Getting Started With VoltDB

VoltDB Inc.

Abstract

This books explains how to get started using VoltDB.

V2.8.1

Getting Started With VoltDB

VoltDB Inc.

V2.8.1

This document was generated on August 06, 2012.

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

This book provides a quick start to using VoltDB.

There are several different ways to familiarize yourself with technologies such as VoltDB, but often the best way is just to jump in and get your hands dirty with the code. This manual describes two different ways to do that:

- Chapter 3, *Hello, World!* walks you though the steps to writing the classic "Hello World" application using VoltDB.
- Chapter 4, *Running the VoltDB Example Applications* describes the sample applications that are provided with the VoltDB software. The samples are complete, working applications that you can build and run to see VoltDB in action. If you like to explore by yourself, the sample applications may be the best approach for you.

You do not need to read all of the chapters. Feel free to choose the approach that suits your learning style the best. However, for experienced programmers who choose to start with the example applications, we still recommend a quick read through Chapter 3. The Hello World tutorial provides a detailed explanation of the components of VoltDB (including the stored procedures, the SQL schema, and the project definition file) that you may find useful.

Of course, before you get started with the code, you need to install the software. The opening chapters of this book provide a brief overview of how VoltDB works and how to install the product. Later chapters explain how to use VoltDB with the Eclipse programming environment and an appendix provides a listing of the completed Hello World application.

For a more thorough description of VoltDB and all of its features, please see the accompanying manual *Using VoltDB*. Both of these books are available on the web from http://www.voltdb.com/.

Chapter 1. Getting Started

VoltDB is revolutionary database technology that combines the speed and performance of noSQL with the robustness and reliability of a traditional OLTP database. The advantages of VoltDB are:

- Best-in-class, easily scalable throughput
- Ease of use of standard SQL syntax
- Full ACID-compliance including rollback, durability, high availability, and crash recovery
- Clustered, shared-nothing architecture:
  - Provides a smooth path for scaling both throughput and data volume
  - Designed for use on commodity computing resources, including "bare metal" servers and cloud computing

This book helps you get started using VoltDB. The accompanying book. *Using VoltDB*, provides complete information about the usage and features of VoltDB.

1.1. How VoltDB Works

VoltDB is a relational database. You can use VoltDB like any other relational database, defining the schema with SQL data definition language (DDL) statements and issuing ad hoc queries.

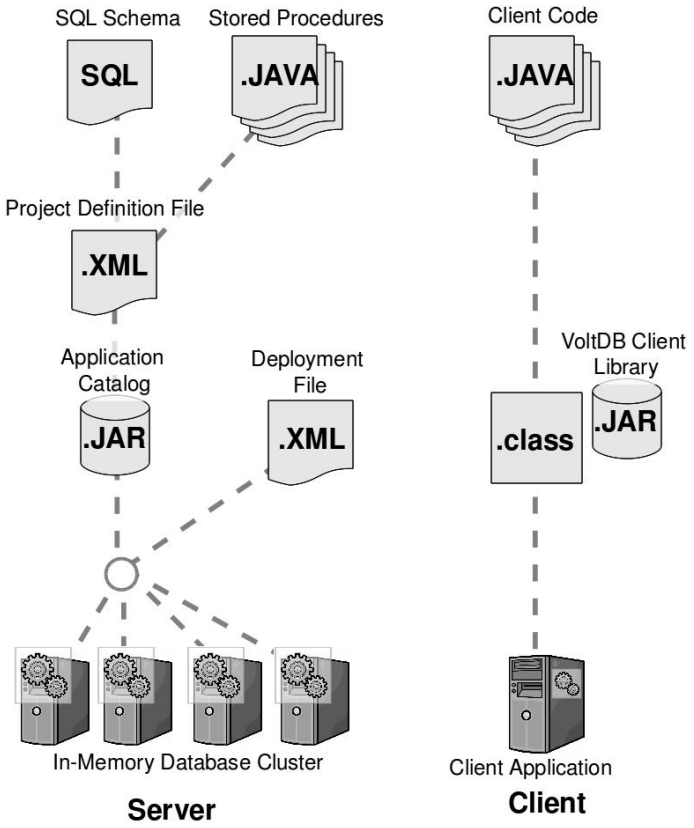
However, to get the most out of your database application, VoltDB provides ways to optimize performance through stored procedures and partitioning.

