIXED: Databases which fail to initialize are no longer added to the server.
* FIXED: Better cleanup of databases which failed to be created.
* FIXED: Query explain over HTTP POST works and no longer returns a NOT_ACCEPTABLE response.
* MODIFIED: Increased performance of database copy.
* MODIFIED: Clearer error message when the server sub-command is mistyped.
* MODIFIED: Improved access to disk indexes resulting in significant query performance improvements in many workloads.
* MODIFIED: Clearer output messages from the 'drop' CLI command.

0.9.4 Changes
---

* FIXED: Order by will correctly use secondary, tertiary, etc. orderings.
* FIXED: Null values in sorting are correctly sorted as the lowest value.
* FIXED: Resolved issue related to re-using a variable multiple times in the same BGP, correct results are now produced.
* FIXED: Changed handling of Channel close events in database server to avoid endless propagation back to the client, particularly on Windows.
* FIXED: Database cleanup on close events to avoid corrupting metadata.
* FIXED: Handling of type errors in regex filter.
* MODIFIED: New algorithm for handling ?x rdf:type ?o and :inst rdf:type ?o BGP's in queries with reasoning resulting in significant performance improvements.
* MODIFIED: ChannelFactory is shared amongst all clients in the same JVM to avoid OOM errors due to an issue with Netty (NETTY-424).
* MODIFIED: Login process to obtain a connection to a database changed to reduce connection overhead.
* MODIFIED: Shiro session timeout handling to avoid erroneous timeouts during long processes, such as bulk loads.

0.9.3 Changes
---

* MODIFIED: Server start script gracefully handles the situation when STARDOG_HOME does not exist, is not a directory, or is not writable
* MODIFIED: Connection string parameters are now validated, errors are thrown when unknown parameters are provided.
* MODIFIED: Return result of empty query results in SPARQL-JSON format when accessing the HTTP query endpoint directly. Returning an empty binding set instead of empty JSON.
* MODIFIED: Checking of Accept headers for the query HTTP endpoint to better handle the case when multiple Accept headers are sent to the endpoint, but only one is relevant for the type of results to be returned.
* MODIFIED: Reporting of query execution time of select queries in CLI to better reflect total execution time.
* FIXED: Resolved issue in query generation arising from a variable being used in a BGP, context statement, and within an optional causing no results to be returned.
* FIXED: Bug in query planner where a join variable was also used in a filter with an equality test.

* FIXED: Bug in ICV when datatype restrictions are used in conjunction with a datatype property
* FIXED: Dropping database on Windows now works; Windows was holding files open and we could not delete them.
* FIXED: NPE when retrieving an unknown option for a database
* FIXED: Inferences were missed because hasValue restrictions were incorrectly filtered in RL & EL profiles

0.9.2.1 Changes
---

* FIXED: NPE when retrieving user permissions

0.9.2 Changes
---
* MODIFIED: Security layer implementation changed to provide better performance when manipulating user, role, and permission information
* MODIFIED: Reasoning & ICV interfaces to be more consistent along consistency checking & validation
* MODIFIED: Major improvements to query re-writing performance. Re-writes are up to 3x faster, resulting in a noticeable improvement in executing queries with reasoning
* MODIFIED: Improved CLI help
* MODIFIED: Friendlier error messages when invalid SPARQL queries are provided in the command line
* MODIFIED: More verbose startup & shutdown logging information
* MODIFIED: Better output for ICV validation when the database is trivially valid
* MODIFIED: A solution to quickly find nearly optimal query plans in case plan optimization takes too long
* MODIFIED: Improved hash join algorithm for faster processing in case there are multiple alternative join keys
* FIXED: Bug in query plan caching that could result in incorrect plans being used for similar queries
* FIXED: Fixed cardinality estimations for some named graph scans
* FIXED: Significantly better cardinality estimations for large plans, particularly involving multiple unions. Can result in improvements of up to 50% when bad joins can be avoided.
* FIXED: Bug in ICV when getting constraints for a new database that does not yet have any constraints
* FIXED: NPE that can arise during query evaluation after loading a large dataset
* FIXED: Searchable configuration property can now be changed after creation
* FIXED: Database load errors no longer prevent kernel from starting
* FIXED: Auto-completion for local files in Stardog Shell works regardless of other parameters used in the command

0.9.1 Changes
---
* ADDED: Stardog Shell
* MODIFIED: Auto-update of statistics now lockless
* MODIFIED: Made visible extension point for custom SPARQL functions.
* FIXED: Cardinality estimation for queries across named graphs

0.9 Changes
---

* ADDED: new native protocol, SNARL, based on Netty and Google's Protocol Buffers
* ADDED: new system catalog for all system info, metadata, etc.
* ADDED: new search system over RDF literals, integrated with SPARQL query eval
* ADDED: new server management: online, offline databases; better server startup, shutdown, and locking
* ADDED: CLI now supports remote db administration, many new subcommands
* ADDED: new Quartz-based job scheduler with cron expression scheduling
* ADDED: new, optimized index support for named graphs and triples-only databases
* ADDED: new, differential write indexes with customizable merge thresholds
* ADDED: new customizable TBOX (i.e., reasoning schema or ontology) extraction & management
* ADDED: new Stardog configuration system, including database creation templates
* ADDED: new reasoning services (explanation, satisfiability, consistency checking) available programmatically
* ADDED: reasoning extended to named graphs
* ADDED: ICV validation extended to named graphs
* ADDED: new or improved support for NQUADS, Trig, TSV, and CSV
* ADDED: improved concurrency in data loading
* ADDED: consistent use of <tt>mmap()</tt> for better performance

* ADDED: load GZIP-compressed files directly, including ZIP files with multiple RDF
* MODIFIED: new security system, including user management, granular roles and permissions model
* MODIFIED: new statistics computation & recomputation algorithms
* MODIFIED: updated documentation
* FIXED: many (many!) known bugs fixed

NOTE: 0.9 introduces backwards incompatible changes in the on-disk format of Stardog databases. The migrate command
in the CLI will not be able to migrate 0.7.3 or earlier databases to the current format. You will need to recreate
the databases if you have an existing Stardog installation.

0.7.3 Changes
---

* ADDED: New SNARL Connection view, ReasoningConnection which exposes the functions provided by the Stardog reasoner.
* ADDED: Support in ReasoningConnection for obtaining the explanation for an inference.
* ADDED: Explanations of IC violations as the expected and/or missing statements that make up the violation
* ADDED: New example for using explanation facilities, updating ICV example.
* MODIFIED: Improved error messages when CLI fails to connect to a database.
* FIXED: Bug in the handling of reflexive properties when they involve TOP
* FIXED: Bug in initialization of TOP operator with bound objects, resolves NPE in execution
* FIXED: Bug in bnode handling in TBox extraction

0.7.2 Changes
---

* ADDED: RDFS & DL reasoning types. DL reasoning type supports schema-only queries.
* ADDED: Execution optimizations for when TOP appears in a query due to reasoning
* MODIFIED: Sesame Repository/Sail implementation to use a SNARL Connection per Sail/RepositoryConnection rather than per Repository.
* MODIFIED: Better formatting of query results in CLI query command including new formatting options.
* FIXED: Retrieving ranges of data properties in reasoner.
* FIXED: Read transaction isolation bug for mulitple connections against the same database.
* FIXED: Bug related to incorrect re-association of Shiro auth credentials with pooled connections.

