Parliament™ User Guide

Ian Emmons

November 1, 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Parliament™

Parliament™ is a high-performance triple store and reasoner designed for the Semantic Web.[1] Parliament's initial development was funded by the Defense Advanced Research Projects Agency (DARPA) through the DARPA Agent Markup Language (DAML) program under the name DAML DB[2] and was extended by Raytheon BBN Technologies (BBN) for internal use in its R&D programs. BBN released Parliament as an open source project under the BSD license[3] on SemWebCentral[4] in 2009. In 2018, BBN migrated the Parliament open source project to GitHub[5] under the same license.

Parliament™ is a trademark of Raytheon BBN Technologies. It is so named because a group of owls is properly called a *parliament* of owls.

1.1 Background

The Semantic Web employs a different data model than a relational database. A relational database stores data in tables (rows and columns) while the Resource Description Framework (RDF)[6] represents data as a directed

[1] http://www.w3.org/2001/sw/
[2] http://www.daml.org/2001/09/damldb/
[3] http://opensource.org/licenses/bsd-license.php
[4] http://parliament.semwebcentral.org/
[5] https://github.com/SemWebCentral/parliament
[6] http://www.w3.org/RDF/

graph of ordered triples of the form (subject, predicate, object). Accordingly, a Semantic Web data store is often called a graph store, knowledge base, or triple store.

A relational database can store a directed graph, and some graph stores are in fact implemented as a thin interface layer wrapping a relational database. However, the query performance of such implementations is usually poor. This is because the only straightforward way to store the graph with the required level of generality is to use a single table to store all the triples, and this schema tends to defeat relational query optimizers.

Early in the Semantic Web's evolution, BBN encountered exactly this problem, and so the graph store we now call Parliament was born. The goal of Parliament was to create a storage mechanism optimized specifically to the needs of the Semantic Web, and the result was a dramatic speed boost for BBN's Semantic Web programs. Since its initial conception, Parliament has served as a core component of several projects at BBN for a number of U.S. Government customers.

1.2 Parliament Architecture

Parliament combines customized versions of the Web interface and query processor of Jena[7] with a high-performance storage engine and an innovative storage layout to deliver a complete triple store solution that is compatible with the RDF, Web Ontology Language (OWL),[8] and the SPARQL Protocol and RDF Query Language (SPARQL)[9] standards from the World Wide Web Consortium (W3C) [3].

In addition, Parliament includes a high-performance rule engine, which applies a set of inference rules to the directed graph of data in order to derive new facts. This enables Parliament to automatically and transparently infer additional facts and relationships in the data to enrich query results. Parliament's rule engine currently implements all of the inference rules of RDF Schema (RDFS) plus selected elements of OWL RL.

Figure 1.1 depicts the layered architecture of Parliament. The storage layer

[7] https://jena.apache.org/
[8] http://www.w3.org/2007/OWL/
[9] http://www.w3.org/2009/sparql/wiki/Main_Page

of Parliament is written in C++, while the remainder is Java code. Integrated with the Jena query processor are a number of useful extras, such as support for named graphs, accelerated reification support, and temporal, geospatial, and numerical indexes [2, 1].

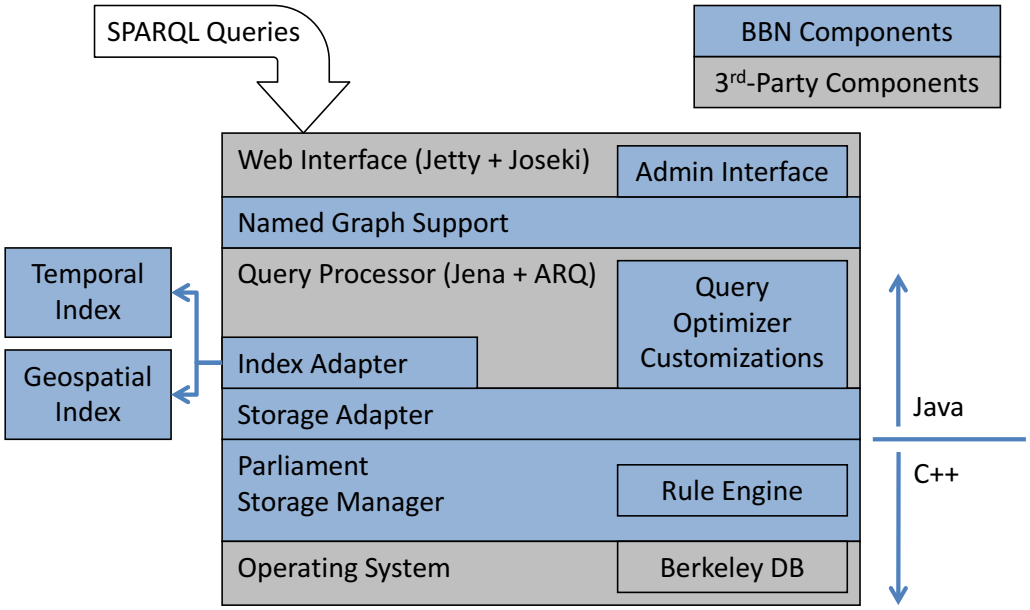

Figure 1.1: Layered Parliament Architecture

Chapter 2

Deploying and Using Parliament™

There are several ways to use Parliament. By far the most common is as a server application that exposes a SPARQL endpoint. The binary distribution directly supports this usage, running Parliament within the included Jetty servlet engine. See Section 2.1 for complete instructions.

Parliament can also be used with other servlet engines such as Tomcat and Glassfish (see Section 2.3) or as an in-process library (Section 2.4).

Pre-built binaries for common platforms are available on the Parliament Web site.[1] You can also build Parliament yourself (see Chapter 3).

This chapter starts with instructions for installing a Parliament server (Section 2.1), followed by a discussion of the Parliament configuration file (Section 2.2). Then comes a detailed discussion of each of the less common deployment scenarios. The chapter concludes with discussions of Parliament utilities and troubleshooting (Sections 2.6 and 2.7).

2.1 Deploying a Parliament Server

A Parliament distribution is a compressed archive containing a ready-to-run Parliament Server, with the following naming convention:

[1] http://parliament.semwebcentral.org/

```
Parliament-vX.Y.Z-toolset.zip
```

Here ***X.Y.Z*** is the Parliament version number and ***toolset*** indicates the platform on which the distribution runs, as shown in Table 2.1. If you are

Toolset	*Corresponding Platform*
`msvc141-64`	64-bit Windows, built with Visual Studio 2017
`msvc141-32`	32-bit Windows, built with Visual Studio 2017
`clang`	Universal binaries for Mac OS X
`gcc-64`	64-bit Ubuntu Linux
`gcc-32`	32-bit Ubuntu Linux

Table 2.1: Toolset-to-Platform Correspondence

building Parliament yourself according to the instructions in Chapter 3, then these archives appear in the `target/distro` directory at the end of the build process.

As shown in Table 2.1, most platforms require that you choose between 32-bit and 64-bit builds. Whenever possible, use the 64-bit version, because 32-bit Parliament is limited to 5 to 10 million statements.

2.1.1 Installing for the First Time

Once you have downloaded a distribution, these steps will yield a functioning Knowledge Base (KB):

1. Extract the archive contents to whatever location you choose. On UNIX-like systems, `/usr/opt/ParliamentKB` is a likely choice. On Windows, `C:\Program Files\ParliamentKB` is customary. The instructions that follow refer to this directory as `ParliamentKB`.

2. On Windows, install the C and C++ run-time libraries. You can find installers for these in the following directory:

   ```
   ParliamentKB/RedistributablePackages
   ```

3. From the `ParliamentKB` directory, invoke the startup script to start the Parliament server. This is called `StartParliament.bat` on Win-

dows and `StartParliament.sh` on other platforms.[2]

2.1.2 Upgrading an Existing Installation

When upgrading an existing Parliament installation to a newer version, the procedures of Section 2.1.1 are generally not sufficient because they ignore the migration of the data. The Parliament development team strives to maintain compatibility of file formats whenever possible. In such cases, all that is required is to move the `data` subdirectory of your Parliament installation into the new Parliament installation. (Of course, if you have customized Parliament's configuration, you will also need to make those changes in the new installation as well.)

