

Parliament™ User Guide

Ian Emmons

March 30, 2011

Contents

1	Introduction	1
1.1	Background	1
1.2	Storage and Inference	2
1.3	Parliament™ Architecture	3
2	Deploying and Using Parliament™	4
2.1	Binary Distribution Contents	5
2.2	The Parliament™ Configuration File	6
2.3	Parliament™ as a Server	11
2.3.1	Parliament™ with Jena, Joseki, and Jetty	11
2.3.2	Parliament™ in Other Servlet Containers	15
2.3.3	Parliament™ with Sesame 1.2 and Tomcat	16
2.4	Using Parliament™ In-Process	19
2.4.1	Parliament™ with Jena	19
2.4.2	Parliament™ with Sesame 1.2	22
2.4.3	Parliament™ via Java	22
2.4.4	Parliament™ via C++	22
2.5	Configuring Indexes	23
2.6	The ParliamentAdmin Utility	23
2.7	Troubleshooting	23
3	Building Parliament™	27
3.1	Platforms and Prerequisites	28
3.2	Configuring Eclipse	30
3.3	Building Berkeley DB	30
3.3.1	Building BDB for Windows	31
3.3.2	Building BDB for Macintosh	31

CONTENTS

iii

3.3.3	Building BDB for Linux	31
3.4	Building the Boost Libraries	32
3.4.1	Building Boost on Windows	34
3.4.2	Building Boost on Macintosh	35
3.4.3	Building Boost on Linux	35
3.5	Configuring Boost.Build	36
3.6	Building Parliament™ Itself	38
A	Building Berkeley DB for Windows	40

List of Figures

1.1	Layered Parliament™ Architecture	3
2.1	Issuing a SPARQL select query with Jena	14
2.2	Using Parliament™ Via a Remote Jena Model	15
2.3	Configuring Parliament™ in Sesame's system.conf File	18
2.4	Creating a Jena Model backed by Parliament™	19
2.5	Using a Jena Model backed by Parliament™	20
A.1	BDB Directory Hierarchy	43

List of Tables

2.1	Joseki Server Connection URL's	13
3.1	Supported Platforms and Compilers	28
3.2	Possible Values of BOOST_TEST_LOG_LEVEL	36
3.3	Boost.Build Search Paths for Configuration Files	37
A.1	Visual Studio Configuration-Platform Pairs	41

Chapter 1

Introduction

Parliament[™] is a high-performance triple store and reasoner designed for the Semantic Web.¹ Parliament² was originally developed under the name DAML DB³ and was extended by BBN Technologies⁴ for internal use in its R&D programs. BBN released Parliament as an open source project under the BSD license⁵ on SemWebCentral⁶ in 2009.

Parliament[™] is a trademark of BBN Technologies, Inc., and 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 RDF⁷ represents data as a directed 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.

¹<http://www.w3.org/2001/sw/>

²<http://parliament.semwebcentral.org/>

³<http://www.daml.org/2001/09/damldb/>

⁴<http://bbn.com/>

⁵<http://opensource.org/licenses/bsd-license.php>

⁶<http://semwebcentral.org/>

⁷<http://www.w3.org/RDF/>

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 called “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 Storage and Inference

Parliament implements a high-performance storage engine that is compatible with the RDF and OWL⁸ standards. However, it is not a complete data management system. Parliament is typically paired with a query processor, such as Sesame⁹ or Jena,¹⁰ to implement a complete data management solution that complies with the RDF, OWL, and SPARQL¹¹ standards for data representation, ontology, and query, respectively.

In addition, Parliament includes a high-performance rule engine, which serves as an efficient means of inference. In other words, the rule engine 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 most of RDFS inference plus selected elements of OWL-Lite.

⁸<http://www.w3.org/2007/OWL/>

⁹<http://www.openrdf.org/>

¹⁰<http://openjena.org/>

¹¹http://www.w3.org/2009/sparql/wiki/Main_Page

1.3 Parliament™ Architecture

Figure 1.1 depicts the layered architecture of Parliament. Parliament is a triple store and a rule engine, but it does not include a query processor. Therefore, it is typically paired with a third-party query processor, such as Jena or Sesame. The core of Parliament is written in C++, but the integration layers for Jena and Sesame are Java code. The Jena integration includes some useful extras, such as support for named graphs and temporal and geospatial indexes.