0.7.1 Changes
---

* ADDED: Canonicalization of literal values on input, either bulk loading or normal add operations.
Canonlicalization can be disabled when creating the database if it is not desired, but is on by default.
* MODIFIED: CLI connection strings no longer assume you are connecting to an embedded store when no
protocol is supplied. You must now specify a protocol.
* FIXED: ExpressionFactory is no longer stripped from the core jar file
* FIXED: Multi-line queries work with CLI now
* FIXED: Bug in loading IC's from file, occasionally constraints could be malformed

0.7 Changes
---

* ADDED: Bulk insert/removal performance on existing databases greatly improved via concurrent application of changes to the index and the introduction of a differential index.
* ADDED: Stardog reasoner now supports OWL2 EL and RL reasoning types. These can be specified at connection time and EL & RL reasoning is performed at query time the same as QL.
* ADDED: Support for 'hybrid' and TBox queries for all reasoning types.
* ADDED: Support for Integrity Constraints.
* MODIFIED: Propogation of reasoner state amongst open connection to the reasoner, state is better shared across connections requiring less overhead to maintain the up to date state.
* MODIFIED: 'delete' was renamed to 'drop' in the command line.
* FIXED: Bug in two-phase commits for changes to a reasoner

* NOTE: The version of Sesame changes with this release from 2.3.3 to 2.3.4, which is not

an official release. There was a bug [1] in the serialization of Sesame result sets for select queries using their binary format which we discovered while working on Stardog 0.7. The fix for SES-852 was included in their 2.6 branch, so we had to backport the change to Sesame 2.3.3; we called that version Sesame 2.3.4.

[1] http://www.openrdf.org/issues/browse/SES-852

0.6.10 Changes
---

* FIXED: Fixed an issue with not always consuming the HTTP response which could lead to not releasing HTTP connections to the pool and it its eventual exhaustion.
* MODIFIED: Improved algebraic rewriting of query plans which avoid multiple evaluation of the same part of the query, e.g., the operator in both parts of a UNION.
* MODIFIED: A better handling of the UNION operator by sorting its output which facilitates joins with other operators in the query plan.
* MODIFIED: A better handling of deep chains of nested joins by maintaining a certain order of intermediate results produced by hash joins and avoiding page loads.
* MODIFIED: An improved, thread-safe implementation of the query plan cache, which is now concurrently accessible from all connections. This allows the server to re-use a previously optimized query plan for a structurally equivalent query executed by another client.
* ADDED: an init startup script for a Stardog server so that it can automatically start with other system services at the boot time.

0.6.9 Changes
---

* FIXED: Fixed an issue pertaining to QL re-writes with optionals that use a filter which was causing incorrect results for some QL queries (ticket #44).
* FIXED: Fixed query evaluation where the presence of a particular combintation of union, optional, and distinct can lead to non-distinct result sets (ticket #48).
* FIXED: Fixed the passing of the disableSecurity flag into the HTTP server
* FIXED: Fixed a bug in the rendering of SPARQL queries using isBlank with remote endpoints
* FIXED: Fixed a bug in query evaluation arising from a certain combination of order by, limit, and optional where the sort key is on a variable bound by an optional pattern.
* FIXED: User specified query bindings specified via Query.parameter are correctly serialized and sent to the remote server.
* MODIFIED: Query rewriting now fails when a query has BGP's that are not supported by the reasoner; this is strict mode. Strict mode can be disabled and the reasoner will re-write only the BGP's it supports. This introduces a JVM flag 'strictReasoning' for controlling this behavior.
* MODIFIED: HTTP connection will now switch over to HTTP POST if the SPARQL query is very long to avoid any potential URL overflow issues.
* ADDED: CLI now includes an 'export' function to dump the contents of part or all of a Stardog database to RDF.

0.6.8 Changes
---

* ADDED: Updated POM to include Avro
* ADDED: Avro schema included in distribution.
* FIXED: Fixed a bug in the caching of the current list of databases, could still access deleted databases in some circumstances. (ticket #42)
* FIXED: Fixed an bug in the handling of unions in the QL reasoner. Nested unions were occasionally getting duplicated leading to incorrect results without the distinct modifier. (ticket #43, #46)
* MODIFIED: Better handling of constant values in construct queries decreasing the time it takes to create and return the Statements constructed by the query.
* MODIFIED: Increased the speed of some index scans by avoiding some page reads
* MODIFIED: Better execution plan re-use & caching, similar to Prepared Statements, in structurally equivalent plans.
* MODIFIED: Significantly improved join-join cardinality estimations.
* MODIFIED: Query optimizer does a better job with join selection and can often pick a better join for the execution.
* MODIFIED: Avro & HTTP connections are eagerly validated by the driver so invalid connections will fail on connect rather than on first action.
* MODIFIED: Transaction ID generation changed to reduce chance of ID collision
* MODIFIED: Internal connection management, both with raw database connections and HTTP based connections to address deadlock issues in highly concurrent environments

```
  0.6.7 Changes
-------------

* FIXED: HTTP error in construct queries via the Jena bindings
* ADDED: New CLI command 'server' which can start any of the installed Stardog servers
* ADDED: New Avro-based RPC protocol for Stardog Connections
* ADDED: Maven POM file included in distribution

0.6.6 Changes
-------------

* FIXED: incorrect query evaluation results in some corner cases involving OPTIONAL
* MODIFIED: Improved query evaluation performance with a new query planner
* MODIFIED: command-line interface to accept global args: --home, --change-buffer-size, --
disable-security
* MODIFIED: command-line add, remove, create commands to include --strict-parsing option

0.6.5 Changes
-------------

* FIXED: Bug in query evaluation where we were missing results when the first
entry in an iterator in a disk index occurs on a page boundry

* ADDED: Support for full-text search via our semantic search engine Waldo.

0.6.2 Changes
-------------

* FIXED: Typo in HTTP protocol, Trig & Trix mimetypes were switched.

* FIXED: Bug in left outer joins when the right hand operator was initially
empty causing the first solution set to be missed.  (ticket #39)

* FIXED: Resolved IllegalArgumentException which was incorrectly thrown
from the Jena integration when a query w/ optionals was executed against
a remote Stardog database returning null values in the response.

* FIXED: Bug in detection of special BNode syntax in queries for ensuring
BNode stability in queries.

0.6.1 Changes
-------------

* FIXED: Debugging exception thrown when using log4j bindings for SLF4j with
the logging level set to debug. (ticket #36)

* FIXED: Issue with calcuation of optionals in joins when the optional binding
is at the beginning or end of an iteration. (ticket #34)

* FIXED: Resolved synchronization issue for connections to an index that
are fetched from the internal connection pool.  Resolves getting already
closed connections from the pool. (ticket #37)

* FIXED: Datatyped literals used in filters were getting munged during
QL re-writes resulting in an exception.  (ticket #35)

* MODIFIED: Iteration over disk indices modified to more efficiently
calcuate which pages are actually required to answer the query and avoid
loading intermediate pages.

0.6 Changes
-----------

* FIXED: Bug pertaining to loading datasets which use literals larger
than 8k.

* FIXED: Bug in the writer locks that could allow multiple writers access
to an index in a multithreaded environment, which usually lead to
corruptted indexes (ticket #29).
```

* FIXED: Issue with the CLI's database create command, multiple calls in rapid succession to create databases caused some of the new databases to be lost (ticket #32).