However, from time to time Parliament's file formats do change, and in such cases a different migration procedure is required. Following this procedure even in those cases where it is not required has some benefits as well, since it frees up any unused space in the Parliament files. This is the way to perform such a migration:

1. Before upgrading, point your browser at your Parliament server. The Uniform Resource Locator (URL) for this looks like `http://<host>:<port>/parliament/`. By default this will be `http://localhost:8089/parliament/`.
2. Click on the "Export" link.
3. Click on the "Export Repository" button. This will start a download whose file name looks like this:

   `parliament-export-localhost-<date>-<time>.zip`

4. When the download finishes, shut down Parliament. Rename the Parliament installation directory if you want your upgraded installation to reside in the same place.
5. Unzip the new version of Parliament and move the resulting installation directory to your desired location.

[2] On all platforms, these scripts locate your Java installation by simply invoking the command `java` and relying on the operating system's native facilities for locating the Java Virtual Machine (JVM) executable. See Section 2.7 for more information.

6. Customize the new installation's settings as required. The most common customizations are the host and port, which are easily modified in the startup script, and the settings in the ParliamentConfig.txt file.
7. Start up the new Parliament installation.
8. Point your browser at the new Parliament server. As above, the URL for this looks like `http://<host>:<port>/parliament/`, or `http://localhost:8089/parliament/` by default.
9. Click on the "Insert Data" link.
10. In the "Import Repository" section, press the "Choose File" button and select the file you downloaded in Step 3 above. Then press the "Import Repository" button.

Once you see the "Insert operation successful" message, your upgraded Parliament server is ready for business.

2.1.3 Usage

Important note: When you shut down Parliament, it is important to shut it down gracefully. Otherwise, the Parliament files may not be flushed to disk before they are closed, and they may be badly corrupted. If you run Parliament as a Windows service or UNIX Daemon (see Section 2.1.4), this is generally not a problem, because the operating system sends a shut-down message at the appropriate times. If, however, you run Parliament explicitly via `StartParliament.sh` (or `StartParliament.bat` on Windows), you will need to shut it down yourself by typing the command "exit" followed by «return» or «enter».

To use the Parliament server, you will need the the connection URLs listed in Table 2.2. (See Section 2.1.4 to configure your server to run on a host name and port other than "localhost" and "8089".) You can use the Parlia-

HTTP interface:	`http://localhost:8089/parliament`
SPARQL endpoint:	`http://localhost:8089/parliament/sparql`
Bulk loader:	`http://localhost:8089/parliament/bulk`

Table 2.2: Parliament Server Connection URLs

ment server interactively by pointing your web browser at the HyperText

Transfer Protocol (HTTP) interface URL from Table 2.2. This simple and (hopefully) self-explanatory web interface is useful for initial data loading, experimenting with queries, exploring a collection of data, troubleshooting, and similar tasks.

In order to issue queries programmatically, you will need to write some code to send SPARQL-compliant HTTP requests. One library that can send such requests on your behalf is Jena itself. Figure 2.1 shows sample code for issuing a SPARQL select query. Note that this code uses the SPARQL

```
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.ResultSet;

void issueSelectQuery(String sparqlSelectQuery)
{
  QueryExecution exec = QueryExecutionFactory.sparqlService(
    "http://localhost:8089/parliament/sparql",
    sparqlSelectQuery);
  for (ResultSet rs = exec.execSelect(); rs.hasNext(); rs.next())
  {
    // Do something useful with the result set here
  }
}
```

Figure 2.1: Issuing a SPARQL select query with Jena

endpoint URL from Table 2.2.

Parliament also provides a Java library to help programmers interact with the Parliament server, which can be found here:

`ParliamentKB/lib/JosekiParliamentClient.jar`

This library leverages the Jena functionality shown above, but exposes a few extra features of the Parliament server that are not accessible within the bounds of SPARQL. To use this approach, first add the above jar file to your class path. Then create a `RemoteModel` object as shown in Figure 2.2. This object allows you to access and manipulate the Parliament repository identified by the URLs passed to the constructor, which are again taken from Table 2.2. Note that the query strings passed to the remote model in Figure 2.2 follow standard SPARQL and SPARQL-UPDATE formats, including support for named graphs. Also note that the `RemoteModel` class

```
import com.bbn.parliament.jena.joseki.client.RemoteModel;

void useRemoteParliamentModel()
{
  // Create model:
  RemoteModel rmParliament = new RemoteModel(
    "http://localhost:8089/parliament/sparql",
    "http://localhost:8089/parliament/bulk");

  // Insert contents of another Jena Model:
  rmParliament.insertStatements(myPopulatedJenaModel);

  // Perform SPARQL SELECT query:
  ResultSet results = rmParliament.selectQuery(myQueryString);

  // Add statements using SPARQL-UPDATE:
  rmParliament.updateQuery(mySparqlUpdateQueryString);
}
```

Figure 2.2: Using Parliament Via a Remote Jena Model

exposes the bulk loading feature of the Parliament server, allowing the loading of large bodies of RDF efficiently.

2.1.4 Configuration

This section explores the most important configuration options for your Parliament server. The installation procedure in Section 2.1.1 simply accepts the defaults for these options, but most deployments will require some level of customization.

Section 2.1.1 mentions Parliament's start script, which is used to directly start the server. However, in many cases it is preferable to run Parliament as a Windows service or as a UNIX daemon.

The script `InstallParliamentService.bat` installs Parliament as a Windows service. In order for this script to have the permissions it requires, you usually need to right-click the Command Prompt shortcut and choose "Run as Administrator." This script locates your Java installation by using the `JAVA_HOME` environment variable.[3] After running this script, use

[3]See Section 2.7 for information about choosing among multiple Java installations.

the Services management console to start and stop Parliament and to run Parliament automatically upon system startup. Uninstall the service with `UninstallParliamentService.bat`.

The `StartParliamentDaemon.sh` script will start Parliament as a daemon on UNIX-like systems. The exact method for wiring this into the operating system's infrastructure for starting and stopping daemon processes varies from platform to platform, and is not covered here.

You can change the host name (or IP address) and port of the SPARQL endpoint by setting `JETTY_HOST` and `JETTY_PORT` in the start script. These default to `localhost` and 8089, respectively. Also, you can control the amount of memory set aside for the Java virtual machine's heap by setting `MIN_MEM` and `MAX_MEM` in the same location. While it is important to allow the JVM sufficient memory, it is also important to realize that a significant portion of Parliament is native code, and therefore does not use the Java heap. Java programmers are often inclined to set `MAX_MEM` close to the total memory of the machine, but this will starve Parliament's native code of the memory it needs to achieve good performance. The default heap size ranges from 128 MB to 512 MB, which is reasonable for machines with 4 to 8 GB of total memory.

Note that if you are running the server as a Windows service, you must change these parameters in the install script, and you will have to uninstall the service and reinstall it to make your changes effective.

The file `ParliamentKB/conf/jetty.xml` configures the Jetty servlet container. Generally, there is no need to modify this file. The Parliament configuration file (`ParliamentConfig.txt`) is discussed in Section 2.2.

2.2 The Parliament Configuration File

Like many pieces of software, Parliament has a configuration file. It is typically called `ParliamentConfig.txt`, but it can be named anything. Here is how Parliament finds its configuration:

1. If the environment variable `PARLIAMENT_CONFIG_PATH` is set, then its value is assumed to be the absolute path of the configuration file. The Parliament server startup scripts use this method to locate the file in

the root directory of the server installation.

2. *On Windows only:* If this environment variable is not set, then Parliament looks for the file `ParliamentConfig.txt` in the same directory as the Parliament Dynamic Link Library (DLL).[4]
3. Otherwise, Parliament looks for the file `ParliamentConfig.txt` in the current working directory of the server process.