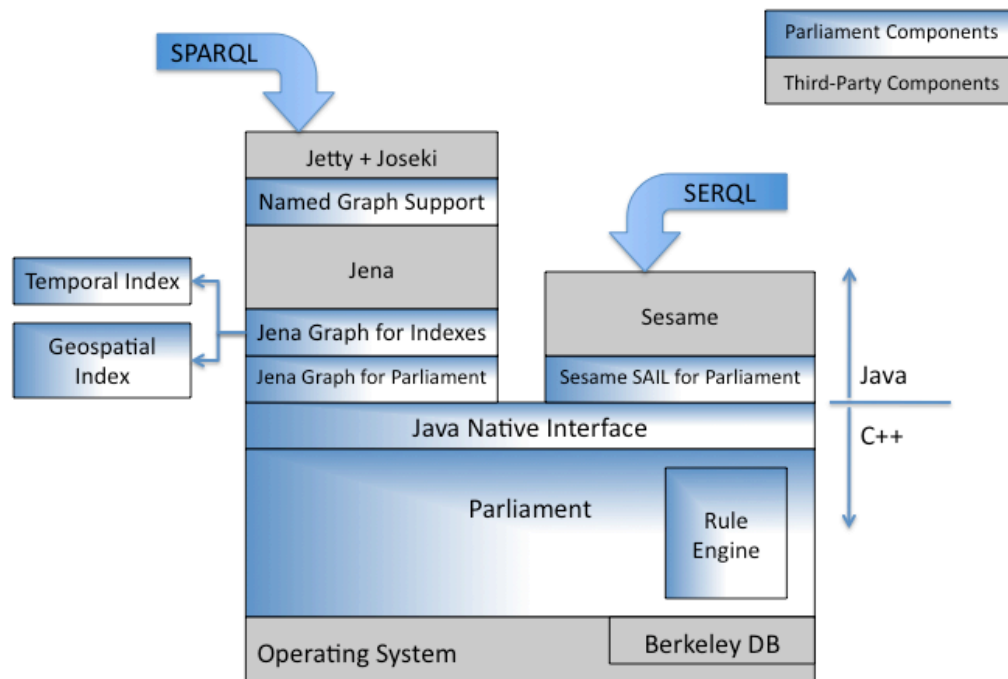


Figure 1.1: Layered Parliament™ Architecture

Chapter 2

Deploying and Using Parliament™

This chapter discusses how to use a binary distribution to deploy Parliament. There are two ways to acquire a binary distribution: either download pre-built binaries from the Parliament web site,¹ or build them yourself, as discussed in Chapter 3.

There are several ways to use Parliament. Most commonly, Parliament is paired with a query processor, such as Jena or Sesame, and a servlet engine, such as Jetty or Tomcat, to implement a complete knowledge base server application. Another way to use Parliament is as an in-process library, again paired with a query processor. Less frequently, Parliament is used as an in-process library by itself, without the benefit of a query processor.

This chapter starts with two topics that are generally useful no matter how you plan to use Parliament: the contents of the binary distribution and the configuration file. The chapter then examines each of the usage modes of Parliament, and concludes with discussions of Parliament utilities and troubleshooting.

Note that as of this writing, Parliament supports Jena 2.6 and Sesame 1.2. In particular, Sesame 2.0 and later are not yet supported.

¹<http://parliament.semwebcentral.org/>

2.1 Binary Distribution Contents

A binary distribution of Parliament is a compressed archive containing a number of different artifacts. (If you are building Parliament yourself according to the instructions in Chapter 3, then the contents of the “target/artifacts” directory are identical to the contents of a binary distribution.) After extracting the archive contents, you will see the following:

- This document (ParliamentUserGuide.pdf)
- A ParliamentKB directory: A nearly ready-to-run knowledge base (KB) server and SPARQL endpoint. This combines the Parliament triple store with the Jetty servlet container, the Joseki² SPARQL endpoint, and the Jena/ARQ query processor.
- One or more directories of native build products: Each such directory contains the native build products for one compiler (libraries, shared libraries, executables, and the like). The directory names correspond to the designation for the compiler according to Boost.Build. For instance, on Windows you will see `msvc-9.0` (Visual Studio 2008) and `msvc-10.0` (Visual Studio 2010). On Macintosh, you will see `darwin-4.0.1` (GCC 4.0.1 on MacOS 10.5) or `darwin-4.2.1` (GCC 4.2.1 on MacOS 10.6).
- An include directory: This contains the header files required to write C++ code that directly calls Parliament. These are rarely used, because most of the time Parliament is accessed by Java code through its JNI interface.
- On Windows, a Redistributable Packages directory: This directory contains installers that update the C run-time libraries for Visual C++ on machines that do not have Visual C++ itself installed. It is recommended that you not install any of these until you discover that they are necessary. See Section 2.7 for details.
- Jar files: These serve a variety of purposes, and their usage will be called out in the following sections.

²<http://www.joseki.org/>

- A javadoc directory: This contains the output of the javadoc tool from all of the Java code in the Parliament code base. In some cases this documentation is nicely done, and in others it clearly needs further work.
- A License directory: This contains the license for Parliament as well as the licenses for all of the components that are used by Parliament.

The sections below show how to use these artifacts to deploy Parliament in several different usage modes.

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 full path of the configuration file.
2. Otherwise, and on Windows only, Parliament will look for the file `ParliamentConfig.txt` in the same directory as the Parliament DLL.

The configuration file is a plain text file containing name-value pairs, comments (preceded by a '#'), and blank lines. 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 the following. (Note that the setting names are case-insensitive.)

kbDirectoryPath The path of the directory from which the Parliament knowledge base files are loaded (and in which they will be stored, if they do not yet exist). Note that this directory must exist before Parliament is started, or an exception will be thrown.
Default: The current working directory, "."

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

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

uriTableName 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 The name of the file containing the mapping from tuples of numeric resource identifiers to numeric statement identifiers. This file is omitted from a Parliament KB by default, and is included only if the “keepDupStmtIdx” setting is turned on. This file is managed by Berkeley DB. *Default:* `stmt2id.db`

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”

keepDupStmtIdx When set to “yes”, Parliament will maintain an extra file which maps tuples of resources to statement identifiers. This allows Parliament to quickly decide if a statement assertion is referencing an already-asserted statement, or if a new statement must be inserted. Generally, this does not result in a noticeable speedup, because the default method for checking for an existing statement (by traversing the shortest of the subject, predicate, and object linked lists) is plenty fast enough. This option also results in a dramatic increase in the amount of disk space consumed. *Default:* “no”

- 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 4GB 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"*

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 setting and “SubclassRule” are turned on, this enables the following inference rule:

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

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 “owl:FP” stands for “owl: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 “owl:IFP” stands for “owl: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™ as a Server

2.3.1 Parliament™ with Jena, Joseki, and Jetty

As noted in Section 2.1, the Parliament binary distribution contains a directory called `ParliamentKB` that is an almost ready-to-run knowledge base (KB) server application built from a combination of the Jetty servlet container, the Joseki SPARQL endpoint, the Jena/ARQ query processor, and the Parliament triple store. This section describes how to deploy this server.