* FIXED: QL query rewriter was mangling the variables used within the query in certain situations causing query results to be incorrect (ticket #30).

* FIXED: QL query rewriter was not correctly respecting variable substitutions with datatyped literals and was subsequently losing the variable binding specified by the user (ticket #31).

* FIXED: FROM/FROM Named clauses were incorrectly set to the HTTP server

* MODIFIED: Improved index page load speed.

* MODIFIED: Reduced memory requirements in the indexes wrt to cache overhead

* MODIFIED: Persistence of memory indexes to disk is now done in parallel to the greatest extent possible yield a significant increase in save (and load) performance. This required changing the on disk format of memory indexes. Old versions of memory indexes are unfortunately, backwards incompatible with these updates. We apologize for the inconvenience.

* MODIFIED: On format of disk-based indexes. This helps us resolve some issues we had with the previous format, and increases load speed, but the index format is backwards incompatible with the previous indexes. However, we provide a migration utility in the command line program for changing old disk indexes to the new format. See the CLI for more information on migrating old indexes (or ping the mailing list).

* MODIFIED: Decreased memory usage in the query engine

* MODIFIED: Added a change buffer in the layer between the SNARL API and the underlying index it is connected to. This allows most changes within a transaction to be pushed into the index at once increasing the performance of commits as the changes can mostly be applied in bulk.

0.5.3 Changes
---

* ADDED: Internal buffer for changes via the Sesame API, triples are no longer added to the Stardog database one at a time via the SAIL API. They are buffered until the buffer is full, a read operation occurs, or the transaction is committed.

* FIXED: Internal bookkeeping for the state of the system information, timestamp for last update time is correctly updated and you should not get the out of sync warnings printed to the console.

* FIXED: Incorrect return code for parse errors in HTTP protocol.

* FIXED: NPE thrown by TBox extractor when encountering non object/data properties.

* FIXED: Correct behavior for SPARQL queries w/ QL reasoning that uses the 'GRAPH' keyword. BGP's in graph contexts are not rewritten as inferences are put into the default, no context, graph. Hybrid queries that include some patterns against the default graph and others against patterns in graph contexts are correctly partially rewritten. (ticket #27)

0.5.2 Changes
---

* MODIFIED: Add, Remove and Clear Contexts now take an optional query parameter, "graph-uri" which specfies the named graph the operation should be executed against. This deprecates the use of this parameter

in the actual URL. See HTTP_PROTOCOL.txt for additional details.

* MODIFIED: Temp files used to track changes during transactions and
to buffer results coming into HTTP server are more aggresively removed.

* FIXED: Specifying 'null' as the context to remove via the Connection
API now actually deletes the default context rather than the entire
contents of the database.

* FIXED: Cast exception caused during QL query execution when the usage
of the property in the query does not match the type (object v data)
of the property in the kb. (ticket #24)

* FIXED: NPE during evaluation of a query with QL reasoning enabled
when the query has a optionals nested inside of a graph statement. (ticket #26)

* FIXED: Could get NPE's during filter evaluation (compare expressions only)
if one of the variables in the expression was not bound to any value
in a BGP.

* FIXED: Malformed literals in inserts could come back and cause errors
when being removed. Errors are now thrown when a malformed literal
(invalid language tag or inconsistent xsd datatype) is inserted into
the database. (ticket #22 & #25)

* FIXED: Resolving BGP's across contexts using wildcards (e.g. graph ?g { ... })
was losing information between the contexts leading to incorrect results
and/or null values in the results. (ticket #20 & #21)

* ADDED: Gremlin integration via our Sesame Sail implementation and the
Blueprints API. See the README for more details on the integration.

* ADDED: statements(subject, predicate, object, context) to the Remover API
and deprecated statement(...); the latter only accepted non-null values, while
the former does accept nulls as wildcards, so it is safe to replace all
usages of one with the other. statement(...) is marked as deprecated and
schedule for removal from the API no earler than 0.6.

0.5.1 Changes
---

* FIXED: Deleting statements with wildcards via a StardogRepository.

* ADDED: Support for the Sesame RepositoryFactory & Config APIs so
Stardog can be used via the OpenRDF shell and Workbench.

* ADDED: Native connection pools for Stardog connections. See the
examples for code demonstrating how to configure and use them.

0.5 Changes
---

* FIXED: Query objects reference to database was not refeshed after
commit or rollback of transaction. (ticket #15)

* FIXED: Bug in queries with graph contexts using unbound variables
via an HTTP connection; the variable was getting mangled.

* MODIFIED: Introduced new Flex Indexes into the Database structure.

* ADDED: Shiro based security layer

0.4.10 Changes
---

* FIXED: Bug in the concurrent calcuation of statistics where a latch could
get stuck when certain types of exceptions occured during processing

* FIXED: Issue in Remove CLI where description did not match behavior;

Specifying only a context now deletes the context as stated in the documentation. (ticket #17)

* FIXED: Bug in removing a context within a durable transaction that caused the transaction record to get corrupted. (ticket #16)

* MODIFIED: Changed Add CLI to print an error message if no files are specified to add to the database.

* MODIFIED: Removed default parse format of RDF/XML from add & remove operations. If a specific format is specified, that is used, otherwise the format based on the file extension will be used. This brings the behavior of add & remove inline with create. RDF/XML is still used as the fallback when no format is specified and there is a non-standard extension on the file.

* MODIFIED: Modified how some filters are evaluated greatly increasing the speed in which basic (and/or/not & compares) filters are executed.

0.4.9 Changes
---

* ADDED: Handling of the TOP concept in the OWL 2 QL reasoner

* ADDED: New query operator which provides more efficient evaluation of queries where order by and limit are used.

* ADDED: System information regarding which databases are present will be reloaded from disk when it is changed by an external source.

* ADDED: JVM flag to disable strict mode of parsing.

* FIXED: Encoding of graph URIs in remove context calls to HTTP server.

* FIXED: Typo in route which effectively hid the remove context endpoint in the HTTP server.

* FIXED: Mangling of ASK queries when sending them to the HTTP server. (ticket #13)

* FIXED: Typo in help information in the add/remove data commands in the CLI.

* FIXED: BOM markers at the start of files will be gracefully handled instead of failing to parse.