- Stored procedures are units of work, where multiple SQL queries and application-specific code are pre-compiled and optimized. Each stored procedure is a separate transaction and succeeds or rolls back as a whole.
- Partitioning allows VoltDB to segment both the data and the stored procedures that access that data. By running stored procedures against a specific partition, multiple transactions can be run in parallel, providing almost linear scalability in terms of transaction throughput.

You tell VoltDB what partitions to create and how to access them by compiling your schema and stored procedures into an *application catalog* that is used to create the database at runtime. You can make small changes to the schema by recompiling the catalog and updating it on the fly. Or, if you need to make more significant changes, such as changing the partitioning, you can save the current contents, restart the database with the new catalog and restore the data to the new schema.

VoltDB is designed to accommodate databases from the simplest (a few tables and ad hoc queries) to the most complex (lots of tables, millions of rows, and hundreds of stored procedures). This flexibility is provided by the *project definition file* that helps you compile the right components into your application catalog and the *deployment file* that specifies how the database is distributed at runtime. Figure 1.1 shows how these components fit together — with your client application — to create a complete solution.

Figure 1.1. The Components of a VoltDB Application



- The database schema defines the layout of the database. VoltDB uses standard SQL DDL syntax for defining the database schema.
- The stored procedures define frequent, repeatable transactions. The stored procedures are defined using Java and SQL.
- The VoltDB project file specifies the location of the schema and the stored procedures. This file is used to compile the resulting VoltDB application catalog.
- The application catalog is combined with information about the configuration of the cluster in the deployment file to create the database instance at runtime.
- The client application uses ad hoc queries and stored procedures to perform database requests, using the methods defined in the VoltDB client library.

Chapter 3, *Hello, World!* describes how to create a simple VoltDB application using the components described above.

Chapter 2. Installing VoltDB

VoltDB is available as both pre-built distributions and as source code. The following sections explain how to obtain and install the VoltDB software depending upon your specific configuration and requirements:

- Section 2.1 describes the operating system and software requirements for developing and running VoltDB applications.
- Section 2.2 explains how to install VoltDB from the distribution kit.
- Section 2.2.1 explains how to update an existing VoltDB installation.
- Section 2.2.2 explains how to do a system-wide installation on Ubuntu and other Debian-based Linux systems.
- Section 2.2.3 explains how to build a distribution kit from the VoltDB source code.

2.1. Operating System and Software Requirements

The following are the requirements for developing and running VoltDB applications.

Table 2.1. Operating System and Software Requirements

Operating System	VoltDB requires a 64-bit Linux-based operating system. Kits are built and qualified on CentOS version 5.6 and Ubuntu versions 10.4 and 10.10. Development builds are also available for Macintosh OSX 10.6[1].
CPU	• Dual core[2] x86_64 processor • 64 bit • 1.6 GHz
Memory	4 Gbytes[3]
Java	Sun JDK 6 update 20 or later
Required Software	NTP[4]
Recommended Software	Ant 1.7 or later Eclipse 3.x (or other Java IDE)

Footnotes:

1. CentOS 5.6 and later and Ubuntu 10.4 and later are the only officially supported operating systems for VoltDB. However, VoltDB is tested on several other POSIX-compliant and Linux-based 64-bit operating systems, including Macintosh OSX 10.6.
2. Dual core processors are a minimum requirement. Four or eight physical cores are recommended for optimal performance.
3. Memory requirements are very specific to the storage needs of the application and the number of nodes in the cluster. However, 4 Gigabytes should be considered a minimum configuration.