The configuration file is a plain text file containing name-value pairs. Blank lines and comment lines (preceded by a '#') are ignored. Many of the settings are Boolean values, in which case the value is case-insensitive and may be "true", "t", "yes", "y", "on", or "1" (for true), or "false", "f", "no", "n", "off", or "0" (for false).

The recognized settings are as follows. Setting names are case-insensitive.

logToConsole Indicates whether to log to the console. *Default: "yes"*

logConsoleAsynchronous Indicates whether to log asynchronously to the console. *Default: "no"*

logConsoleAutoFlush Indicates whether console log entries should be flushed after each log operation. *Default: "yes"*

logToFile Indicates whether to log to a file. *Default: "yes"*

logFilePath The path of the log file. May include the following placeholders to add contextual information to the file name:

- **`%N`** Cardinal number of the log file. A number between the % and the N will left-pad with zeros to that many places.
- **`%Y`** Year
- **`%m`** Month
- **`%d`** Day
- **`%H`** Hours
- **`%M`** Minutes
- **`%S`** Seconds

Default: `log/ParliamentNative%3N_%Y-%m-%d_%H-%M-%S.log`

[4] Or "shared library" on UNIX-like systems. For brevity, we will just use "DLL."

logFileAsynchronous Indicates whether to log asynchronously to the file. *Default: "no"*

logFileAutoFlush Indicates whether file log entries should be flushed after each log operation. *Default: "yes"*

logFileRotationSize The maximum approximate size (in bytes) of the log file. When this value is exceeded, the log file will be rotated. *Default: 10 MB*

logFileMaxAccumSize The maximum accumulated size of rotated log files. When this value is exceeded, the oldest log file is deleted. *Default: 150 MB*

logFileMinFreeSpace The minimum amount of free space on the volume where log files are stored. When the free space dips below this value, old log files will be deleted so as to free up enough space. *Default: 100 MB*

logFileRotationTimePoint The time of day at which the log file should be rotated regardless of its size. Must be in the format "HH:MM:SS". *Default: 2:00 AM*

logLevel The global logging level. Must be one of TRACE, DEBUG, INFO, WARN, and ERROR. *Default: "INFO"*

logChannelLevel Overrides the global logging level for a single channel. (Generally a channel is equivalent to a class in the source code.) The value of this setting takes the form

«channel-name» = «log-level»

where *«log-level»* takes one of the values listed for the global logging level above. This option may be repeated to override the level for multiple channels. *Default: None*

kbDirectoryPath The path of the Parliament knowledge base files. If they do not yet exist, they will be created here. *Default: The current working directory, "."*

stmtFileName The name of the memory-mapped file containing records of statements. *Default:* `statements.mem`

rsrcFileName The name of the memory-mapped file that contains resource records. *Default:* `resources.mem`

uriTableFileName The name of the memory-mapped file containing resource strings (URIs and literals). *Default:* `uris.mem`

uriToIntFileName The name of the file containing the mapping from numeric resource identifiers to resource strings. This file is managed by Berkeley DB. *Default:* `u2i.db`

stmtToIdFileName This obsolete setting is now ignored. It indicated the name of a file that mapped tuples of numeric resource identifiers to numeric statement identifiers. This file was omitted by default, and was included only if the setting "keepDupStmtIdx" was turned on. The default value was `stmt2id.db`. If your configuration had "keepDupStmtIdx" turned off, then you can safely delete these two settings from your configuration file. If it was turned on, then delete the file named by this setting and then delete these two settings from your configuration file.

keepDupStmtIdx This obsolete setting is now ignored. If set to "yes", Parliament would maintain an extra file (named by the "stmtToIdFileName" option). It was intended to be a performance optimization, but it rarely helped. It defaulted to "no". If you had this turned off then you can simply delete this and the "stmtToIdFileName" options from your configuration. If it was turned on, then delete the file named by the "stmtToIdFileName" option and then delete these two settings from your configuration file.

readOnly When set to "yes", prevents Parliament from changing the underlying storage files in any way. *Default: "no"*

fileSyncTimerDelay The number of milliseconds between flushings of the KB files to disk. The flush is performed asynchronously, and so has minimal impact on overall performance. This decreases the chances of a file corruption, and it limits the amount of time required to shut down Parliament gracefully. Set to zero to disable flushing the files to disk. This setting applies only to Parliament deployed as a server using Jena, Joseki, and Jetty. Any other deployment will ignore this setting. *Default: "15000"*

initialRsrcCapacity The number of resources Parliament should allocate space for when creating a new KB. *Default: "100000"*

avgRsrcLen The average resource length (in characters) that Parliament should anticipate when allocating space in a new KB. *Default: "64"*

rsrcGrowthFactor The factor by which to increase the resource table size when Parliament runs out of space in the file. This must be larger than one. *Default: "1.25"*

initialStmtCapacity The number of statements Parliament should allocate space for when creating a new KB. *Default: "500000"*

stmtGrowthFactor The factor by which to increase the statement table size when Parliament runs out of space in the file. This must be larger than one. *Default: "1.25"*

bdbCacheSize The amount of memory to be devoted to the Berkeley DB cache. The portion before the comma is the total cache size (with a k for kilobytes, m for megabytes, g for gigabytes). The portion after the comma specifies how many segments the memory should be broken across, for compatibility with systems that limit the size of single memory allocations. On systems with 4 GB or more of memory, setting this to "256m,1" generally seems to be optimal. *Default: "32m,1"*

runAllRulesAtStartup Causes Parliament to evaluate each of the enabled rules at startup against the existing statements in the KB to see if any new statements should be materialized. Generally, this is unnecessary, because the rules are always evaluated against each statement as it is asserted. However, if some of the rules were initially disabled when statements were being asserted, and are then enabled, it may be necessary to turn this on for one startup of Parliament to ensure that the state of the KB is in sync with the rule settings. Alternately, in such a case this setting may be left off and the ParliamentAdmin tool may be run against the KB with the "guaranteeEntailments" option. Note that when this setting is turned on, KB the startup time may be lengthy. *Default: "no"*

normalizeTypedStringLiterals Instructs Parliament to convert typed string literals to plain literals, both upon insert and at query time, as mandated in RDF 1.1. In Parliament versions before the implemen-

tation of this option (versions 2.7.9 and prior), the behavior was as if the option were set to "no". Therefore, if you are migrating a Parliament installation originally created with one of these older versions to a newer version, you must do one of two things:

1. Either before or after the upgrade, create an export of the entire triple store. To do this, use the "Export Repository" option on the Export page of Parliament's Web-based administrative interface. Then shut down Parliament and delete its `data` directory. This is the location identified by the `kbDirectoryPath` setting in the configuration file. Then, after upgrading, import the backup using the "Import Repository" option on the Insert Data page of the administrative interface.
2. Set this option to "no".

The first option is highly recommended. It requires some effort, but will bring your triple store into compliance with RDF 1.1 and deliver slightly better performance to boot. *Default: "yes"*

SubclassRule Enables the following inference rules:

$$A \subset B \wedge B \subset C \Rightarrow A \subset C$$

$$X \in A \wedge A \subset B \Rightarrow X \in B$$

where $\subset$ means "is a sub-class of" and $\in$ means "is of type". *Default: "on"*

inferRdfsClass When both this setting and "SubclassRule" are turned on, this enables the following inference rule:

$$A \subset B \Rightarrow A \in \text{rdfs:Class} \wedge B \in \text{rdfs:Class}$$

where $\subset$ means "is a sub-class of" and $\in$ means "is of type". *Default: "off"*

inferOwlClass When both this setting and "SubclassRule" are turned on, this enables the following inference rule:

$$A \subset B \Rightarrow A \in \text{owl:Class} \wedge B \in \text{owl:Class}$$

where $\subset$ means "is a sub-class of" and $\in$ means "is of type". *Default: "off"*

inferRdfsResource When both this and the "SubclassRule" settings are turned on, the following inference rule is enabled:

$$A \subset B \Rightarrow A \subset \text{rdfs:Resource} \wedge B \subset \text{rdfs:Resource}$$

where $\subset$ means "is a sub-class of". *Default: "off"*

inferOwlThing When both this setting and "SubclassRule" are turned on, this enables the following inference rule:

$$A \subset B \Rightarrow A \subset \text{owl:Thing} \wedge B \subset \text{owl:Thing}$$

where $\subset$ means "is a sub-class of". *Default: "off"*

SubpropertyRule Enables the following inference rules:

$$P \sqsubset Q \wedge Q \sqsubset R \Rightarrow P \sqsubset R$$

$$P \sqsubset Q \wedge P(X,Y) \Rightarrow Q(X,Y)$$

where $\sqsubset$ means "is a sub-property of". *Default: "on"*

DomainRule Enables the following inference rule:

$$\text{rdfs:domain}(P,C) \wedge P(X,Y) \Rightarrow X \in C$$

Default: "on"