* FIXED: If copy of system info on disk is more recent than cached copy in memory, it will not be overwritten on shutdown. (ticket #11)

* FIXED: UnsupportedOperationException thrown when clearing a context within a transaction. (ticket #12)

* FIXED: IllegalArgumentException thrown when literals are used in query atoms going through rewriting. (ticket #9)

* FIXED: ClassCastException when non-QL axioms are used. (ticket #8)

* FIXED: Issue with deleting directories when STARDOG_HOME contains relative paths. (ticket #6)

* FIXED: NullPointerException when top level scope contains an empty set of query atoms. (ticket #7)

* FIXED: Closing temporary files in the data directory used for some file operations.

* FIXED: Concurrent modification exception from an iterator when persisting memory indexes to disk asynchronously.

* FIXED: HTTP protocol was dropping context information from queries.

* MODIFIED: Queueing of jobs during concurrent loading to avoid running out of memory when processing large datasets. This also reduces the amount of memory used during loading.

* MODIFIED: Altered algorithm to get better estimates of join cardinalities.

* MODIFIED: Memory databases to use new statistics algorithm.

0.4.8 Changes
---

* ADDED: Creation of db indices is done in parallel. It is an experimental feature which can be disabled via the JVM flag (disableConcurrent); note, that concurrent loading requires more RAM than serial loading. Either give Stardog more RAM (i.e., as much as you cann spare) on startup or pass the disableConcurrent flag.

* FIXED: Query atoms of the form { ?a ?a ?b } now return correct results

* FIXED: Bug in multi-key joins

* FIXED: Bug in QL reasoning optimizer for detecting empty joins

* FIXED: File exhaustion bug. When Sesame integration was used with the HTTP server, the server did not always close the temporary files used to track the incoming edits causing some systems to report too many files were open.

* FIXED: Bug in query evaluation where the named graph bound in a pattern could cause an exception during execution.

* FIXED: Cardinality estimates in index statistics

* FIXED: bug in stardog.sh script so that it can run anywhere (thanks Alin Dreghiciu for this fix)

* MODIFIED: The name of the file being added/removed from a database in the command line is now printed.

0.4.7 Changes
---
* Fixed an NPE in the HTTP server (reported by Brian Sletten)

* make the QL axioms parser more strictly respect the OWL spec, especially w/r/t to the set of datatypes that QL supports

Validating RDF with OWL Integrity Constraints

Authors:

Héctor Pérez-Urbina (http://clarkparsia.com/about/profiles/hector/), Clark & Parsia, LLC
Evren Sirin (http://clarkparsia.com/about/profiles/evren/), Clark & Parsia, LLC
Kendall Clark (http://clarkparsia.com/about/profiles/kendall/), Clark & Parsia, LLC

Abstract

This document proposes a method for validating Semantic Web and Linked Data by providing an alternative, integrity constraint (IC) semantics for OWL. A model-theoretic semantics based on the Closed World Assumption and a weak variant of the Unique Name Assumption is given for OWL axioms that are thereby interpreted as ICs. The document includes a structural specification in order to augment an ontology with a set of OWL ICs and a brief description of possible implementation approaches.

Table of Contents

1. Introduction

This document proposes a method of constraining and validating Semantic Web and Linked Data (i.e., RDF) instance data using Integrity Constraints (ICs) modeled as OWL axioms. The proposal enables an OWL ontology to be interpreted as a set of ICs; i.e., checks that must be satisfied by the information explicitly present or information that may be inferred. We define a model-theoretic semantics for OWL ICs based on the Closed World Assumption and a weak variant of the Unique Name Assumption and briefly describe

feasible implementation strategies.

In some use cases and for some requirements, OWL users intend OWL axioms to be interpreted as ICs. However, the direct semantics of OWL [***OWL 2 Direct Semantics** (#owl-2-direct-semantics)*] does not interpret OWL axioms in this way; thus, the consequences that one can draw from such ontologies differ from the ones that some users intuitively expect and require. In other words, some users want to use OWL as a validation or constraint language for RDF instance data, but that is not possible using OWL software that correctly implements the existing semantics of OWL. This document addresses that situation by providing a different semantics—compatible with the existing semantics—for OWL axioms which may be used, together with appropriate software, to validate RDF instance data.

To see the nature of the problem, consider an OWL ontology that describes terms and concepts regarding the product inventory of a supermarket. The ontology includes the classes *Product* and *Provider*, the object property *hasProvider*, and the data property *hasID*. Suppose we want to impose the following ICs on the data:

I. Each product must have an ID
II. Only products can have IDs
III. Products must not have more than one provider

These constraints could be interpreted in the following way:

I. Whenever an instance $product_i$ of *Product* is added to the ontology, a check should be performed to verify whether the ID of $product_i$ has been specified; if not, the update should be rejected.
II. Whenever an instance $\langle product_i, ID_i \rangle$ of *hasID* is added to the ontology, a check should be performed to verify whether $product_i$ is an instance of *Product*; if not, the update should be rejected.
III. Whenever an instance $\langle product_i, provider_j \rangle$ of *hasProvider* is added to the ontology, a check should be performed to verify whether another provider $provider_j$ has been specified for $product_i$; if so, the update should be rejected.

These constraints can be concisely and unambiguously represented as OWL axioms:

I.
```
Class: Product
  hasID some literal
```

II.
```
DataProperty: hasID
  Domain: Product
```

III.
```
ObjectProperty: hasProvider
  Characteristics: Functional
```

However, these axioms will not be interpreted as checks by software which implements the standard OWL semantics. In fact, according to the standard OWL semantics, we have that:

I. Adding a product without an ID to the ontology does not raise an error, but leads to the inference that the product in question has an unknown ID.
II. Adding a tuple $\langle product_i, ID_i \rangle$ to the ontology without $product_i$ being an instance of *Product* does not raise an error, but leads to the inference that $product_i$ is an instance of *Product*.
III. Adding a tuple $\langle product_i, provider_j \rangle$ having specified a previous provider $provider_j$ for $product_i$ does not raise an error, but leads to the inference that $provider_i$ and $provider_j$ denote the same individual.

In some cases, users want these inferences; but in others, users want integrity constraint violations to be detected, reported, repaired, etc.

OWL adopts the Open World Assumption (OWA) and does not adopt the Unique Name Assumption (UNA). These design choices make it very difficult to treat these axioms as ICs. On the one hand, due to OWA, a statement must not be inferred to be false on the basis of failures to prove it; therefore, the fact that a piece of information has not been specified (e.g., a product's ID) does not mean that such information does not exist. On the other hand, the absence of UNA allows two different constants to refer to the same individual (e.g., $provider_i$ and $provider_j$).

The standard interpretation of OWL axioms that are intended to be interpreted as ICs is inappropriate for some use cases and applications; therefore, it is useful to define an alternate semantics for OWL based on IC. An IC semantics together with associated software will increase the number of satisfied users of OWL because OWL and the software will then behave as those users intuitively expect and require.