1. Copy the `ParliamentKB` directory to the location where you would like it to reside. Any location will do. On Windows, `C:\Program Files\ParliamentKB` is a likely choice. On UNIX-like systems, a customary spot might be `/usr/local/ParliamentKB`.
2. From your binary distribution of Parliament, choose one of the directories of native build products that is appropriate for your operating system. (Remember, these are the directories that are named after the compiler that produced them.) Depending on the operating system, there may be some additional considerations:
 - On most platforms, you will need to further choose between 32-bit and 64-bit builds. On such platforms, these reside in sub-directories `32` and `64` within the compiler directory. You should use the 64-bit version for knowledge bases with more than 5 to 10 million statements.
 - The Macintosh build contains three-way universal binaries for the x64, x86, and PPC hardware architectures. Use a 64-bit version of Java to put Parliament into 64-bit mode. The `-d32` and `-d64` command line switches to the `java` command select 32-bits and 64-bits, respectively.

- If you have chosen to run the `msvc-8.0` (Visual C++ 2005) or `msvc-9.0` (Visual C++ 2008) build on a non-development machine, then you may need to install the run-time libraries. You can find installers for these in the “Redistributable Packages” subdirectory of your binary distribution of *Parliament*.
3. Once you have chosen your build, copy the following files from it to the `ParliamentKB` directory:
 - All of the DLL’s or shared libraries — there should be two. One is *Parliament* itself, and the other is Berkeley DB. The exact filenames differ due to platform-specific naming conventions.
 - `ParliamentAdmin` (or `ParliamentAdmin.exe` on Windows)
 - *Do not* copy `ParliamentConfig.txt`, because the `ParliamentKB` directory already contains a copy of this file that has been customized for this usage.
 4. On Windows only, copy the contents of `ParliamentKB\windows-32` (for a 32-bit deployment) or `ParliamentKB\windows-64` (for a 64-bit KB) into `ParliamentKB`. This will enable the *Parliament* server to run as a Windows service.
 5. Customize the *Parliament* configuration file `ParliamentConfig.txt` as necessary. In particular, the `kbDirectoryPath` setting defaults to the directory `ParliamentKB/data`. To place your *Parliament* KB files in a different location, simply 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 KB’s and switching between them.
 6. There are a number of scripts in the `ParliamentKB` directory used to run the server:
 - `StartParliament.sh` (`StartParliament.bat` on Windows) starts *Parliament* directly.
 - On Windows, `InstallParliamentService.bat` creates a service to run the *Parliament* KB. After running this script, the KB can be started from the Services management console. The

`UninstallParliamentService.bat` script uninstalls the service.

7. You can control the amount of memory set aside for the Java virtual machine by setting `MIN_MEM` and `MAX_MEM` in the `start` (or `install`) script. 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 JVM 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. These settings default to 128 MB and 512 MB, which are reasonable for values for machines with 4 to 8 GB of total memory.

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

8. The Jetty configuration file at `ParliamentKB/conf/jetty.xml` controls the runtime configuration for the server. By default, this tells the server to run on port 8080 and to deploy any war file located in the `ParliamentKB/webapps` directory.

At this point you have a working SPARQL endpoint whose underlying storage layer is a Parliament instance. Table 2.1 lists the connection URL's, which are discussed below. You should substitute the proper

HTTP interface:	<code>http://localhost:8080/parliament</code>
SPARQL endpoint:	<code>http://localhost:8080/parliament/sparql</code>
Bulk loader:	<code>http://localhost:8080/parliament/bulk</code>

Table 2.1: Joseki Server Connection URL's

host name for “localhost” in these URL's whenever you want to access the server remotely.

You can use the Parliament server interactively by pointing your web browser at the HTTP interface URL in Table 2.1. More commonly, however, you will want to issue queries programmatically. Typically, this means writing some code to send SPARQL-compliant HTTP requests, or using a library that can do so on your behalf. One such library is

Jena itself. Figure 2.1 shows sample code for issuing a SPARQL select query.

```
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:8080/parliament/sparql", // SPARQL endpoint URL
        sparqlSelectquery); // SPARQL query string
    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

Parliament also provides a Java library, `JosekiParliamentClient.jar`, to help programmers interact with the Parliament server. 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. You can find this file in the Parliament binary distribution (see Section 2.1). Then create a `RemoteModel` object as shown in Figure 2.2. You can treat this like any Jena Model object to access and manipulate the Parliament repository identified by the URL's passed to the constructor. 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 exposes the bulk loading feature of the Parliament server, allowing the loading of large bodies of RDF without running afoul of the limitations of Joseki's SPARQL UPDATE implementation.

One final cautionary note: When you want to shut down the KB, it is important to shut it down gracefully. Otherwise, the Parliament files will not be flushed to disk before they are closed, and they will almost certainly be badly corrupted. If you run Parliament as a Windows service, this is generally not a problem, because Windows takes care of

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

void useRemoteParliamentModel()
{
    // Create model:
    RemoteModel rmParliament = new RemoteModel(
        "http://localhost:8080/parliament/sparql", // SPARQL endpoint URL
        "http://localhost:8080/parliament/bulk"); // bulk-insert URL

    // 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

sending a shut-down message at the appropriate times. If, however, you run Parliament explicitly via `StartParliament.sh` (`StartParliament.bat` on Windows), you will need to shut it down yourself. This is easily done by typing the command “exit” followed by «return» or «enter».

2.3.2 Parliament™ in Other Servlet Containers

The Parliament distribution contains a war file that can be deployed in any servlet container, such as Apache Tomcat or Glassfish. This section contains pointers for deploying Parliament in a servlet container other than the provided Jetty.

Apache Tomcat Notes

When deploying Parliament to Tomcat, keep in mind the following:

- The Parliament libraries must be on the `java.library.path`. If the libraries are not already on the path, set the `JAVA_OPTS` environ-

ment variable in `catalina.sh` or `catalina.bat` to contain the following:

```
-Djava.library.path=«parliament library dir»
```

- By default, Parliament stores its data files in the current working directory. In the case of Tomcat, this will be the `bin` subdirectory of the location where you install Tomcat. To change this, change the `kbDirectoryPath` setting in `ParliamentConfig.txt` to the absolute path where you would like your files to reside.
- If Parliament is redeployed into a running instance of Tomcat, the server will need to be restarted.

Glassfish Notes

When deploying Parliament to Glassfish, note the following:

- Glassfish sets its own `java.library.path`. The Parliament libraries must either be copied into `«Glassfish install dir»/lib` or the Glassfish configuration must be altered such that the libraries are available on `java.library.path`.
- By default, Parliament stores its data files in the current working directory. In the case of Glassfish, this will be `«Glassfish install dir»/domains/domain1/config`. To change this, change the `kbDirectoryPath` setting in `ParliamentConfig.txt` 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.

2.3.3 Parliament[™] with Sesame 1.2 and Tomcat

To create a knowledge base server based on Parliament, Sesame 1.2.x,³ and Tomcat, do the following:

³Note that Sesame 2.0 and greater are not yet supported. Anyone willing to contribute work towards a Sesame 3.0 integration is most welcome to do so!

1. Install Apache Tomcat 5.5 or greater.⁴
2. Install Sesame 1.2.x.⁵
3. Deploy Sesame into Tomcat. Instructions on how to do this can be found in the file `doc/users/usersguide.html`, within the Sesame 1.2.x distribution, under section 2.2.
4. Copy the files `ParliamentSail.jar` and `Parliament.jar` into the directory `webapps/sesame/WEB-INF/lib` under the Tomcat installation.
5. From your binary distribution of Parliament, choose one of the directories of native build products that is appropriate for your operating system. (Remember, these are the directories that are named after the compiler that produced them.) Depending on the operating system, there may be some additional considerations:
 - On most platforms, you will need to further choose between 32-bit and 64-bit builds. On such platforms, these reside in sub-directories 32 and 64 within the compiler directory. You should use the 64-bit version for knowledge bases with more than 5 to 10 million statements.
 - The Macintosh build contains three-way universal binaries for the x64, x86, and PPC hardware architectures. Use a 64-bit version of Java to put Parliament into 64-bit mode. The `-d32` and `-d64` command line switches to the `java` command select 32-bits and 64-bits, respectively.
 - If you have chosen to run the `msvc-8.0` (Visual C++ 2005) or `msvc-9.0` (Visual C++ 2008) build on a non-development machine, then you may need to install the run-time libraries. You can find installers for these in the “Redistributable Packages” subdirectory of your binary distribution of Parliament.
6. Once you have chosen your build, copy the following files from it to the `bin` sub-directory of your Tomcat installation:

⁴<http://tomcat.apache.org/>

⁵<http://www.openrdf.org/>

- All of the DLL's or shared libraries — there should be two. One is Parliament itself, and the other is Berkeley DB. The exact filenames differ due to platform-specific naming conventions.
 - ParliamentAdmin (ParliamentAdmin.exe on Windows)
 - ParliamentConfig.txt
7. Update webapps/sesame/WEB-INF/system.conf to add the XML element shown in Figure 2.3 to the <repositorylist> element. You should replace C:/MyKbDirectory/ with your own directory — this is the location where Parliament will create its data files, and the directory must exist before starting Sesame. (You should also change the id and title.)

```
<repository id="MyRepository">
  <title>My RDF Repository</title>
  <sailstack>
    <sail class=
      "com.bbn.parliament.sesame.sail.KbSyncRdfSchemaRepository"/>
    <sail class="com.bbn.parliament.sesame.sail.KbRdfSchemaRepository">
      <param name="dir" value="C:/MyKbDirectory"/>
    </sail>
  </sailstack>
  <acl worldReadable="true" worldWritable="true"/>
</repository>
```

Figure 2.3: Configuring Parliament[™] in Sesame's system.conf File

8. Customize the Parliament configuration file ParliamentConfig.txt as necessary. Note that the kbDirectoryPath setting is ignored in favor of the dir parameter in the system.conf file (see Figure 2.3).
9. Start Tomcat. Tomcat uses port 8080 by default, so you may see an error message if this port is already in use.
10. Connect to <http://localhost:8080/sesame/> using your web browser, and select "My RDF Repository".

From this point on, simply use Sesame as you normally would, and all RDF storage and access will be redirected to the underlying Parliament instance.

One final cautionary note: When you want to shut down the KB, it is important to shut it down gracefully. Otherwise, the Parliament files will not be flushed to disk before they are closed, and they will almost certainly be badly corrupted. Consult the Tomcat documentation for shutdown instructions.

2.4 Using Parliament™ In-Process

2.4.1 Parliament™ with Jena

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 2.4. The createParliamentModel method returns

```
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 2.4: Creating a Jena Model backed by Parliament™

a Jena model 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 2.5 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 2.5: Using a Jena Model backed by *Parliament*TM

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

1. From your binary distribution, you will need *JenaParliament.jar* and *Parliament.jar*, 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.)
2. From your binary distribution of *Parliament*, choose one of the directories of native build products that is appropriate for your

⁶<http://openjena.org/>

operating system. (Remember, these are the directories that are named after the compiler that produced them.) Depending on the operating system, there may be some additional considerations:

- On most platforms, you will need to further choose between 32-bit and 64-bit builds. On such platforms, these reside in sub-directories 32 and 64 within the compiler directory. You should use the 64-bit version for knowledge bases with more than 5 to 10 million statements.
 - The Macintosh build contains three-way universal binaries for the x64, x86, and PPC hardware architectures. Use a 64-bit version of Java to put Parliament into 64-bit mode. The `-d32` and `-d64` command line switches to the `java` command select 32-bits and 64-bits, respectively.
 - If you have chosen to run the `msvc-8.0` (Visual C++ 2005) or `msvc-9.0` (Visual C++ 2008) build on a non-development machine, then you may need to install the run-time libraries. You can find installers for these in the “Redistributable Packages” subdirectory of your binary distribution of Parliament.
3. Once you have chosen your build, copy the following files from it to a convenient location (e.g., the same directory used for the jar files above):
- All of the DLL’s or shared libraries — there should be two. One is Parliament itself, and the other is Berkeley DB. The exact filenames differ due to platform-specific naming conventions.
 - `ParliamentAdmin` (`ParliamentAdmin.exe` on Windows)
 - `ParliamentConfig.txt`
4. Ensure that the binaries are on your operating system’s library path. On Windows this is the `Path` environment variable. On Macintosh, use `DYLD_LIBRARY_PATH`. Linux and many UNIX platforms use `LD_LIBRARY_PATH`, though some differ. Consult your operating system documentation to be sure you use the correct environment variable.

5. Customize the Parliament configuration file `ParliamentConfig.txt` as necessary. In particular, the `kbDirectoryPath` setting defaults to the current working directory “.”. To place your Parliament KB files in a different location, simply 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 KB’s 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 `createParliamentModel` with a line that creates a `Config` instance using the default constructor and then set the `Config` fields yourself. This may be useful if you wish to centralize your application’s configuration in a single file.

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

2.4.2 *Parliament*[™] with Sesame 1.2

[Yet to be written]

2.4.3 *Parliament*[™] via Java

[Yet to be written]

2.4.4 *Parliament*[™] via C++

[Yet to be written]

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 Java Virtual Machine (JVM) when the Java code requests the loading of a DLL,⁷ 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’ve followed all the deployment instructions correctly for your chosen mode of deployment.
- 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.
- Make sure that your system is up-to-date with respect to patches. For Windows, this means a visit to the Windows Update⁸ web site. Most other operating systems have a similar facility.

⁷On UNIX-like systems, these are called “shared libraries”, but for the sake of brevity, we will use DLL (Dynamically Linked Library) as a generic term for all operating systems.

⁸<http://windowsupdate.microsoft.com/>

- The process by which Windows searches for DLL's is somewhat complex, so understanding it can often help pinpoint the problem. Starting with Windows Server 2003 and Windows XP SP1, 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

Prior to Windows Server 2003 and Windows XP SP1, the search order was a little bit different:

- The directory where the executable module for the current process is located
- The current directory
- The Windows system directory
- The Windows directory
- The directories listed in the PATH environment variable

Note that placing DLL's in the Windows or Windows system directories is strongly discouraged.

- UNIX-like systems find shared libraries by simply searching the directories in the library search path environment variable. The name of this variable differs by system: It is DYLD_LIBRARY_PATH on Macintosh, whereas on Linux it is LD_LIBRARY_PATH. Other UNIX-like systems may have still other names for this variable.
- According to the Java documentation, the JVM's list of system properties always includes a property called `java.library.path`, which is defined as a "List of paths to search when loading libraries". This suggests that all of the above operating system-specific rules could be circumvented by providing a suitable definition for this system property for the JVM in which *Parliament*

is loaded. Unfortunately, experience indicates that this property is, at best, extremely unreliable.

- When the JVM loads the Parliament DLL, Parliament itself loads several other DLL's. Even if the JVM is able to load Parliament, an unsatisfied link error will result if Parliament cannot load one of its subsidiary DLL's. Furthermore, the error message accompanying the unsatisfied link error usually does not indicate which DLL caused the load failure.

The subsidiary DLL's fall into the following three categories:

- The Berkeley DB DLL: In the deployment procedures detailed in this document, this DLL should reside in the same location as Parliament itself, and so the JVM should be able to load it.
- The C and C++ run-time libraries: On Windows, if you have chosen to run the `msvc-8.0` (Visual C++ 2005) or `msvc-9.0` (Visual C++ 2008) binaries on a non-development machine, then you may need to install the run-time libraries. You can find installers for these in the `Redistributable Packages` subdirectory of your binary distribution of Parliament.
- Various operating system DLL's: These are rarely a problem, because the OS needs to be able to find these to function.

On UNIX-like systems there is often a command called `ldd` which will show from where the operating system is trying to load subsidiary DLL's. On Macintosh, `ldd` does not exist, but the command `otool -L` performs a similar function. On Windows, there is no direct equivalent, but “Dependency Walker”⁹ and the command-line tool `dumpbin` (part of the Visual Studio installation) are helpful in this regard.

- Sometimes running `ParliamentAdmin` can help diagnose unsatisfied link errors, because this tool also loads the Parliament DLL, but it does so without involving Java. (This is because `ParliamentAdmin` is written in C++.) This technique can be used to isolate the error as either an operating system problem (if `ParliamentAdmin` fails to load the Parliament DLL) or a Java problem (if `ParliamentAdmin`

⁹<http://dependencywalker.com/>

succeeds). Even running just `ParliamentAdmin -v` to print the version requires loading the Parliament DLL, so this is an easy test.

- 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 happens, 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. Eventually, we hope to add a .Net managed code interface as well. 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, an XML-based 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 2008, SP1	msvc-9.0
	Visual Studio 2010	msvc-10.0
MacOS 10.5	Xcode 3.1.3	darwin-4.0.1
MacOS 10.6	Xcode 3.2.4	darwin-4.2.1
Ubuntu 10.04	GCC 4.4.3	gcc-4.4

Table 3.1: Supported Platforms and Compilers

Windows versions XP and above are supported. The 64-bit build has been tested only on AMD-64 or EM64T hardware (often collectively known as x64), but not Itanium (IA64). You need only one of the supported Visual Studio versions, but any combination can be installed simultaneously if you wish. Parliament’s capacity is much higher when running as a 64-bit process,¹ so one of those compilers is highly recommended. Note that Visual studio’s 64-bit compilers are *not* installed by default, so you will have to choose the “Custom” option during installation and select them explicitly. You can also re-run the Visual Studio installer to add the 64-bit tools to an existing installation.