RangeRule Enables the following inference rule:

$$\text{rdfs:range}(P,C) \wedge P(X,Y) \Rightarrow Y \in C$$

Default: "on"

EquivalentClassRule Enables the following inference rule:

$$\text{owl:equivalentClass}(A,B) \Rightarrow A \subset B \wedge B \subset A$$

where $\subset$ means "is a sub-class of". *Default: "on"*

EquivalentPropRule Enables the following inference rule:

$$\text{owl:equivalentProperty}(P,Q) \Rightarrow P \sqsubset Q \wedge Q \sqsubset P$$

where $\sqsubset$ means "is a sub-property of". *Default: "on"*

InverseOfRule Enables the following inference rule:

$$\text{owl:inverseOf}(P, Q) \wedge P(X, Y) \Rightarrow Q(Y, X)$$

Default: "on"

SymmetricPropRule Enables the following inference rule:

$$P \in \text{owl:SymmetricProperty} \wedge P(X, Y) \Rightarrow P(Y, X)$$

Default: "on"

FunctionalPropRule Enables the following inference rule:

$$P \in \text{owl:FP} \wedge P(Z, X) \wedge P(Z, Y) \Rightarrow \text{owl:sameAs}(X, Y)$$

where "FP" stands for "FunctionalProperty". *Default: "on"*

InvFunctionalPropRule Enables the following inference rule:

$$P \in \text{owl:IFP} \wedge P(X, Z) \wedge P(Y, Z) \Rightarrow \text{owl:sameAs}(X, Y)$$

where "IFP" stands for "InverseFunctionalProperty". *Default: "on"*

TransitivePropRule Enables the following inference rule:

$$P \in \text{owl:TransitiveProperty} \wedge P(X, Y) \wedge P(Y, Z) \Rightarrow P(X, Z)$$

Default: "on"

2.3 Parliament in Other Servlet Containers

The most common way to deploy Parliament uses the Jetty servlet container. This scenario is discussed above in Section 2.1. However, the Parliament distribution contains a war file that can be deployed in any servlet container. This section discusses how to deploy Parliament in other servlet containers, such as Apache Tomcat.

1. First acquire and install Apache Tomcat[5] according to its own installation instructions. This will result in a directory containing your Tomcat installation, which, in keeping with the customary nomenclature of Tomcat, we will call `CATALINA_HOME` here.

[5] http://tomcat.apache.org/

2. Create a directory for the native Parliament binaries. This can reside anywhere on your machine, but just to be concrete we will call this directory `CATALINA_HOME/bin/parliament`.
3. Copy the native binaries into this directory from the `bin` directory of the Parliament binary distribution.
4. On Windows only, install the run-time libraries by running the installer in the `RedistributablePackages` subdirectory of your Parliament distribution.
5. Once you have chosen your build directory, copy all of the files from it into the `CATALINA_HOME/bin/parliament` directory.
6. In your favorite text editor, open the file

   ```
   CATALINA_HOME/bin/parliament/ParliamentConfig.txt
   ```

   and change the following line:

   ```
   kbDirectoryPath = .
   ```

   to point to a directory suitable for storing your knowledge base. In an operational deployment of Parliament, this is often set to a large drive dedicated to the storage of the knowledge base, such as a Redundant Array of Inexpensive Disks (RAID), Network-Attached Storage (NAS), or Storage Area Network (SAN). For testing purposes, set it to a relative path, such as:

   ```
   kbDirectoryPath = data
   ```

   This stores the knowledge base in `CATALINA_HOME/bin/data`.
7. If you have not already done so while installing Tomcat, create a file called `setenv.sh` in `CATALINA_HOME/bin` and within it set the environment variable `CATALINA_OPTS` like so:

```
MIN_MEM=128m
MAX_MEM=512m
PARLIAMENT_BIN="$CATALINA_HOME/bin/parliament"
export CATALINA_OPTS=-Djava.library.path=$PARLIAMENT_BIN
export CATALINA_OPTS="$CATALINA_OPTS -Xms$MIN_MEM -Xmx$MAX_MEM"
export PARLIAMENT_CONFIG_PATH=$PARLIAMENT_BIN/ParliamentConfig.txt
export LD_LIBRARY_PATH=$PARLIAMENT_BIN
```

On Windows, the file name should be `setenv.bat` and its content should look like this:

```
set MIN_MEM=128m
set MAX_MEM=512m
set PARLIAMENT_BIN=%CATALINA_HOME%\bin\parliament
set CATALINA_OPTS=-Xms%MIN_MEM% -Xmx%MAX_MEM%
set PARLIAMENT_CONFIG_PATH=%PARLIAMENT_BIN%\ParliamentConfig.txt
set PATH=%PARLIAMENT_BIN%;%PATH%
```

Note that you can control the amount of memory set aside for the Java virtual machine by changing `MIN_MEM` and `MAX_MEM` in the scripts above. While it is important to allow the JVM sufficient memory, it is also important to realize that Parliament is native code, and therefore does not use the Java heap. Java programmers are often inclined to set `MAX_MEM` close to the total memory of the machine, but this will starve Parliament of the memory it needs to achieve good performance. The default values given above, 128 MB and 512 MB, are reasonable values for machines with 4 to 8 GB of total memory.

8. Copy `parliament.war` from the Parliament distribution into Tomcat's appbase directory, which defaults to `CATALINA_HOME/webapps`.
9. Start Tomcat according to its documentation. For instructions on using your Parliament server, see Section 2.1.3.
10. If you redeploy Parliament into a running Tomcat instance, the server will need to be restarted.
11. *Please note:* When you want to shut down the KB, it is important to shut it down gracefully. Otherwise, the Parliament files may not be flushed to disk before they are closed, and they may be badly corrupted. If you run Tomcat as a Windows service or UNIX Daemon, this is generally not a problem, because the operating system sends a shut-down message at the appropriate time. If, however, you run Tomcat explicitly via its startup script, you will need to shut it down yourself with Tomcat's stop script.

Glassfish Notes

When deploying Parliament to Glassfish, note the following:

- Glassfish sets it's own `java.library.path` property. The Parliament libraries must either be copied into «`Glassfish install dir`»`/lib` or you must alter the Glassfish configuration such that the libraries are available on `java.library.path`.
- By default, Parliament stores it's data files in the current working directory. In the case of Glassfish, this will be «`Glassfish install dir`»`/domains/domain1/config`. To change this, edit the configuration file, `ParliamentConfig.txt`, to set `kbDirectoryPath` to the location (absolute path) where you would like your files to reside.
- If Parliament is redeployed into a running instance of Glassfish, the server will need to be restarted.

2.4 Using Parliament In-Process

Jena is a popular toolkit for manipulating RDF data in Java programs. The central concept in the Jena class library is the Model interface. An object of type Model represents a graph of RDF data, and features many methods for manipulating that data. With just a few lines of code, you can create a Jena Model that is backed by an instance of Parliament, as demonstrated by Figure 2.3. The `createParliamentModel` method returns a Jena model

```
import java.io.File;
import com.bbn.parliament.jena.graph.KbGraph;
import com.bbn.parliament.jena.graph.KbGraphFactory;
import com.bbn.parliament.jni.Config;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;

private Model createParliamentModel()
{
  KbGraph baseGraph = KbGraphFactory.createDefaultGraph();
  return ModelFactory.createModelForGraph(baseGraph);
}
```

Figure 2.3: Creating a Jena Model backed by Parliament

backed by a Parliament instance. With this model, you can do any of the things you normally do with a Jena model, such as call `read` on it to load a file into it, or query it.