As formally defined in **Section 2 (# 2)** , our approach allows for a standard OWL ontology *O* to import a set of IC ontologies—OWL ontologies that are to be interpreted as ICs. Note that the IC semantics for OWL defined in this document is a strict extension of the standard OWL semantics: in case *O* imports no IC ontology, then *O* should be interpreted as a standard OWL ontology.

2. Structural Specification

An OWL ontology that is to be interpreted as a set of ICs is called an *IC ontology*. We slightly extend the structural specification of OWL in order to allow ontologies to import a set of IC ontologies. We do so by introducing a new annotation property which is defined analogously to owl:imports—the annotation property that is used to import standard ontologies defined by OWL 2 [***OWL 2 Specification** (#ref-owl-2-specification)*].

We use an annotation property that resembles owl:imports with a different namespace: http://www.w3.org/Submission/owlic/. In the following, we denote this annotation property as ic:imports for brevity.

An example usage of this annotation property is given in the following:

```
Namespace(ic = <http://www.w3.org/Submission/owlic/>)
Ontology(<http://www.example.com/instanceOntology>
        Import(<http://www.example.com/schemaOntology>)
        Annotation(ic:imports <http://www.example.com/constraintsOntology>)
        ...
)
```

where instanceOntology imports the standard axioms of schemaOntology and imports constraintsOntology as an IC ontology. This import approach to relating ICs to other OWL ontologies gives enough flexibility to users without too much maintenance cost and negligible impact on existing tools. See **Implementation Remarks (# 5)** for a more detailed discussion of this design choice.

An OWL ontology can import a set of IC ontologies via ic:imports. An IC ontology can import a set of IC ontologies via ic:imports as well. And, of course, an IC ontology can import a set of standard ontologies via owl:imports as usual.

The *import closure* of an IC ontology is defined in the same vein as the import closure for standard OWL ontologies. The *IC import closure* of a standard or IC ontology O is a set containing all the IC ontologies that O imports via the ic:imports annotation property. The *IC closure* of a standard or IC ontology O is the smallest set that contains all the axioms from each ontology O' in the IC import closure of O.

3. OWL IC Semantics

We refer to the definitions of datatype map, vocabulary, and OWL interpretation and model in OWL 2 [***OWL 2 Direct Semantics (#owl-2-direct-semantics)***].

3.1 IC-Interpretations

Let $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$ be a datatype map and let $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ be a vocabulary over D. An *IC-interpretation* $\mathcal{I} = (\Delta_I, \Delta_D, I, U, \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for D and V is an 11-tuple with the following structure:

- Δ_I is a nonempty set called the *object domain*.
- Δ_D is a nonempty set disjoint with Δ_I called the *data domain* such that $(DT)^{DT} \subseteq \Delta_D$ for each datatype $DT \in V_{DT}$.
- $I = (\Delta_I, \Delta_D, \cdot^{C_I}, \cdot^{OP_I}, \cdot^{DP_I}, \cdot^{I}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ is an OWL interpretation for D and V.
- $U = \{U_1, U_2, \ldots, U_n\}$ is a set where each $U_j = (\Delta_{I_j}, \Delta_{D_j}, \cdot^{C_j}, \cdot^{OP_j}, \cdot^{DP_j}, \cdot^{I_j}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for $1 \leq j \leq n$ is an OWL interpretation for D and V.
- $\cdot^C$ is the *class interpretation function* that assigns to each class $C \in V_C$ a subset $(C)^C \subseteq \Delta_I$ such that

- $(owl{:}Thing)^C = \Delta_I$,
- $(owl{:}Nothing)^C = \varnothing$, and
- $(C)^C = \{x^I \mid x \in V_I$ and for each $U_j \in U$ we have that $x^{I_{U_j}} \in (C)^{C_j}\}$.

- $\cdot^{OP}$ is the *object property interpretation function* that assigns to each object property $OP \in V_{OP}$ a subset $(OP)^{OP} \subseteq \Delta_I \times \Delta_I$ such that
  - $(owl{:}topObjectProperty)^{OP} = \Delta_I \times \Delta_I$,
  - $(owl{:}bottomObjectProperty)^{OP} = \varnothing$, and
  - $(OP)^{OP} = \{(x^I, y^I) \mid x \in V_I$, y $\in$ V_I, and for each $U_j \in U$ we have that $(x^{I_{U_j}}, y^{I_{U_j}}) \in (OP)^{OP_j}\}$.
- $\cdot^{DP}$ is the *data property interpretation function* that assigns to each data property $DP \in V_{DP}$ a subset $(DP)^{DP} \subseteq \Delta_I \times \Delta_D$ such that
  - $(owl{:}topDataProperty)^{DP} = \Delta_I \times \Delta_D$,
  - $(owl{:}bottomDataProperty)^{DP} = \varnothing$, and
  - $(DP)^{DP} = \{(x^I, lt^{LT}) \mid x \in V_I$, $lt \in V_{LT}$, and for each $U_j \in U$ we have that $(x^{I_{U_j}}, lt^{LT}) \in (DP)^{DP_j}\}$.
- $\cdot^I$ is the *individual interpretation function* that assigns to each individual $a \in V_I$ an element $(a)^I \in \Delta_I$.
- $\cdot^{DT}$ is the *datatype interpretation function* that assigns to each datatype $DT \in V_{DT}$ a subset $(DT)^{DT} \subseteq \Delta_D$ such that
  - $\cdot^{DT}$ is the same as in D for each datatype $DT \in N_{DT}$, and
  - $(rdfs{:}Literal)^{DT} = \Delta_D$.
- $\cdot^{LT}$ is the *literal interpretation function* that is defined as $(lt)^{LT} = (LV, DT)^{LS}$ for each $lt \in V_{LT}$, where LV is the lexical form of lt and DT is the datatype of lt.
- $\cdot^{FA}$ is the *facet interpretation function* that is defined as $(F, lt)^{FA} = (F, (lt)^{LT})^{FS}$ for each $(F, lt) \in V_{FA}$.

The extensions of $\cdot^C$, $\cdot^{OP}$, and $\cdot^{DT}$ to class expressions, object property expressions, and data ranges respectively, are defined analogously to OWL 2 [***OWL 2 Direct Semantics*** *(#owl-2-direct-semantics)*]. For example, we extend $\cdot^C$ to the class expression $ObjectIntersectionOf(CE_1 \ldots CE_n)$ as $(CE_1)^C \cap \ldots \cap (CE_n)^C$. However, we extend $\cdot^C$ to $ObjectComplementOf(CE)$ as $\{x^I \mid x \in V_I\} \setminus (CE)^C$—that is, the complement of a class expression is defined with respect to the set of *named individuals* as opposed to the object domain. The complete extensions for $\cdot^C$, $\cdot^{OP}$, and $\cdot^{DT}$ can be found in the **Appendix** *(#appendix)*.

3.2 Axiom Satisfaction

Satisfaction of an IC-interpretation $\mathcal{I}$ with respect to a given axiom is defined analogously to satisfaction of standard intepretations defined in OWL 2 [***OWL 2 Direct Semantics*** *(#owl-2-direct-semantics)*]. For example, $\mathcal{I}$

satisfies the axiom *SubClassOf(CE$_1$ CE$_2$)* if $(CE_1)^C \subseteq (CE_2)^C$. The complete definitions for axiom satisfaction can be found in the **Appendix (#appendix)**.