4. NTP minimizes time differences between nodes in a database cluster, which is critical for VoltDB. All nodes of the cluster should be configured to synchronize against the same NTP server. Using a single local NTP server is recommended, but not required.

2.2. Installing VoltDB

VoltDB is distributed as a compressed tar archive for each of the supported platforms. The best way to install VoltDB is to unpack the distribution kit as a folder in the home directory of your personal account, like so:

```
$ tar -zxvf voltdb-2.8.1.tar.gz -C $HOME/
```

Installing into your personal directory gives you full access to the software and is most useful for development.

If you are installing VoltDB on a production server where the database will be run, you may want to install the software into a standard system location so that the database cluster can be started with the same commands on all nodes. The following shell commands install the VoltDB software in the folder `/opt/voltdb`:

```
$ sudo tar -zxvf voltdb-2.8.1.tar.gz -C /opt
$ cd /opt
$ sudo mv voltdb-2.8.1 voltdb
```

Note that installing as root using the sudo command makes the installation folders read-only for non-privileged accounts. Which is why installing in $HOME is recommended for running the sample applications and other development activities.

2.2.1. Upgrading an Existing VoltDB Installation

If you are upgrading an existing installation of VoltDB, you have two choices:

- You can unpack the new version as a separate installation. VoltDB does this by default, since the tar file contains the version number in the folder name. Note that if you do install new versions alongside an existing installation, any existing Ant build files or shell scripts you have for building and running VoltDB applications will continue to use the older version.

- You can replace your existing installation with the new version. To do this, you need to delete the folder with your current installation and then follow the instructions for unpacking the new kit. For example, the following shell commands unpack the new version under the user's home directory, delete an old installation, and replace it:

  ```
  $ tar -zxvf voltdb-2.8.1.tar.gz -C $HOME
  $ cd $HOME
  $ rm -R voltdb/
  $ mv voltdb-2.8.1 voltdb
  ```

2.2.2. Performing a System-Wide Installation on Ubuntu

If you plan on using VoltDB on Ubuntu or another Debian-based Linux system, there is a Debian package available to simplify the installation process. Using the Debian package installs VoltDB in the system directories, making VoltDB available to all users of the system without them having to individually configure their PATH variable.

To install the Debian package, download the package from the VoltDB web site. Then, from an account with root access issue the following command:

```
$ sudo dpkg -i voltdb_2.8.1_amd64.deb
```

The advantages of using the Debian install package are:

- The installation is completed in a single command. No additional set up is required.
- VoltDB becomes available to all system users.
- Upgrades are written to the same location. You do not need to modify your application scripts or move files after each upgrade.

However, there are a few changes to behavior that you should be aware of if you install VoltDB using the Debian package:

- The VoltDB libraries are installed in /usr/lib/voltdb. When compiling stored procedures, you must include this location in your Java classpath.
- The sample applications are installed into the directory /usr/share/voltdb/samples/. Because this is a system directory, users cannot run the samples directly in that location. Instead, first copy the folder containing the sample application you want to run and paste a copy into your home directory structure. Then run the sample from your copy. For example:

```
$ cp -r /usr/share/voltdb/samples/voter ~/
$ cd ~/voter
$ ./run.sh
```

2.2.3. Building a New VoltDB Distribution Kit

If you want to build the VoltDB software from source (for example, if you want to test recent development changes), you must first fetch the VoltDB source files. The VoltDB sources are stored in a GitHub repository accessible from the VoltDB community web site.

The VoltDB sources are designed to build and run on 64-bit Linux-based or 64-bit Macintosh platforms. However, the build process has not been tested on all possible configurations. Attempts to build the sources on other operating systems may require changes to the build files and possibly to the sources as well.

Once you obtain the sources, use Ant to build a new distribution kit for the current platform:

```
$ ant dist
```

The resulting distribution kit is created as `obj/release/volt-n.n.nn.tar.gz` where *n.n.nn* identifies the current version and build numbers. Use this file to install VoltDB according to the instructions in Section 2.2, “Installing VoltDB”.