The `useParliamentModel` method shown in Figure 2.4 demonstrates how to call `createParliamentModel`. The important thing to note here is the try-finally construct. This guarantees that the model is closed properly, even if an exception is thrown. Closing the model is crucial, because if you don't, the Parliament files will not be flushed to disk before they are closed, and they will almost certainly be badly corrupted.

```
import com.hp.hpl.jena.rdf.model.Model;

void useParliamentModel()
{
  Model kbModel = createParliamentModel();
  try
  {
    // Use the model according to the Jena documentation.
    // For instance, to issue a SPARQL query:
    QueryExecution exec = QueryExecutionFactory.create(
      sparqlQueryString, kbModel);
    for (ResultSet rs = exec.execSelect(); rs.hasNext(); rs.next())
    {
      // Do something useful with the result set here
    }
  }
  finally
  {
    if (kbModel != null && !kbModel.isClosed())
    {
      kbModel.close();
      kbModel = null;
    }
  }
}
```

Figure 2.4: Using a Jena Model backed by Parliament

Naturally, there is some setup required to make this work:

1. You will need `JenaParliament.jar` and `Parliament.jar` from your binary distribution, along with the Jena jar files, which you can download from the Jena web site.[6] Place all of these jar files in a convenient location and ensure that they are added to your class path. (If you are using Eclipse, add them to the build path.)

[6] https://jena.apache.org/

2. Copy the following files from the `bin` directory of the Parliament distribution to a convenient location, such as the directory used for the jar files above:
   - All of the DLLs or shared libraries.
   - `ParliamentAdmin` (`ParliamentAdmin.exe` on Windows)
   - `ParliamentConfig.txt`
3. Ensure that the JVM can find the DLLs or shared libraries at run-time. For Mac OS X, add their directory to the Java system property `java.library.path`. On Windows, add the directory to the `Path` environment variable. On Linux, it's `LD_LIBRARY_PATH`. Many flavors of UNIX also use `LD_LIBRARY_PATH`, though some differ. Consult your operating system documentation to be sure you use the correct variable.
4. On Windows only, install the run-time libraries by running the installer in the `RedistributablePackages` subdirectory of your Parliament distribution.
5. Customize the configuration file `ParliamentConfig.txt` as desired. In particular, the `kbDirectoryPath` setting defaults to the current working directory. To place your Parliament KB files in a different location, change this setting to the directory of your choice. This is especially useful for placing the KB files on a drive array, or for maintaining several KBs and switching between them.
6. Set the `PARLIAMENT_CONFIG_PATH` environment variable to the configuration file's path. On Windows, if this is not set the file will be loaded from the same directory as the Parliament DLL, which is often a useful default. On other platforms there is no default, and so this environment variable must be set.

If you do not wish to use the Parliament configuration file to configure your application, you can replace the first line of the method in Figure 2.3, `createParliamentModel`, with a line that creates a Config instance using the default constructor and then set the Config fields yourself. This may be useful for centralizing your application's configuration in a single file.

If `createParliamentModel` throws an `UnsatisfiedLinkError` exception, see Section 2.7 for possible resolutions.

2.5 Configuring Indexes

[Yet to be written]

2.6 The ParliamentAdmin Utility

[Yet to be written]

2.7 Troubleshooting

Because Parliament is most often used within a Java environment, the most common problem is the dreaded "Unsatisfied Link Error". This error is generated by the JVM when the Java code requests the loading of a DLL, but the JVM is unable to load that DLL. There are quite a few reasons why you may encounter an unsatisfied link error, and unfortunately, the JVM is not very good about generating an illuminating error message. So, should you encounter this problem, here are some things to keep in mind:

- Double-check that you have followed all the deployment instructions correctly for your chosen mode of deployment.
- Make sure that your system is up-to-date with respect to patches. For Windows, this means a visit to the Windows Update web site.[7] Most other operating systems have a similar facility.
- Be sure that a 64-bit build of Parliament is loaded by a 64-bit version of Java, and that a 32-bit build of Parliament is loaded by a 32-bit version of Java. On Macintosh, this is not a problem, because Parliament is built as a universal binary.
- When the JVM loads the Parliament DLL, Parliament itself loads several other DLLs. Even if the JVM is able to load Parliament, an unsatisfied link error will result if Parliament cannot load one of its subsidiary DLLs. Furthermore, the error message accompanying the unsatisfied link error usually does not indicate which DLL caused the load failure. The subsidiary DLLs fall into the following three categories:

[7] http://windowsupdate.microsoft.com/

  - The Berkeley DB and Boost DLLs: In the deployment procedures detailed in this document, these DLLs should reside in the same location as Parliament itself.
  - The C and C++ run-time libraries: On Unix-like systems these should be available automatically. On Windows you may need to install the Visual Studio run-time libraries. You can find installers for these in the `RedistributablePackages` subdirectory of your binary distribution of Parliament.
  - Various operating system DLLs: These are rarely a problem, because the OS needs to be able to find these to function.

- The Java system property `java.library.path` is a list of paths that Java searches when loading native libraries. This suggests that all of the operating system-specific rules for locating DLLs can be circumvented by providing a suitable definition for this system property. Unfortunately, when Java loads a library that itself loads other libraries (as Parliament does), Java can consult `java.library.path` only when loading the top-level library, because the loading process for its dependents is managed entirely within the operating system. Thus, this property does not solve the problem for Parliament.
- Running `ParliamentAdmin` helps to diagnose unsatisfied link errors, because this tool also loads the Parliament DLL, but it does so without involving Java (because `ParliamentAdmin` is written in C++). This technique can be used to isolate the error to either the operating system level (when `ParliamentAdmin` fails to load the DLL) or to the Java level (when `ParliamentAdmin` succeeds). Even running "`ParliamentAdmin -v`" to get the version requires loading the Parliament DLL, so this is an easy test.
- Multiple Java installations can confuse the DLL loading process. The scripts `StartParliament.sh` and `StartParliament.bat` find Java by simply invoking the command `java` and relying on the operating system to locate the installation. In these cases, the `JAVA_HOME` environment variable is not used, unless the operating system uses it. On Windows, the process by which the `java` command finds and loads the JVM is quite complex. The following web page sheds some light on how you can manage this process:

http://mindprod.com/jgloss/javaexe.html#MULTIPLES

In the case of the `InstallParliamentService` script, the situation is quite different. The `java` command is not used at all, and instead the service executable directly loads the `jvm.dll` library. This is located by using the `JAVA_HOME` environment variable at the time the service is installed. (This means that if you upgrade your Java installation after installing the service, you will have to uninstall and reinstall the service in order for it to find the `jvm.dll` library.)

- The process by which Windows searches for DLLs is somewhat complex, so understanding it can often help pinpoint the problem. The search process is as follows:
  - The Windows system directory
  - The Windows directory
  - The directory where the executable module for the current process is located
  - The current directory
  - The directories listed in the `PATH` environment variable

  Note that placing DLLs in the Windows or Windows system directories is strongly discouraged.

  Both the graphical tool Dependency Walker[8] and the command-line tool `dumpbin` (part of the Visual Studio installation) show from where Windows is trying to load subsidiary DLLs. (Note that some operating system DLLs themselves load many other libraries, some by magic. This can confuse Dependency Walker, causing it to report missing DLLs. These error messages are, however, red herrings and should be ignored.)

- On Mac OS X, the JVM finds the Parliament shared library by consulting the Java system property `java.library.path`, and the operating system looks for its dependencies in the same directory. The command "`otool -L`" will show from where the operating system is trying to load subsidiary shared libraries.