3.3 Axiom IC-Satisfaction

Let $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$ be a datatype map and let $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ be a vocabulary over D.

- Let I be an OWL interpretation for D and V. With E_I we denote the set of equality relations between named individuals satisfied by I. That is, $E_I = \{ \langle a, b \rangle \mid a \in V_I, b \in V_I, \text{ and } a^I = b^I \}$ where $\cdot^I$ is the individual interpretation function of I.
- Let I and J be OWL interpretations for D and V. We say that $I \lessdot_{eq} J$ iff the following conditions are satisfied:
  - for every $C \in V_C$, I satisfies $C(a)$ iff J satisfies $C(a)$
  - for every $R \in V_{OP}$, I satisfies $R(a, b)$ iff J satisfies $R(a, b)$
  - for every $S \in V_{DP}$, I satisfies $S(a, lt)$ iff J satisfies $S(a, lt)$
  - $E_I \subset E_J$
- Let O be an OWL ontology. With $Mod(O)$ we denote the set containing exactly all the models of O with respect to D and V. With $Mod_{ME}(O)$ we denote the set containing exactly all the models with minimal equality between named individuals. That is, $Mod_{ME}(O) = \{ I \mid I \in Mod(O)$ and there is no J such that $J \in Mod(O)$ and $J \lessdot_{eq} I \}$.
- Let O be an OWL ontology and α be an axiom. We say that O *IC-satisfies* α iff for all $I \in Mod_{ME}(O)$, we have that the IC-interpretation $\Gamma = (\Delta_I, \Delta_D, I, Mod_{ME}(O), \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for D and V satisfies α.

3.4 Inference Problem

We are mainly interested in the following inference problem:

Ontology Validation. Let O be an OWL ontology. We say that O is *Valid* iff for all axioms α in the IC closure of O, it holds that O *IC-satisfies* α.

4. Implementation Remarks

4.1 Implementing IC Syntax

As discussed in **Section 2 (#_2)**, we use standard OWL syntax for ICs; store ICs in a separate document; and define a new annotation property analogous to owl:imports, that will associate a standard OWL ontology

with a set of ICs defined for that ontology.

The motivation for this design choice is to minimize the effects of ICs on existing tools. From the perspective of creation and maintenance, users can continue using existing ontology authoring toolsets. For example, one can use an OWL editor to create ICs and store them in a document. The ontology for which the constraints are written can be augmented with the IC import annotation easily, since this is a standard OWL annotation. With an OWL editor that allows users to open and edit multiple ontologies at the same time, the regular ontology and the IC ontology can be edited together. Several OWL editors provide the feature to move axioms between ontologies; hence, one can easily change the interpretation of an axiom just by moving it from the regular ontology to the IC ontology.

The only issue in using an existing OWL editor is as follows: when a user opens a regular ontology that links to an IC ontology, the editor will not open the IC ontology automatically. The user needs to look at the ontology annotation and open the IC ontology manually. However, this is not a serious issue since it is a relatively simple extension for editors to recognize this annotation. It is safe to assume that such extensions will be available, especially if ICs start to be widely used.

Our approach has no impact on OWL existing reasoners that do not support ICs: since the annotation property has no semantic effect, they would not process that annotation. Therefore, there is no additional work that needs to be done to hide the ICs in order to avoid unintended inferences that would occur if they are inadvertently interpreted as regular OWL axioms.

Note that our approach does not require an IC ontology to be identified as such. However, in case an ontology is to be exclusively interpreted as a set of ICs, one might use an ontology annotation to make this fact explicit. As usual, such annotations are for informational purposes only and have no effect on the semantics.

4.2 Implementing IC Semantics

The IC semantics described in this document is strongly related to the semantics presented in a paper [***TSBM10*** *(#TSBM10)*] giving a formal integrity constraint semantics for the description logic SROIQ. Based on the correspondence between SROIQ and OWL 2 semantics [***OWL 2 Direct Semantics*** *(#owl-2-direct-semantics)*], the semantics we present here has been adapted to OWL 2 and extended to support datatypes.

As discussed in the paper, there is a close relationship between IC semantics and queries that have negation as failure (NAF) operator. This is interesting from a practical point of view because a validator for OWL IC can be implemented in a straightforward way: each axiom in an IC ontology defined with respect to an OWL ontology *O* can be effectively transformed into a SPARQL query that can be later answered over *O* using the SPARQL entailment regime that corresponds to *O*.

As a simple example of the translation, consider the IC presented in **Example 6** *(#example-6)* above:

```
Class: Supervisor
   SubClassOf: supervises some Employee
```

The translation of this IC to SPARQL would yield the following SPARQL query:

```
ASK WHERE {
   ?x rdf:type :Supervisor .
   OPTIONAL {
      ?x :supervises ?y .
      ?y rdf:type :Employee .
   }
   FILTER ( !bound( ?y ) )
}
```

If the execution of the query over an ontology *O* returns `true`, we can conclude that the IC has been violated by *O*; and, therefore, that *O* is *not* IC-valid with respect to this constraint. Note that the query uses the OPTIONAL/FILTER/!BOUND pattern to encode NAF. However, it is likely that SPARQL 1.1 [***SPARQL 1.1*** *(#SPARQL-1.1)*] will make NAF more clearly visible syntatically, perhaps via `NOT EXISTS` as in current drafts.

It has been shown that SPARQL [***SPARQL*** *(#SPARQL)*] has the same expressive power as nonrecursive datalog programs with NAF [***AG08*** *(#AG08)*]. Therefore, it is possible to translate OWL ICs to a set of rules that will be evaluated over an ontology *O*. Such rules can be written using RIF Framework for Logic Dialects [***RIF-FLD*** *(#RIF-FLD)*] with the `Naf` operator:

```
Forall ?x ?y (
    invalid() :- And (
        ?x[rdf:type -> :Supervisor]
        Naf And (
            ?x[:supervises -> ?y]
            ?y[rdf:type -> :Employee] )))
```

This rule uses the `Naf` operator for encoding NAF and defines an arbitrary RIF predicate `invalid` to detect the condition that an ontology *O* is invalid with respect to ICs. Implementations would be free to choose a different name for the predicate.

Details of the translation are out of the scope here; interested readers are referred to the formal semantics paper mentioned previously [***TSBM10*** *(#TSBM10)*]. Translation-based IC validation is one of many possibilities to implement IC validation and has been mentioned here as an example. An IC validator conforming to the IC semantics described here can also be implemented with different approaches.

Acknowledgements

We wish to thank the following people for their assistance: Pavel Klinov, Michael Smith, Michael Grove, Jiao Tao, and Peter Patel-Schneider. We thank members of the OWLED community, including the anonymous reviewers, who gave us very early feedback on using OWL as integrity constraints, including, most helpfully, use cases and requirements. We also acknowledge the support of NIST SBIR funding under the auspices of which this document was prepared.