2.3. Setting Up Your Environment

VoltDB comes with shell command scripts that simplify the process of developing and deploying VoltDB applications. These scripts are in the /bin folder under the installation root and define short-cut commands for executing many VoltDB actions. To make the commands available to your session, you must include the /bin directory as part your PATH environment variable.

You can add the `/bin` directory to your PATH variable by redefining PATH. For example, the following shell command adds `/bin` to the end of the environment PATH, assuming you installed VoltDB as `/voltdb-n.n` in your $HOME directory:

```
$ export PATH="$PATH:$HOME/voltdb-n.n/bin"
```

To avoid having to redefine PATH every time you create a new session, you can add the preceding command to your shell login script. For example, if you are using the bash shell, you would add the preceding command to the `$HOME/.bashrc` file.

Chapter 3. Hello, World!

To understand the basics of how VoltDB works, it is useful to take a look at a simple example application. Programming tools traditionally use "Hello, World" as their example, so we shall as well.

> **Note**
>
> There are many tools, such as integrated development environments (IDEs) and language-sensitive editors, that can make your development effort easier. However, to keep the example simple, we will not assume any additional tools beyond VoltDB, Java, and a text editor.

At its most basic, VoltDB is a database and stores and retrieves data from database tables. So for our Hello World application we will store the words "hello" and "world" in the database and then retrieve them. To make it a little more interesting, we will store the two words in multiple languages in separate fields in the database table. The language will be used as the primary key for the table.

- Section 3.1 explains how to set up a working environment for the tutorial.
- Section 3.2 explains how to create the SQL DDL file that defines the database schema.
- Section 3.3 explains how to partition your database tables.
- Section 3.4 and Section 3.5 explain how to write the Java files that define the stored procedures the application will use.
- Section 3.6 explains how to write the Java client application that will interact with the database.
- Section 3.7 explains how to create the XML file that describes the project.
- Section 3.8 through Section 3.10 explain how to build and run the completed Hello World Application.

This document walks you through the exercise of creating each of the source files you need for the Hello World application and explains what they do. You can find a complete listing of the sources files in Appendix A, *The Completed Hello World Application*, as well as online in the `doc/tutorials` folder after you install the VoltDB software.

3.1. Setting Up the Environment

As with any application, it is always best to start with a clean slate. Create a new directory to use as a workspace and set default to that directory:

```
$ mkdir helloworld
$ cd helloworld
```

We will use this directory for storing all of the files we create for the Hello World application.

3.2. Defining the Database Schema

The VoltDB database schema is defined using standard SQL. For our sample, we create a schema including one table with three fields. Open your text editor and create the file `helloworld.sql` including the following code:

```
CREATE TABLE HELLOWORLD (
   HELLO VARCHAR(15),
   WORLD VARCHAR(15),
   DIALECT VARCHAR(15) NOT NULL,
   PRIMARY KEY (DIALECT)
);
```

3.3. Partitioning Database Tables

Hello world is a very simple example with a limited set of data, so it does not require partitioning. However, partitioning is critical to the performance and scalability of VoltDB applications. So it is a good idea to learn to use partitioning early on.

When VoltDB partitions a database table, it partitions both the content and the processing that accesses it. The partition in which a row ends up depends on the value of one of the row's columns, called the partitioning column.

To optimize the performance of your application, you should choose a partitioning column that uniquely identifies the rows that are being accessed during each transaction. For the hello world application, records are accessed according to the language. So we want to define DIALECT as the partitioning column.

You define how a table is partitioned using the PARTITION TABLE statement in the database schema. Add the following statement to your `helloworld.sql` file after the definition of the helloworld table:

```
PARTITION TABLE HELLOWORLD ON COLUMN DIALECT;
```

3.4. Writing the Stored Procedure for Inserting Records

We need two stored procedures: one to load records into the database and one to retrieve a matched pair based on the language specified. Stored procedures are written in Java with calls to special VoltDB classes that let you execute the stored procedures and handle the return values.