[8] http://dependencywalker.com/

- Other UNIX-like systems find shared libraries by searching the directories in the library search path environment variable. The name of this variable differs by system: On Linux and several others it is `LD_LIBRARY_PATH`, though some differ. On most UNIX-like systems the command `ldd` will show from where the operating system is trying to load subsidiary shared libraries.
- A debug build of Parliament will not work on Windows unless the corresponding version of Visual Studio is installed. This is because the debug versions of the C and C++ run-time libraries are included only with Visual Studio.
- More rarely, unsatisfied link errors can result from a bad build of Parliament that causes the symbols exported from the DLL to be named incorrectly. If this happen, the JVM will succeed in loading the DLL, but fail to find the entry points it needs. This results in an unsatisfied link error that is indistinguishable from the cases where the JVM cannot load the DLL at all. This condition is more likely to occur on Windows, and usually has something to do with the symbol `BUILDING_KBCORE` not being defined on the compiler command line.

Chapter 3

Building Parliament™

Parliament is a cross-platform, mixed-language library. It's core is written in portable C++, but it also has a Java interface. As a result of both the cross-platform and multi-language requirements, the build infrastructure for Parliament requires a little bit of work to configure. This chapter is your guide through that process.

Parliament's build infrastructure has two main parts. The top-level portion is based on ant, a build tool widely used in the Java development community. This portion of the infrastructure builds the Java half of the Parliament code base, and it also invokes the second portion, which is based on Boost.Build. Boost.Build is a system that is well-adapted to building C++ code. It has the advantages of being portable and much simpler to use than make files. It is also the standard build system of the Boost project, whose libraries are used by the C++ portion of Parliament.

This chapter will step through the libraries and tools that Parliament depends upon and show you how to configure them on your system. At the end of this chapter, you should have a working copy of the Parliament source code from which you can build Parliament binaries.

3.1 Platforms and Prerequisites

You will need to acquire and install one or more appropriate compilers for each operating system on which you wish to build. Parliament has been

tested on the platform and compiler combinations shown in Table 3.1. Note that the last column shows the corresponding Boost.Build toolset name, which will be used extensively in the sections below as we configure the Parliament build infrastructure.

Operating System	**Compiler**	**Toolset**
Windows (32- and 64-bit)	Visual Studio 2017	msvc-14.1
Mac OS X 10.10.3	Xcode 6.3.1	clang
Ubuntu 15.10 (64-bit)	GCC 5.2.1	gcc
Centos 6.6 (64-bit)	GCC 4.4.7	gcc

Table 3.1: Supported Platforms and Compilers

Windows versions 7 and above are supported. The 64-bit build has been tested on AMD-64 or EM64T hardware (often collectively known as x64). Parliament's capacity is much higher when running as a 64-bit process,[1] so one of those compilers is highly recommended.

On Macintosh, Parliament builds as a universal binary (for the x64 and x86 architectures) using Apple's Xcode development tools.

Parliament assumes the presence of the Java Developer Kit (JDK), version 8 or above. Furthermore, you will need a 64-bit JVM in order to run a 64-bit build of Parliament.

- On Windows, download and install the JDK from Oracle. The 32-bit and 64-bit versions are separate downloads and installations.
- On Macintosh, download and install the JDK from Oracle.
- On Linux, you may need to install one or more packages, depending on your particular distribution.

You will need Apache Ant version 1.10.1 or later[2] and Apache Ivy version 2.4.0 or later.[3] Install these according to their documentation. Finally, you

[1] On 32-bit Windows Parliament runs out of virtual address space after storing 5 to 10 million statements.

[2] `http://ant.apache.org/`

[3] `http://ant.apache.org/ivy`

need a client for the git version control system.[4]

Once you have these prerequisites installed, you can clone the Parliament code base from here:

```
https://github.com/SemWebCentral/parliament
```

3.2 Configuring Eclipse

The Parliament code base includes Eclipse projects for all of the Java and C++ code. These are useful for inspecting and editing code, but it is important to note that they are not the official build mechanism. In fact, as of this writing, the C++ Eclipse projects do not build at all. (The Java projects do build correctly, but they are still not the official build mechanism.) This may be corrected in the future, but the Java Native Interface (JNI) layer between the Java and native code makes this complex. Therefore the C++ projects are merely an editing convenience.

To setup Eclipse, you need the Mars version or later of the Eclipse Integrated Development Environment (IDE) for Java Developers, plus the Eclipse C/C++ Developers Toolkit (CDT). One way to acquire this set of components is to download the Eclipse IDE for Java,[5] install it, and use its "Install New Software" menu item to download and install the CDT. The procedure for this changes a bit between Eclipse releases, but you can find instructions on the CDT web site.[6]

To use Eclipse with your Parliament working copy, first choose (or create) an Eclipse workspace. Then import all existing projects from within your Parliament working copy. To do so, choose Import from the File menu and select "Existing Projects into Workspace" under the General category. Press the Next button, and enter the root directory of your Parliament working copy in the "Select root directory" box. Press the Select All button, make sure that "Copy projects into workspace" is unchecked, and press the Finish button. At this point all of the Parliament projects will be displayed in the Package Explorer view.

[4] https://git-scm.com/downloads
[5] http://www.eclipse.org/
[6] http://www.eclipse.org/cdt/

3.3 Building Berkeley DB

Parliament uses Berkeley DB (often abbreviated BDB), an embedded database manager from Oracle.[7] Because Parliament is open source, this use of Berkeley DB also falls under an open source license. The following procedures are based on BDB version 5.3.28.

Note that Parliament versions 2.7.6 through 2.7.9 was released with Berkeley DB 6.1, but we have rolled back to 5.3 because Oracle sneakily changed their open source license from a BSD-like license to AGPL, which is too restrictive for our purposes.

3.3.1 Building BDB for Windows

The build infrastructure for Berkeley DB is not particularly friendly for Windows. Therefore, the Parliament source code repository includes pre-built Berkeley DB libraries (both 32- and 64-bit) for all supported versions of Visual Studio.[8] If you need to update the pre-built libraries, e.g., for a new version of Berkeley DB or to build with a different compiler, see Appendix A.

Define the following environment variables so that the Parliament build infrastructure can find the libraries:

```
BDB_VERSION=53
BDB_HOME=«dir»\lib\bdb
```

where «`dir`» is the absolute path of your Parliament working copy.

3.3.2 Building BDB for Macintosh

On Macintosh, Berkeley DB follows the usual pattern of software based on the autoconf/automake/libtool suite. Specifically, expand the BDB distribution archive file, and change to the `build_unix` subdirectory. Then issue the following commands:

```
env CC=clang CFLAGS="-fvisibility=default -arch x86_64"
```

[7] http://www.oracle.com/us/products/database/berkeley-db/

[8] Note that Oracle does provide an already-compiled distribution, but it includes only a 32-bit build.

```
     ../dist/configure
make
sudo make install
```

The `CFLAGS` setting above causes the build to produce universal binaries. You can tidy up after the build with the command `make realclean`. Once you have built and installed Berkeley DB, define the following environment variables so that the Parliament build infrastructure can find the libraries:

```
BDB_VERSION=5.3
BDB_HOME=/usr/local/BerkeleyDB.5.3
```

With recent Apple compiler versions, the `make` command above may fail with an error message about `__atomic_compare_exchange`. If so, replace all instances of that identifier with `__atomic_compare_exchange_db` in the file `src/dbinc/atomic.h`. Then run the `make` command again.

3.3.3 Building BDB for Linux

On Linux, Berkeley DB follows the usual pattern of software based on the autoconf/automake/libtool suite. Here we modify that procedure slightly to create both 32- and 64-bit builds. (For Linux distributions that only support 64-bits, skip the 32-bit commands below.) First, decompress and unarchive the BDB distribution, and change to the `build_unix` subdirectory. Next decide where you want to install Berkeley DB. The default location is `/usr/local/BerkeleyDB.5.3`, but whatever location you choose will be called «`dir`» in the instructions below. Once you have chosen this location, issue the following commands:

```
env CFLAGS="-m32" ../dist/configure --prefix=«dir»/32
make
sudo make install
make realclean
env CFLAGS="-m64" ../dist/configure --prefix=«dir»/64
make
sudo make install
```

You can tidy up after the build with the command `make realclean`. Once you have built and installed Berkeley DB, define the following environment variables so that the Parliament build infrastructure can find the libraries:

```
BDB_VERSION=5.3
BDB_HOME=«dir»
```

With the newest GCC compiler, the `make` command above may fail with an error message about `__atomic_compare_exchange`. If so, replace all instances of that identifier with `__atomic_compare_exchange_db` in the file `src/dbinc/atomic.h`. Then run the `make` command again.