References

[OWL 2 Direct Semantics]

OWL 2 Web Ontology Language: Direct Semantics *(http://www.w3.org/TR/owl2-direct-semantics/)*. Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 27 October 2009, **http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/** (http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/). Latest version available at **http://www.w3.org/TR/owl2-direct-semantics/** (http://www.w3.org/TR/owl2-direct-semantics/).

[OWL 2 Specification]

OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax *(http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/)*. Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, **http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/** (http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/). Latest version available at **http://www.w3.org/TR/owl2-syntax/** (http://www.w3.org/TR/owl2-syntax/).

[SPARQL]

SPARQL Query Language for RDF *(http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/)*. Eric Prud'hommeaux and Andy Seaborne, eds. W3C Recommendation, 15 January 2008, **http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/** (http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/). Latest version available as **http://www.w3.org/TR/rdf-sparql-query/** (http://www.w3.org/TR/rdf-sparql-query/).

[SPARQL-1.1]

SPARQL Query Language 1.1 *(http://www.w3.org/TR/2010/WD-sparql11-query-20100126/)*. Steve Harris and Andy Seaborne, eds. W3C Working Draft, 26 January 2010, **http://www.w3.org/TR/2010/WD-sparql11-query-20100126/** (http://www.w3.org/TR/2010/WD-sparql11-query-20100126/). Latest version available as **http://www.w3.org/TR/sparql11-query/** (http://www.w3.org/TR/sparql11-query/).

[TSBM10]

Integrity Constraints in OWL. Jiao Tao, Evren Sirin, Jie Bao, Deborah L. McGuinness. In Proc. of the 24th Conference on Artificial Intelligence (AAAI 2010). Atlanta, USA. 2010.

[AG08]

The Expressive Power of SPARQL. Renzo Angles, Claudio Gutierrez. In Proc. of the 7th International Semantic Web Conference (ISWC 2008). Karlsruhe, Germany. 2008.

Appendix A

A.1 Extensions of Interpretation Functions

A.1.1 Class Expressions

The class interpretation function $\cdot^{\mathcal{C}}$ is extended to class expressions as shown in Table 1. For S a set, $\#S$

denotes the number of elements in S.

Table 1. Interpreting Class Expressions

Class Expression	Interpretation $\cdot^{\mathcal{C}}$
ObjectIntersectionOf(CE_1 ... CE_n)	$(CE_1)^C \cap \ldots \cap (CE_n)^C$
ObjectUnionOf(CE_1 ... CE_n)	$(CE_1)^C \cup \ldots \cup (CE_n)^C$
ObjectComplementOf(CE)	$\{ x^I \mid x \in V_I \} \setminus (CE)^C$
ObjectOneOf(a_1 ... a_n)	$\{ (a_1)^I, \ldots, (a_n)^I \}$
ObjectSomeValuesFrom(OPE CE)	$\{ x \mid \exists\, y\colon (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \}$
ObjectAllValuesFrom(OPE CE)	$\{ x \mid \forall\, y\colon (x, y) \in (OPE)^{OP}$ implies $y \in (CE)^C \}$
ObjectHasValue(OPE a)	$\{ x \mid (x, (a)^I) \in (OPE)^{OP} \}$
ObjectHasSelf(OPE)	$\{ x \mid (x, x) \in (OPE)^{OP} \}$
ObjectMinCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} \geq n \}$
ObjectMaxCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} \leq n \}$
ObjectExactCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} = n \}$
ObjectMinCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} \geq n \}$
ObjectMaxCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} \leq n \}$
ObjectExactCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} = n \}$
DataSomeValuesFrom(DPE_1 ... DPE_n DR)	$\{ x \mid \exists\, y_1, \ldots, y_n\colon (x, y_k) \in (DPE_k)^{DP}$ for each $1 \leq k \leq n$ and $(y_1, \ldots, y_n) \in (DR)^{DT} \}$
DataAllValuesFrom(DPE_1 ... DPE_n DR)	$\{ x \mid \forall\, y_1, \ldots, y_n\colon (x, y_k) \in (DPE_k)^{DP}$ for each $1 \leq k \leq n$ imply $(y_1, \ldots, y_n) \in (DR)^{DT} \}$
DataHasValue(DPE lt)	$\{ x \mid (x, (lt)^{LT}) \in (DPE)^{DP} \}$
DataMinCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}\} \geq n \}$
DataMaxCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP} \} \leq n \}$
DataExactCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP} \} = n \}$
DataMinCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} \geq n \}$
DataMaxCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} \leq n \}$
DataExactCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} = n \}$

A.1.2 Object Property Expressions

The object property interpretation function $\cdot^{OP}$ is extended to object property expressions as shown in Table 2.

Table 2. Interpreting Object Property Expressions

Object Property Expression	Interpretation $\cdot^{OP}$
ObjectInverseOf(OP)	$\{ (x, y) \mid (y, x) \in (OP)^{OP} \}$

A.1.3 Data Ranges

The datatype interpretation function $\cdot^{DT}$ is extended to data ranges as shown in Table 3. All datatypes in OWL 2 are unary, so each datatype *DT* is interpreted as a unary relation over Δ_D — that is, as a set $(DT)^{DT} \subseteq \Delta_D$. OWL 2 currently does not define data ranges of arity more than one; however, by allowing for *n*-ary data ranges, the syntax of OWL 2 provides a "hook" allowing implementations to introduce extensions such as comparisons and arithmetic. An *n*-ary data range *DR* is interpreted as an *n*-ary relation $(DR)^{DT}$ over Δ_D — that is, as a set $(DT)^{DT} \subseteq (\Delta_D)^n$.

Table 3. Interpreting Data Ranges

Data Range	Interpretation $\cdot^{DT}$
DataIntersectionOf(DR_1 ... DR_n)	$(DR_1)^{DT} \cap \ldots \cap (DR_n)^{DT}$
DataUnionOf(DR_1 ... DR_n)	$(DR_1)^{DT} \cup \ldots \cup (DR_n)^{DT}$
DataComplementOf(DR)	$(\Delta_D)^n \setminus (DR)^{DT}$ where *n* is the arity of *DR*
DataOneOf(lt_1 ... lt_n)	$\{ (lt_1)^{LT}, \ldots, (lt_n)^{LT} \}$
DatatypeRestriction(DT F_1 lt_1 ... F_n lt_n)	$(DT)^{DT} \cap (F_1, lt_1)^{FA} \cap \ldots \cap (F_n, lt_n)^{FA}$

A.2 Satisfaction of Axioms

A.2.1 Class Expression Axioms

Satisfaction of OWL 2 class expression axioms in Γ with respect to an ontology *O* is defined as shown in Table 4.