We create the stored procedure to load the data first. Start a text editor to create a new Java class file called `Insert.java`.

Your VoltDB stored procedures must start by importing the appropriate VoltDB libraries. Type (or cut and paste) the following statements into `Insert.java`:

```
import org.voltdb.*;

@ProcInfo(
    partitionInfo = "HELLOWORLD.DIALECT: 2",
    singlePartition = true
)
```

Line 1 imports the VoltDB libraries. Lines 3-6 provide additional information VoltDB needs to determine where the stored procedure should execute. Of particular note, partitionInfo is used to identify the table being accessed (HELLOWORLD), the column that VoltDB uses to partition the table (DIALECT), and the parameter value used to locate the correct partition (in this case, "2", which indicates the third argument to the stored procedure since the count is zero-based). VoltDB uses this information to optimize the structure and partitioning of the resulting database catalog.

Next, start a Java class using the name of the stored procedure. In this case, the class name is Insert:

```
public class Insert extends VoltProcedure {
```

Note that the class extends the prototype VoltProcedure, which provides additional functions that are used when writing the body of the stored procedure to execute database statements and handle the return values.

The stored procedure itself consists of two main statements: the definition of an SQL statement template (using question marks where values will be filled in later) and a method to actually execute the procedure. Type the following code into your file insert.java:

```
  public final SQLStmt sql = new SQLStmt(
      "INSERT INTO HELLOWORLD VALUES (?, ?, ?);"
  );

  public VoltTable[] run( String hello,
                          String world,
                          String language)
      throws VoltAbortException {
```

The method contains the steps of the stored procedure. In this case, there is just one step: inserting a record into the HELLOWORLD table. This is done in two parts. First the SQL statement is put into a queue, specifying the necessary data to complete the statement template. (In this case, the variables *hello*, *world*, and *language* replace the question marks in the template). Then the queue is executed. Add the following statements to your file:

```
      voltQueueSQL( sql, hello, world, language );
      voltExecuteSQL();
```

Your procedure is complete. Normally, you want to evaluate the return value from the SQL statement. But for now you can assume the statement succeeds. Return a null and complete the method and class by adding closing braces.

```
      return null;
   }
}
```

Save the file and close the text editor. The completed procedure file is shown in Example A.4, “Insert.java”.

3.5. Writing the Stored Procedure For Retrieving Records

The second stored procedure retrieves the words for "hello" and "world" based on the language specified. This procedure starts very much like Insert.java. So start the text editor to create a new Java class file called `Select.java` and add the code to import the VoltDB libraries, specify the partition, and begin a class named Select. Note that the code is almost identical, except for the class name and the parameter being used as the partitioning value, because we are accessing the same table and partition.

```
import org.voltdb.*;

@ProcInfo(
    partitionInfo = "HELLOWORLD.DIALECT: 0",
    singlePartition = true
)

public class Select extends VoltProcedure {
```

Next, we need to write the SQL statement that fetches the necessary data. In this case we want to find a record based on the field DIALECT and return the values for HELLO and WORLD. So the statement looks like the following. Note that we again use a question mark for the specific value of DIALECT, since that will be passed as a parameter to the stored procedure. We also provide the appropriate arguments when we add the SQL statement to the queue.

```
  public final SQLStmt sql = new SQLStmt(
      "SELECT HELLO, WORLD FROM HELLOWORLD " +
      " WHERE DIALECT = ?;"
  );

  public VoltTable[] run( String language)
      throws VoltAbortException {
          voltQueueSQL( sql, language );
          return voltExecuteSQL();
      }
}
```

The major difference between the insert and the select procedures is that in select we must find a way to return the values fetched from the database. VoltDB provides a utility to help you do this called the VoltTable. VoltTable is an array used as the return value of all procedure calls. The Select procedure returns the VoltTable array containing the rows returned by the Select statement(s) directly to the calling client application.

Your procedure is complete. Save the file and close the text editor. The completed procedure is shown in Example A.5, “Select.java”.