On Macintosh, Parliament builds as a three-way universal binary (for the x64, x86, and PPC architectures) using Apple’s Xcode development tools. (Xcode includes customized versions of GCC 4.0.1 and 4.2.1.)

On Linux, Parliament has been explicitly tested only on Ubuntu 10.04. On that distribution, Parliament supports both 32- and 64-bit builds (x86 and x64). Note that you may need to install extra packages to

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

enable cross-compilation. For instance, the author installed the 64-bit edition of Ubuntu, and then added the “g++-4.4-multilib” package.

Parliament assumes the presence of the Java Developer Kit (JDK), version 6 or above. On Macintosh, this is built-in, but you will need to download and install one for Windows, and you may need to install one or more packages on Linux. Note that you will need a 64-bit JVM in order to run a 64-bit build of Parliament. On Windows 64-bit Java is a separate download and installation. On Macintosh there are several ways to switch between 32- and 64-bit Java. One is to open the Java Preferences (located in the Applications/Utilities/Java folder) and drag a 64-bit Java version to the top of the “Java Application Version” list. Another is to use the `-d32` and `-d64` command line switches to the `java` command. Other approaches are detailed on Apple’s web site.²

You will need Apache Ant version 1.7.0 or later. Again, on Macintosh this is built-in. Windows users can acquire Ant from the Apache web site,³ and Linux users can typically get ant through their package manager.

Finally, you need to install a client for the Subversion version control system.⁴ There are several different clients, depending on your operating system and your preferred mode of usage, but any of them are fine for our purposes here. On Macintosh the command line client is built-in.

Once you have a working Subversion client, you can check out a working copy of the Parliament code base like so:

```
svn --username «name» co
  https://projects.semwebcentral.org/svn/parliament/trunk «dir»
```

Change «dir» to a convenient location on your local machine. For anonymous access, the change «name» to “anonsvn” with password “anonsvn”. For project developer access via DAV, substitute your user ID for «name» and enter your site password when prompted.

²<http://developer.apple.com/java/javaleopard.html>

³<http://ant.apache.org/>

⁴<http://subversion.tigris.org/>

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 will not build at all. This may be corrected in the future, but the JNI interface between Java and native code makes this complex. Therefore the C++ projects are merely an editing convenience for now.

To setup Eclipse, you need the Helios version (3.6.1) or later of the Eclipse IDE for Java Developers, plus the Eclipse C/C++ Developers Toolkit (CDT) plug-ins. One way to acquire this set of components is to download the Eclipse IDE for Java Developers⁵, install it, run it, and then use the “Install New Software” menu item to download and install the CDT. The procedure for this changes 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.

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 fol-

⁵<http://www.eclipse.org/>

⁶<http://www.eclipse.org/cdt/>

⁷<http://www.oracle.com/us/products/database/berkeley-db/>

lowing procedures are based on BDB version 5.0.26, which is the latest as of this writing.

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 versions of Berkeley DB for Visual Studio 2010 (MSVC 10) and Visual Studio 2008 (MSVC 9). Both 32 and 64-bit builds are included.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.

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, decompress and un-archive the BDB distribution, and change to the `build_unix` subdirectory. Then issue the following commands:

```
env CFLAGS="-arch x86_64 -arch i386 -arch ppc" ../dist/configure
make
sudo make install
```

(The `CFLAGS` setting 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 variable to represent the installation directory:

```
BDB_HOME=/usr/local/BerkeleyDB.5.0
```

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

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

slightly to create both 32- and 64-bit builds. 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.5.0`, 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 variable to represent the installation directory:

```
BDB_HOME=«dir»
```

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 license¹⁰ encourages both commercial and non-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++

⁹<http://boost.org/>

¹⁰<http://www.boost.org/users/license.html>

Standards Committee's¹¹ Library Technical Report (TR1)¹² and will be in the new C++0x Standard now being finalized. C++0x will also include several more Boost libraries in addition to those from TR1. More Boost libraries are proposed for TR2.¹³

To get started, download the Boost libraries source distribution (version 1.46.1 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_46_1
```

or

```
set BOOST_ROOT=C:\boost_1_46_1
```

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 one Boost library that is not header-only called `Boost.Test`. We will discuss how to build `Boost.Test` below.

The second major Boost component is `Boost.Build`. This is a cross-platform build system (located in the sub-directory `tools/build/v2`) that is implemented in an interpreted language, and the language interpreter is a command line program called `bjam`. The Boost community does not provide `bjam` binaries. Rather, the Boost distribution contains a bootstrapping script that builds `bjam` from source on your platform. At a command line, change to the `BOOST_ROOT` directory and issue this command:

```
./bootstrap.sh --show-libraries
```

or

¹¹<http://www.open-std.org/jtc1/sc22/wg21/>

¹²<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1745.pdf>

¹³<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1810.html>

```
bootstrap.bat --show-libraries
```

The command line option causes the script to show the libraries that are not header-only. We don't really care about this, but it prevents the script from building those libraries, which we will do in a more targeted way below. Once the script finishes, you will find the bjam executable 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. To build 32-bit libraries, issue the following command:

```
bjam -q --build-dir=msvc-32/build --stagedir=msvc-32/stage
--with-test toolset=msvc-10.0,msvc-9.0 address-model=32
variant=debug,release threading=multi link=shared
runtime-link=shared stage
```

Note that this command assumes you have all supported versions of Visual Studio installed. If this is not the case, you will need to adjust the toolset argument in the obvious way. (Refer to Table 3.1 for the correspondence between Visual Studio versions and toolset identifiers.)

For 64-bit libraries, issue the following command:

```
bjam -q --build-dir=msvc-64/build --stagedir=msvc-64/stage
--with-test toolset=msvc-10.0,msvc-9.0 address-model=64
variant=debug,release threading=multi link=shared
runtime-link=shared stage
```

Again, adjust the toolset argument for the compilers you have installed. Remember that you need to explicitly choose to install Microsoft's 64-bit compilers when you install Visual Studio. Failure to do so will result in mysterious error messages whose cause is not clear.

The build process may issue several warnings about missing components, such as Python and ICU. These warnings are innocuous.

When the build finishes, the libraries you need will be located under `msvc-32/stage/lib` and `msvc-64/stage/lib`. The directories `msvc-32/build` and `msvc-64/build` contain 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:

```
bjam -q --build-dir=darwin/build --stagedir=darwin/stage
--layout=versioned --with-test architecture=combined
address-model=32_64 variant=debug,release threading=multi
link=shared runtime-link=shared stage
```

The build process may issue several warnings about missing components, such as Python and ICU. These warnings are innocuous.

When the build finishes, the libraries you need will be located under `darwin/stage/lib`. The directory `darwin/build` 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. To build 32-bit libraries, issue the following command:

```
bjam -q --build-dir=gcc-32/build --stagedir=gcc-32/stage
--layout=versioned --with-test address-model=32
variant=debug,release threading=multi link=shared
runtime-link=shared stage
```

For 64-bit libraries, issue the following command:

```
bjam -q --build-dir=gcc-64/build --stagedir=gcc-64/stage
--layout=versioned --with-test address-model=64
variant=debug,release threading=multi link=shared
runtime-link=shared stage
```

The build process may issue several warnings about missing components, such as Python and ICU. These warnings are innocuous.

When the build finishes, the libraries you need will be located under `gcc-32/stage/lib` and `gcc-64/stage/lib`. The directories `gcc-32/build` and `gcc-64/build` contain 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_HOME`: On Windows, set this to the absolute path of the `lib/db` sub-directory of your Parliament working copy. On Unix-like platforms, the default value is `/usr/local/BerkeleyDB.5.0`.
- `BOOST_ROOT`: As described above
- `BOOST_BUILD_PATH`: Set this to the sub-directory `tools/build/v2` of `BOOST_ROOT`.
- `BOOST_TEST_LOG_LEVEL`: This controls Parliament's unit test output. Possible values and their meanings are listed in Table 3.2.

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

Table 3.2: Possible Values of `BOOST_TEST_LOG_LEVEL`

- `JAVA_HOME`: Point this to your JDK installation (version 6 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	\$HOME
	\$BOOST_BUILD_PATH	\$BOOST_BUILD_PATH
Windows:	%SystemRoot%	
	%HOMEDRIVE%%HOMEPATH%	%HOMEDRIVE%%HOMEPATH%
	%HOME%	%HOME%
	%BOOST_BUILD_PATH%	%BOOST_BUILD_PATH%

Table 3.3: Boost.Build Search Paths for Configuration Files

Some people prefer to keep these configuration files together with their Boost.Build installation, and so choose the location `BOOST_BUILD_PATH`. Note that 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 the `targets/artifacts` directory. (See Section 2.1 for a discussion of this directory.) The `ant clean` command will delete all build products.

You can control the Parliament build through the file `build.properties` in the root of your working copy. 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, then copy `build.properties.default` to `build.properties` and customize it. The file itself contains instructions. Please customize `build.properties.default` directly only if you intend to change the official release build options.

The `build.properties` file also contains an option “`skipNativeUnitTest`” that disables the native code unit tests. This is important when cross-compiling, e.g., building 64-bit binaries on 32-bit Windows. If you use this option, be sure to re-enable the unit tests before you commit any changes to the Subversion repository.

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`: Java client library for sending requests to a Joseki-Parliament SPARQL endpoint
- `jena/SpatialIndexProcessor`: A `JenaGraph` extension for processing spatial queries efficiently
- `jena/TemporalIndexProcessor`: A similar extension for processing

temporal queries efficiently

- LUBM: A customized version of the Lehigh University Benchmark¹⁴ for easy performance testing of Parliament
- Parliament: The native code at the heart of Parliament and its JNI interface
- Sesame/CSameAsSail: A Sesame¹⁵ plug-in that enables Sesame 1.2 to perform same-as inferencing
- Sesame/LuceneSail: A Sesame 1.2 plug-in that integrates full-text search with SeRQL queries
- Sesame/ParliamentSail: An integration that enables Sesame 1.2 to use Parliament as a storage layer¹⁶
- Sesame/SameAsSail: Another Sesame plug-in that enables Sesame 1.2 to perform same-as inferencing
- Sesame/Swrl2Sesame: SWRL-to-Sesame conversion

When working on the native code portions of Parliament, it can be useful to run the `bjam` 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

```
bjam -q «build-options»
```

Here «build-options» is a placeholder for one set of build options from the `build.properties` file described above. The `-q` option causes `bjam` to quit immediately whenever an error occurs.

¹⁴<http://swat.cse.lehigh.edu/projects/lubm/>

¹⁵<http://www.openrdf.org/>

¹⁶Anyone willing to contribute work towards a Sesame 3.0 integration is most welcome!

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 2010 (MSVC 10) and Visual Studio 2008 (MSVC 9), and in both 32- and 64-bit flavors.¹

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 such a scenario does not apply to you, then you may skip this appendix.

1. Download the Windows source code package for Berkeley DB and unzip it to two temporary directories somewhere on your local disk. Name these locations `bdb-msvc-9.0` and `bdb-msvc-10.0`. Unless otherwise specified, all paths mentioned below are relative to one of these locations.
2. In Visual Studio 2008, open this solution file:

```
bdb-msvc-9.0\build_windows\Berkeley_DB.sln
```

Click “Finish” in the Visual Studio Conversion Wizard.

¹Note that Oracle does provide an already-compiled distribution, but it includes only a 32-bit build.

- For each configuration-platform pair listed in Table A.1, choose “Build / Configuration Manager” from the menu and select that configuration 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

- Close Visual Studio 2008. If you are using the Premium (or higher) edition of Visual Studio 2010, and you have installed the IA64 cross-compiler option, then you can skip this step. Otherwise, you must make the following edits in the Visual Studio project file(s):²

Open the file `bdb-msvc-10.0\build_windows\db.vcproj` in any text editor, and remove this line from the file:

```
<Platform Name="IA64"/>
```

Now remove the following section:

```
<Configuration Name="Debug|IA64" OutputDirectory=...>
  <Tool Name="VCPreLinkEventTool"/>
  <Tool Name="VCResourceCompilerTool"/>
  <Tool Name="VCXMLDataGeneratorTool"/>
  <Tool Name="VCManagedWrapperGeneratorTool"/>
  <Tool Name="VCAuxiliaryManagedWrapperGeneratorTool"/>
  <Tool Name="VCCLCompilerTool" Optimization=.../>
  <Tool Name="VCLinkerTool" AdditionalOptions=.../>
  <Tool Name="VCPreBuildEventTool"/>
  <Tool Name="VCPoostBuildEventTool"/>
  <Tool Name="VCCustomBuildTool"/></Configuration>
<Configuration Name="Release|IA64" OutputDirectory=...>
  <Tool Name="VCPreLinkEventTool"/>
```

²<http://forums.oracle.com/forums/thread.jspa?messageID=4266781>

```

<Tool Name="VCResourceCompilerTool"/>
<Tool Name="VCXMLDataGeneratorTool"/>
<Tool Name="VCManagedWrapperGeneratorTool"/>
<Tool Name="VCAuxiliaryManagedWrapperGeneratorTool"/>
<Tool Name="VCCLCompilerTool" Optimization=.../>
<Tool Name="VCLinkerTool" AdditionalOptions=.../>
<Tool Name="VCPreBuildEventTool"/>
<Tool Name="VCPostBuildEventTool"/>
<Tool Name="VCCustomBuildTool"/></Configuration>

```

Finally, delete this section:

```

<FileConfiguration Name="Debug|IA64"><Tool Name=.../>
</FileConfiguration>
<FileConfiguration Name="Release|IA64"><Tool Name=.../>
</FileConfiguration>

```

You can easily find these sections by searching for “IA64”. In the steps that follow, we will be building only the Berkeley DB DLL itself. If you wish to build any of the utility applications or examples, you will need to make similar modifications to those project files as well. When you are finished, save and close the project file.

5. Using Visual Studio 2010 and the remaining copy of BDB, `bdb-msvc-10.0`, repeat steps 2 and 3. Note that if you did not convert all of the project files in the previous step, the conversion wizard will report errors, but you can ignore them.
6. Close Visual Studio 2010. Create the directory hierarchy shown in Figure A.1 at the root level of your Subversion working copy.
7. From the Visual Studio 2008 build directory, `bdb-msvc-9.0`, copy the following files into the `msvc-9.0\32` directory created above:

```

build_windows\Win32\Debug\libdb50d.dll
build_windows\Win32\Debug\libdb50d.lib
build_windows\Win32\Debug\libdb50d.pdb
build_windows\Win32\Release\libdb50.dll
build_windows\Win32\Release\libdb50.lib
build_windows\Win32\Release\libdb50.pdb

```

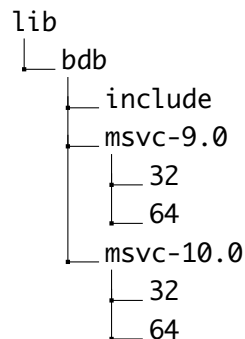


Figure A.1: BDB Directory Hierarchy

8. Also from `bdb-msvc-9.0`, copy the following files into the `msvc-9.0\64` directory created above:

```

build_windows\x64\Debug\libdb50d.dll
build_windows\x64\Debug\libdb50d.lib
build_windows\x64\Debug\libdb50d.pdb
build_windows\x64\Release\libdb50.dll
build_windows\x64\Release\libdb50.lib
build_windows\x64\Release\libdb50.pdb

```

9. From the Visual Studio 2010 build directory, `bdb-msvc-10.0`, copy the following files into the `msvc-10.0\32` directory created above:

```

build_windows\Win32\Debug\libdb50d.dll
build_windows\Win32\Debug\libdb50d.lib
build_windows\Win32\Debug\libdb50d.pdb
build_windows\Win32\Release\libdb50.dll
build_windows\Win32\Release\libdb50.lib
build_windows\Win32\Release\libdb50.pdb

```

10. And finally from `bdb-msvc-10.0`, copy the following files into the `msvc-10.0\64` directory created above:

```

build_windows\x64\Debug\libdb50d.dll
build_windows\x64\Debug\libdb50d.lib
build_windows\x64\Debug\libdb50d.pdb
build_windows\x64\Release\libdb50.dll
build_windows\x64\Release\libdb50.lib

```

`build_windows\x64\Release\libdb50.pdb`

11. From any of the Visual Studio build directories, copy the following files into the include directory created above:

`build_windows*.h`

12. Commit the changes made above to the Parliament Subversion repository.
13. Finally, delete the Visual Studio 2008 and 2010 build directories `bdb-msvc-9.0` and `bdb-msvc-10.0`.