Table 4. Satisfaction of Class Expression Axioms in an Interpretation

Axiom	Condition
SubClassOf(CE_1 CE_2)	$(CE_1)^C \subseteq (CE_2)^C$

EquivalentClasses(CE_1 ... CE_n)	$(CE_j)^C = (CE_k)^C$ for each $1 \le j \le n$ and each $1 \le k \le n$
DisjointClasses(CE_1 ... CE_n)	$(CE_j)^C \cap (CE_k)^C = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
DisjointUnion(C CE_1 ... CE_n)	$(C)^C = (CE_1)^C \cup \ldots \cup (CE_n)^C$ and $(CE_j)^C \cap (CE_k)^C = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$

A.2.2 Object Property Expression Axioms

Satisfaction of OWL 2 object property expression axioms in Γ with respect to an ontology O is defined as shown in Table 5.

Table 5. Satisfaction of Object Property Expression Axioms in an Interpretation

Axiom	Condition
SubObjectPropertyOf(OPE_1 OPE_2)	$(OPE_1)^{OP} \subseteq (OPE_2)^{OP}$
SubObjectPropertyOf(ObjectPropertyChain(OPE_1 ... OPE_n) OPE)	$\forall\, y_0, \ldots, y_n$: $(y_0, y_1) \in (OPE_1)^{OP}$ and ... and $(y_{n-1}, y_n) \in (OPE_n)^{OP}$ imply $(y_0, y_n) \in (OPE)^{OP}$
EquivalentObjectProperties(OPE_1 ... OPE_n)	$(OPE_j)^{OP} = (OPE_k)^{OP}$ for each $1 \le j \le n$ and each $1 \le k \le n$
DisjointObjectProperties(OPE_1 ... OPE_n)	$(OPE_j)^{OP} \cap (OPE_k)^{OP} = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
ObjectPropertyDomain(OPE CE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $x \in (CE)^C$
ObjectPropertyRange(OPE CE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $y \in (CE)^C$
InverseObjectProperties(OPE_1 OPE_2)	$(OPE_1)^{OP} = \{ (x, y) \mid (y, x) \in (OPE_2)^{OP} \}$
FunctionalObjectProperty(OPE)	$\forall\, x, y_1, y_2$: $(x, y_1) \in (OPE)^{OP}$ and $(x, y_2) \in (OPE)^{OP}$ imply $y_1 = y_2$
InverseFunctionalObjectProperty(OPE)	$\forall\, x_1, x_2, y$: $(x_1, y) \in (OPE)^{OP}$ and $(x_2, y) \in (OPE)^{OP}$ imply $x_1 = x_2$
ReflexiveObjectProperty(OPE)	$\forall\, x$: $x \in \Delta_I$ implies $(x, x) \in (OPE)^{OP}$
IrreflexiveObjectProperty(OPE)	$\forall\, x$: $x \in \Delta_I$ implies $(x, x) \notin (OPE)^{OP}$

SymmetricObjectProperty(OPE)	$\forall x, y$: $(x, y) \in (OPE)^{OP}$ implies $(y, x) \in (OPE)^{OP}$
AsymmetricObjectProperty(OPE)	$\forall x, y$: $(x, y) \in (OPE)^{OP}$ implies $(y, x) \notin (OPE)^{OP}$
TransitiveObjectProperty(OPE)	$\forall x, y, z$: $(x, y) \in (OPE)^{OP}$ and $(y, z) \in (OPE)^{OP}$ imply $(x, z) \in (OPE)^{OP}$

A.2.3 Data Property Expression Axioms

Satisfaction of OWL 2 data property expression axioms in I with respect to an ontology O is defined as shown in Table 6.

Table 6. Satisfaction of Data Property Expression Axioms in an Interpretation

Axiom	Condition
SubDataPropertyOf(DPE_1 DPE_2)	$(DPE_1)^{DP} \subseteq (DPE_2)^{DP}$
EquivalentDataProperties(DPE_1 ... DPE_n)	$(DPE_j)^{DP} = (DPE_k)^{DP}$ for each $1 \leq j \leq n$ and each $1 \leq k \leq n$
DisjointDataProperties(DPE_1 ... DPE_n)	$(DPE_j)^{DP} \cap (DPE_k)^{DP} = \varnothing$ for each $1 \leq j \leq n$ and each $1 \leq k \leq n$ such that $j \neq k$
DataPropertyDomain(DPE CE)	$\forall x, y$: $(x, y) \in (DPE)^{DP}$ implies $x \in (CE)^{C}$
DataPropertyRange(DPE DR)	$\forall x, y$: $(x, y) \in (DPE)^{DP}$ implies $y \in (DR)^{DT}$
FunctionalDataProperty(DPE)	$\forall x, y_1, y_2$: $(x, y_1) \in (DPE)^{DP}$ and $(x, y_2) \in (DPE)^{DP}$ imply $y_1 = y_2$

A.2.4 Datatype Definitions

Satisfaction of datatype definitions in Γ with respect to an ontology O is defined as shown in Table 7.

Table 7. Satisfaction of Datatype Definitions in an Interpretation

Axiom	Condition
DatatypeDefinition(DT DR)	$(DT)^{DT} = (DR)^{DT}$

A.2.5 Keys

Satisfaction of keys in Γ with respect to an ontology O is defined as shown in Table 8.

Table 8. Satisfaction of Keys in an Interpretation

Axiom	Condition
HasKey(CE (OPE_1 ... OPE_m) (DPE_1 ... DPE_n))	$\forall x, y, z_1, \ldots, z_m, w_1, \ldots, w_n$: if $x \in (CE)^C$ and $ISNAMED_O(x)$ and $y \in (CE)^C$ and $ISNAMED_O(y)$ and $(x, z_i) \in (OPE_i)^{OP}$ and $(y, z_i) \in (OPE_i)^{OP}$ and $ISNAMED_O(z_i)$ for each $1 \leq i \leq m$ and $(x, w_j) \in (DPE_j)^{DP}$ and $(y, w_j) \in (DPE_j)^{DP}$ for each $1 \leq j \leq n$ then x = y

A.2.6 Assertions

Satisfaction of OWL 2 assertions in Γ with respect to an ontology O is defined as shown in Table 9.

Table 9. Satisfaction of Assertions in an Interpretation

Axiom	Condition
SameIndividual(a_1 ... a_n)	$(a_j)^I = (a_k)^I$ for each $1 \leq j \leq n$ and each $1 \leq k \leq n$
DifferentIndividuals(a_1 ... a_n)	$(a_j)^I \neq (a_k)^I$ for each $1 \leq j \leq n$ and each $1 \leq k \leq n$ such that $j \neq k$
ClassAssertion(CE a)	$(a)^I \in (CE)^C$
ObjectPropertyAssertion(OPE a_1 a_2)	$((a_1)^I, (a_2)^I) \in (OPE)^{OP}$
NegativeObjectPropertyAssertion(OPE a_1 a_2)	$((a_1)^I, (a_2)^I) \notin (OPE)^{OP}$
DataPropertyAssertion(DPE a lt)	$((a)^I, (lt)^{LT}) \in (DPE)^{DP}$
NegativeDataPropertyAssertion(DPE a lt)	$((a)^I, (lt)^{LT}) \notin (DPE)^{DP}$