3.6. Writing the Client Application

Now that you have defined your database schema and written the stored procedures, you can write the client code to call the stored procedures. Our client application is very simple: it calls the Insert procedure repeatedly to load the database, then it calls Select to retrieve and display the hello world message in the language of your choice.

Again, start your text editor and create a new file called `Client.java`. Start your client code by importing the necessary system and VoltDB libraries and starting the client class.

Note

For serious programming, you will want to organize your Java code into packages. For example, keeping your stored procedures in one package and your client classes in another. When you do this, you need to import the procedures package as well. But to keep this sample simple, we are keeping all code in a single directory,

```
import org.voltdb.*;
import org.voltdb.client.*;

public class Client {
```

The first thing your application needs to do is create a client connection to the database. This is done by creating a org.voltdb.client.Client (using the ClientFactory) and opening the connection, like so:

```
public static void main(String[] args) throws Exception {

    /*
     * Instantiate a client and connect to the database.
     */
    org.voltdb.client.Client myApp;
    myApp = ClientFactory.createClient();
    myApp.createConnection("localhost");
```

Note that when you create the connection, you specify the network node where the database server is running. If the database is running on a cluster, you can specify any one of the cluster nodes when you create a connection. In fact, you can create connections to multiple nodes in the cluster to help distribute the work by invoking the createConnection method once for each node.

But for now, we are running the database and the client locally. So you can specify the node as "localhost" when you create the connection for your Hello World application.

Once you open the database connection, you are ready to call the stored procedures. To call a VoltDB stored procedure, use the volt.Client.callProcedure method. For our example, we first call the Insert procedure several times, passing in the three arguments it requires.

```
/*
 * Load the database.
 */
myApp.callProcedure("Insert", "Hello", "World", "English");
myApp.callProcedure("Insert", "Bonjour", "Monde", "French");
myApp.callProcedure("Insert", "Hola", "Mundo", "Spanish");
myApp.callProcedure("Insert", "Hej", "Verden", "Danish");
myApp.callProcedure("Insert", "Ciao", "Mondo", "Italian");
```

Next we call the Select procedure to retrieve the message in a language of your choice. For the example code, we will use Spanish, but you can choose any of the languages stored by the Insert procedure.

To retrieve the message, we not only want to call the procedure, we also want to decipher the results that are in the VoltTable array returned by the stored procedure. The following code segment shows how this is done. First we check to make sure we have valid results by ensuring that the status from the procedure call is SUCCESS. Then we get the actual results from the ClientResponse object with the GetResults method. To evaluate the results, we then fetch the first row of the first VoltTable in the array. Finally, we use the getString method to fetch the individual fields from the returned table row and output them to standard output.

Type this into your `Client.java` file.

```
    /*
     * Retrieve the message.
     */
    final ClientResponse response = myApp.callProcedure("Select",
                                                        "Spanish");
    if (response.getStatus() != ClientResponse.SUCCESS){
        System.err.println(response.getStatusString());
        System.exit(-1);
    }

    final VoltTable results[] = response.getResults();
    if (results.length == 0 || results[0].getRowCount() != 1) {
```

```
            System.out.printf("I can't say Hello in that language.\n");
            System.exit(-1);
        }

        VoltTable resultTable = results[0];
        VoltTableRow row = resultTable.fetchRow(0);
        System.out.printf("%s, %s!\n", row.getString("hello"),
                                       row.getString("world"));
    }
}
```

Your client application is now done. See Example A.6, “Client.java” for a complete listing of the `client.java` source code.

3.7. Creating the Project Definition

Now that you have written the source code, you need to create the project definition file that VoltDB uses to create the database catalog. The project definition file is an XML file that identifies the database schema and stored procedures.