3.4 Building the Boost Libraries

Since the Boost project is unfamiliar to many, here is an introduction taken from the Boost web site:[9]

> Boost provides free peer-reviewed portable C++ source libraries.
>
> We emphasize libraries that work well with the C++ Standard Library. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications. The Boost license encourages both non-commercial and commercial use.
>
> We aim to establish "existing practice" and provide reference implementations so that Boost libraries are suitable for eventual standardization. Ten Boost libraries are already included in the C++ Standards Committee's Library Technical Report (TR1) and in the new C++11 Standard. C++11 also includes several more Boost libraries in addition to those from TR1. More Boost libraries are proposed for TR2.

To get started, download the Boost source distribution (version 1.68.0 or later) from the Boost web site. Unpack this to a handy location on your disk. This location may be anywhere you like, but note that it is not temporary. We will call this location `BOOST_ROOT`, and you need to define an environment variable pointing there, such as

```
export BOOST_ROOT=~/boost_1_68_0
```

or

```
set BOOST_ROOT=C:\boost_1_68_0
```

[9] http://boost.org/

From Parliament's point of view, there are two primary components contained within `BOOST_ROOT`. The first (and most obvious) is the boost libraries themselves. Most of these are so-called "header-only" libraries, meaning that there is no pre-compiled library code or shared or dynamic library. All of the code of such libraries is referenced via `#include` directives and compiled along with the calling code. Such libraries are extremely convenient, because they require virtually zero setup. Parliament uses several Boost libraries that are not header-only. We will discuss how to build these libraries below.

The second major Boost component is Boost.Build. This is a cross-platform build system (located in the sub-directory `tools/build`) that is written in a specialized interpreted language whose interpreter is a command line program called `b2`. The Boost community does not provide `b2` binaries. Rather, the Boost distribution contains a bootstrapping script that builds `b2` from source on your platform. At a command line, go to the `BOOST_ROOT` directory and issue the appropriate one of these commands:

`bootstrap.bat`	(on Windows)
`./bootstrap.sh --with-toolset=clang`	(on Macintosh)
`./bootstrap.sh`	(on Linux)

Once the script finishes, you will find the executable `b2` (or `b2.exe` on Windows) in `BOOST_ROOT`. Move this binary to any location on your path.

Building the Boost libraries using Boost.Build is relatively simple. To build the minimal set of libraries required for Parliament, follow the directions below for your platform.

3.4.1 Building Boost on Windows

Open a Command Prompt and change to the `BOOST_ROOT` directory and issue the following command:

```
b2 -q --build-dir=build-msvc --stagedir=stage-msvc --with-atomic
--with-chrono --with-date_time --with-filesystem --with-locale
--with-log --with-regex --with-system --with-test --with-thread
toolset=msvc-14.1 define=BOOST_TEST_ALTERNATIVE_INIT_API
address-model=64,32 variant=debug,release threading=multi
link=shared,static runtime-link=shared stage
```

The build process may issue warnings about missing components, such as Python and ICU. These warnings are innocuous. When the build finishes, the libraries will be located in `stage-msvc/lib`. The directory `build-msvc` contains the intermediate build products and can be deleted to save disk space.

3.4.2 Building Boost on Macintosh

Open a Terminal window, change to the `BOOST_ROOT` directory, and issue the following command:

```
b2 -q --build-dir=build-clang --stagedir=stage-clang --with-atomic
--with-chrono --with-date_time --with-filesystem --with-locale
--with-log --with-regex --with-system --with-test --with-thread
--layout=versioned threading=multi toolset=clang
define=BOOST_TEST_ALTERNATIVE_INIT_API address-model=32_64
variant=debug,release link=shared,static runtime-link=shared
cxxflags="-std=c++17 -fvisibility=default" linkflags=-std=c++17
stage
```

The build process may issue warnings about missing components, such as Python and ICU. These warnings are innocuous. When the build finishes, the libraries will be located under `stage-clang/lib`. The directory `build-clang` contains the intermediate build products and can be deleted to save disk space.

3.4.3 Building Boost on Linux

At a shell, change to the `BOOST_ROOT` directory and issue the following command:

```
b2 -q --build-dir=build-gcc --stagedir=stage-gcc --layout=versioned
--with-atomic --with-chrono --with-date_time --with-filesystem
--with-locale --with-log --with-regex --with-system --with-test
--with-thread define=BOOST_TEST_ALTERNATIVE_INIT_API
address-model=64,32 variant=debug,release threading=multi
link=shared,static runtime-link=shared cxxflags=-std=c++17
linkflags=-std=c++17 stage
```

If your compiler does not support the C++17 standard, change “17” to “14” or “11” in the above command (in two places). The build process may issue

warnings about missing components, such as Python and ICU. These warnings are innocuous. When the build finishes, the libraries will be located in `stage-gcc/lib`. The directory `build-gcc` contains the intermediate build products and can be deleted to save disk space.

3.5 Configuring Boost.Build

Next we need to configure Boost.Build so that it can build Parliament. Start by defining the following environment variables:

- `BDB_VERSION`: As described above in Section 3.3.
- `BDB_HOME`: As described above in Section 3.3.
- `BOOST_VERSION`: The version of Boost (currently 1_68_0).
- `BOOST_ROOT`: As described above in Section 3.4.
- `BOOST_BUILD_PATH`: The sub-directory `tools/build` of `BOOST_ROOT`.
- `BOOST_TEST_LOG_LEVEL`: Controls the output volume of the C++ unit tests. Possible values and their meanings are listed in Table 3.2.

Value	Meaning
`all`	report all log messages
`success`	the same as all
`test_suite`	show test suite messages
`message`	show user messages *(useful default)*
`warning`	report warnings issued by user
`error`	report all error conditions
`cpp_exception`	report uncaught C++ exceptions
`system_error`	report system-originated non-fatal errors
`fatal_error`	report only fatal errors
`nothing`	do not report any information

Table 3.2: Possible Values of `BOOST_TEST_LOG_LEVEL`

- `JAVA_HOME`: The location of your JDK installation, which must be version 8 or higher.

Next we create two Boost.Build configuration files, `site-config.jam` and `user-config.jam`. Boost.Build reads these files on startup. The two are separate so that the first one can be installed and maintained by a system administrator, and the second by the individual user. These files can be placed in a number of locations. Table 3.3 explains where Boost.Build searches to find these files.

OS	site-config.jam	user-config.jam
Unix-like:	/etc $HOME $BOOST_BUILD_PATH	$HOME $BOOST_BUILD_PATH
Windows:	%SystemRoot% %HOMEDRIVE%%HOMEPATH% %HOME% %BOOST_BUILD_PATH%	%HOMEDRIVE%%HOMEPATH% %HOME% %BOOST_BUILD_PATH%

Table 3.3: Boost.Build Search Paths for Configuration Files

Some people prefer to keep these files together with their Boost.Build installation, placing them in `BOOST_BUILD_PATH`. There are example `site-config.jam` and `user-config.jam` files in that directory, and so you will have to replace (or rename) them if you choose this option. Others prefer to separate the configuration files from the Boost.Build installation so that they can update to a new version of Boost.Build without having to first save `site-config.jam` and `user-config.jam` and then restore them after the update is complete. On Windows, using the `HOME` location requires defining the environment variable `HOME`, because Windows does not define it by default.

Within your working copy of the Parliament repository, in the directories `doc/MacOS`, `doc/Windows`, and `doc/Linux`, you will find example configuration files for Macintosh, Windows, and Linux respectively that you can copy and customize. The most important customization that you need to make is to remove (or comment out) any lines in `user-config.jam` for compiler versions that you have not installed.