Open your text editor and create a file called `project.xml`. Since the file is XML, you must include the XML declaration as the first line, followed by the root element, project:

```
<?xml version="1.0"?>
<project>
```

Next you define the database you are creating. Within the <database> element, you include tags that identify the components you created earlier, the database schema and the stored procedures. Enter the following code into your project.xml file:

```
    <database>
       <schemas>
           <schema path='helloworld.sql' />
       </schemas>
       <procedures>
           <procedure class='Insert' />
           <procedure class='Select' />
       </procedures>
```

Note that you specify the path to the schema as a file path, but the stored procedures are classes so are specified by name only (not a file path).

Finish your definition file by ending the database and project elements (as below) and save the resulting file.

```
    </database>
</project>
```

3.8. Building the Hello World Application

Now you are ready to build your application. The process for building VoltDB applications is as follows:

1. Compile the Java source files for the client application and stored procedures

2. Compile the database definitions into an application catalog

Note

Compiling Java source files and building the application catalog involve dependencies on several VoltDB modules. The easiest way to build a VoltDB application is to use a build procedure such as Ant, make, or a shell script to automate the process.

A shell script, `run.sh`, is included with the Hello World source files online for those who want to use it. However, to familiarize you with the components that make up a VoltDB application, the following sections describe the individual commands that are used to build and run VoltDB applications that are normally automated.

To compile the Java source files, you must make sure your Java classpath includes your working directory as well as the VoltDB Jar files created when you installed VoltDB. For the purposes of this example, we assume VoltDB is installed in your account's home directory as $HOME/voltdb. To specify a classpath that includes both your helloworld directory and the VoltDB jar files, define the environment variable CLASSPATH, as in the following example. If you are no longer in your working directory, you will want to set default to that directory now as well:

```
$ cd $HOME/helloworld
$ CLASSPATH="./:/opt/voltdb/lib/*:/opt/voltdb/voltdb/*"
$ export CLASSPATH
```

Now compile the client application and stored procedure source files using the javac command:

```
$ javac Client.java
$ javac Insert.java
$ javac Select.java
```

Finally, you compile the database schema and project definition file to create the application catalog. To do this, you use the `voltcompiler` command specifying the path to your stored procedure class files, the project definition file (project.xml), and the name of the catalog file to create as output. The command to compile the application catalog for the hello world example is as follows:

```
$ voltcompiler ./ project.xml helloworld.jar
```

3.9. Defining Your System Configuration

When you compile the Java sources and create the application catalog, you are almost ready to create your database and run the application. The last step is to specify exactly how you want the database to be configured and the hardware it will run on. You do this in the VoltDB deployment file.

The deployment file is an XML file, just like the project definition file. Except that rather than describing the structure of the database, the deployment file describes the size and configuration of the cluster that the database will run on, including the following information:

- The number of servers that will be used
- The number of sites per server

In the deployment file, the root element is the `<deployment>` tag, and the basic information is specified as attributes of the `<cluster>` element. For the purposes of the Hello World application, we will define a cluster using just the current machine (that is, one node) and two partitions. Use your text editor to create a new file called deployment.xml and enter the following text:

```
<?xml version="1.0"?>
```

```
<deployment>
   <cluster hostcount="1"
            sitesperhost="2"
   />
   <httpd enabled="true">
      <jsonapi enabled="true" />
   </httpd>
</deployment>
```

The deployment file can be used to enable and configure several other features of a VoltDB database. For example, in the deployment file you just created, it enables the HTTP and JSON interfaces. Other features that you can control with the deployment file are described in more detail in *Using VoltDB*. But for now, we will focus on the basics. Save and close your file and you are ready to run your application.

3.10. Running the Hello World Application

Running VoltDB applications consists of two separate actions: starting the database server and running the client application.

It is easiest to execute these two actions in separate processes. Create two terminal sessions and set default to the working directory you created at the beginning of this tutorial. If you created a new process, you may need to redefine CLASSPATH. For example:

```
$ cd $HOME/helloworld
$ CLASSPATH="./:/opt/voltdb/lib/*:/opt/voltdb/voltdb/*"
$ export CLASSPATH
```

In one terminal session, start your database server. You do this by invoking the `voltdb` command and specifying:

- The application catalog
- The deployment file
- The host node of the cluster

When running a cluster with multiple machines, the host node provides startup services for the cluster as a whole, including coordinating the cluster configuration and hosting the application catalog. Once startup is complete, the host's role is complete and it becomes a peer of all the other nodes.