3.6 Building Parliament Itself

You are now ready to build Parliament. To do so, issue the command `ant` from the root directory of your Parliament working copy — *but don't try this until you have read the next couple of paragraphs*. This command will build the entire repository, including both native and Java code, and create a distribution-ready package in this directory:

`targets/Parliament-v`***`X.Y.Z`***`-`***`toolset`***

where ***`X.Y.Z`*** is the Parliament version number and ***`toolset`*** indicates the platform on which the distribution runs, as shown in Table 2.1. The `ant clean` command will delete all build products.

The file `build.properties`, located in the root of your working copy, controls the Parliament build. This file does not exist by default. If the build does not find it, it uses `build.properties.default` instead. The latter contains the build options used to create an official release of Parliament, which typically includes several different release builds. To build a single debug variant, copy `build.properties.default` to `build.properties` and then customize the latter. The file itself contains instructions. Please change `build.properties.default` directly only if you intend to update the official release build options.

The `build.properties` file also contains an option that disables the native code unit tests, `skipNativeUnitTest`. This is important when cross-compiling, e.g., building 64-bit binaries on 32-bit Windows. If you use this option, be sure to change it only in `build.properties`, not `build.properties.default`.

For more targeted builds, ant can be run from many sub-directories in your working copy. Here is a road map to the various sub-projects:

- `jena/JenaGraph`: Enables Jena to use Parliament for storing models
- `jena/JosekiExtensions`: Extensions to Joseki that, together with JenaGraph, create a SPARQL endpoint on top of Parliament
- `jena/JosekiParliamentClient`: A client-side Java library for communicating with a Joseki-Parliament SPARQL endpoint
- `jena/SpatialIndexProcessor`: A JenaGraph add-on for processing

spatial queries efficiently

- `jena/TemporalIndexProcessor`: A similar add-on to speed up temporal queries
- `LUBM`: A customized version of the Lehigh University Benchmark[10] for easy performance testing of Parliament
- `Parliament`: The native code at the heart of Parliament, plus its JNI interface

When working on the native code portions of Parliament, it can be useful to run the `b2` portion of the build directly. To do so, change directory to `KbCore` (for the Parliament DLL itself), `AdminClient` (for the command line interface to Parliament), or `Test` (for the unit tests). These directories are located within the `Parliament` sub-directory of your working copy. Then issue the command

```
b2 -q «build-options»
```

Here «`build-options`» is a placeholder for one set of build options from `build.properties`, described above. The `-q` option causes `b2` to quit immediately whenever an error occurs, so that you do not have to scroll up through the build output to verify that the build was successful.

[10] http://swat.cse.lehigh.edu/projects/lubm/

Appendix A

Building Berkeley DB for Windows

The build infrastructure for Berkeley DB is not particularly friendly for Windows. Therefore, the Parliament source code repository includes pre-built versions of Berkeley DB for Visual Studio, in both 32- and 64-bit flavors. (Oracle does provide an already-compiled binary distribution, but it includes only a 32-bit build.)

This appendix is provided primarily to guide the developer who needs to update the pre-built libraries, e.g., for a new version of Berkeley DB or to build with a different compiler. If this does not apply to you, then you may skip this appendix.

1. Download the source code distribution for Berkeley DB and unzip it to a directory of your choice on your local disk. Unless otherwise specified, all paths mentioned below are relative to this location.
2. In Visual Studio 2017, open this solution file:

   ```
   build_windows\Berkeley_DB_vs2010.sln
   ```

   Allow the Visual Studio Conversion Wizard to convert the projects to its file format.
3. For each configuration-platform pair listed in Table A.1, choose Build and then Configuration Manager from the menu and select that con-

figuration and platform. Then, in the Solution Explorer, right-click the "db" project and choose "Project Only / Build Only db" from the menu.

Configuration	Platform
Debug	Win32
Release	Win32
Debug	x64
Release	x64

Table A.1: Visual Studio Configuration-Platform Pairs

4. Close Visual Studio.
5. Create the directory hierarchy shown in Figure A.1 at the root level of your dependencies directory.

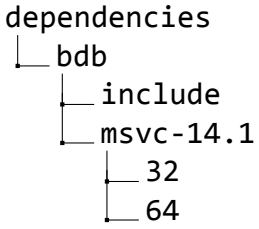

Figure A.1: BDB Directory Hierarchy

6. Copy these files into the `msvc-14.1\32` directory created in Step 5 above:

```
build_windows\Win32\Debug\libdb53d.dll
build_windows\Win32\Debug\libdb53d.lib
build_windows\Win32\Debug\libdb53d.pdb
build_windows\Win32\Release\libdb53.dll
build_windows\Win32\Release\libdb53.lib
build_windows\Win32\Release\libdb53.pdb
```

7. Copy these files into the `msvc-14.1\64` directory created in Step 5 above:

```
build_windows\x64\Debug\libdb53d.dll
build_windows\x64\Debug\libdb53d.lib
build_windows\x64\Debug\libdb53d.pdb
build_windows\x64\Release\libdb53.dll
build_windows\x64\Release\libdb53.lib
build_windows\x64\Release\libdb53.pdb
```

8. Copy these files into the `include` directory created in Step 5 above:

   ```
   build_windows\*.h
   ```

9. Commit the above changes to Parliament's source code repository.
10. Finally, delete the build directory that you created in Step 1 by unzipping the source code distribution.

Glossary

BBN Raytheon BBN Technologies , Inc. (pp. 1, 2)

CDT Eclipse C/C++ Developers Toolkit is a set of Eclipse plug-ins to facilitate C and C++ development in that environment. (p. 29)

DAML DARPA Agent Markup Language (p. 1)

DARPA Defense Advanced Research Projects Agency (p. 1)

DLL Dynamic Link Library (pp. 11, 22–26, 38)

HTTP HyperText Transfer Protocol (pp. 7, 8)

IDE Integrated Development Environment (p. 29)

JNI Java Native Interface is a facility within the Java platform to allow Java code to directly call native code. (pp. 29, 38)

JVM Java Virtual Machine (pp. 5, 10, 19, 22–26, 28)

KB Knowledge Base (pp. 5, 13, 14, 19, 22)

NAS Network-Attached Storage (p. 18)

OWL Web Ontology Language (p. 2)

Parliament Parliament™ is BBN's triple store, so named because "parliament" is the collective noun for a group of owls. A triple store is a specialized database tuned to the unique needs of the Semantic Web data representation. (pp. i, i–iii, 1–3, 2–9, 8–10, 9–14, 13–15, 14–18, 17–20, 19–21, 20–22, 21–24, 23–28, 27–30, 29–33, 32–37, 36–39, 41)

RAID Redundant Array of Inexpensive Disks (p. 18)

RDF Resource Description Framework (pp. 1, 2, 8, 14, 15, 20)

RDFS RDF Schema (p. 2)

SAN Storage Area Network (p. 18)

SPARQL SPARQL Protocol and RDF Query Language . (If this seems confusing, it is because SPARQL is a recursive acronym.) (pp. 2, 4, 7, 8, 10, 37)

URL Uniform Resource Locator is the address of a page on the World Wide Web, typically starting with the prefix "http:". (pp. 6–8)

W3C World Wide Web Consortium (p. 2)

Bibliography

[1] Robert Battle and Dave Kolas. “Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL”. In: *Semantic Web* 3.4 (Oct. 2012), pp. 355–370. URL: http://www.semantic-web-journal.net/sites/default/files/swj176_2.pdf (cit. on p. 3).

[2] Dave Kolas. *Spatiotemporal Indexing in Parliament with Jena*. Report. Jena Users Workshop, 2010 Semantic Technology Conference, San Francisco, CA: Raytheon BBN Technologies, June 23, 2010. URL: http://asio.bbn.com/2010/06/semtech/Parliament_Spatiotemporal_Indexing.pdf (cit. on p. 3).

[3] Dave Kolas, Ian Emmons, and Mike Dean. “Efficient Linked-List RDF Indexing in Parliament”. In: *Proceedings of the Fifth International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2009)* (Washington, DC). Lecture Notes in Computer Science 5823. Springer, Oct. 2009, pp. 17–32. URL: http://ceur-ws.org/Vol-517/ (cit. on p. 2).