Since you are running the sample locally, the host (and only) node in the cluster is localhost.

Finally, if you are using the VoltDB Enterprise Edition, you must specify the location of your VoltDB license file. Since the license file is ignored in the Community Edition, you can include the license argument whichever version you are using. For example:

```
$ voltdb catalog helloworld.jar   \
         deployment deployment.xml   \
         host localhost \
         license /opt/voltdb/voltdb/license.xml
```

VoltDB will display information about the catalog and start the database.

Now switch to your second terminal session and start the client application, which in this case is Client. The client will start and display the words "hello world" in the language you specified in the source file:

```
$ java Client
Hola, Mundo!
$
```

Congratulations! You have completed your first VoltDB application. You will find the finished source files in Appendix A, *The Completed Hello World Application*.

3.11. Next Steps

Now that you are done, if you want to test yourself, go back into the source files and make some changes. Things you might try are:

- Change the language that the client application uses to fetch output from the Select stored procedure.
- Add the words "Hello" and "World" in another language (say, German) by adding another call to the Insert stored procedure.
- Add another field, COLOR, to the table in the SQL DDL and update the stored procedures and client application appropriately to store and retrieve the phrase "Hello, green world!" in multiple languages.

Note that although the Hello World example works and demonstrates how to perform basic database functions using VoltDB, it does not show off VoltDB's best features: throughput and scalability. To better understand how to design applications that take advantage of VoltDB key benefits, see the Hello World Revisited tutorial in the *VoltDB Performance Guide*.

Chapter 4. Running the VoltDB Example Applications

Besides the Hello World application, VoltDB comes with three example applications that demonstrate the capabilities and performance of VoltDB. The example applications include:

- Voltcache — demonstrates how to use VoltDB as an in-memory cache. Caches, such as memcached, are a popular technique to provide rapid access to the most frequently used content from a larger, slower database. They improve performance on read access to popular content. However, caches do little to improve write performance. Using VoltDB as a cache provides improvement to both read and write performance, while adding transactional consistency and durability.

- Voltkv — demonstrates how to create a Key-Value Store using VoltDB. Key-Value Stores are popular for web applications because of their simplicity, performance, and scalability. Using VoltDB to create a Key-Value Store adds transactions, durability, and automatic partitioning without sacrificing the other characteristics.

- Voter — simulates a telephone voting application, similar to what you might find associated with a TV talent show or other popularity contest. The Voter application comes with client applications written to several alternate VoltDB interfaces, including JDBC and JSON.

The example applications are packaged with the software. After installing VoltDB, you can find the examples in the examples folder. For instance, if you install VoltDB in the folder `/opt/voltdb`, the examples are in the folder `/opt/voltdb/examples`. If you install VoltDB locally in your account's $HOME directory, the examples are in the folder `$HOME/voltdb-n.n.nn/examples` where *n.n.nn* is the version number of VoltDB.

All examples include a shell script (run.sh) for building and running the server and a client in the default configuration. Additionally, each example folder contains a README that provides specifics about the application, configuration options, and how to run that application.

4.1. Building and Running the Example Applications

The examples come with a shell script to automate the building and running of the application. To build an example, set default to the subdirectory for that example and execute the `run.sh` script using the argument *catalog*. For example, the following commands build the voter example, assuming you start from the examples folder:

```
$ cd voter
$ ./run.sh catalog
```

Once you build the example, you can run it using the script argument *server* and *client*. Note that there are two separate processes that you need to run. Start the server process first, using the **./run.sh server** command, then open a new terminal window and use the **./run.sh client** command to start the client application.

The server, as its name implies, acts as the database server. The client performs the application logic and displays any messages appropriate to the application. (Note that using the shell script without an argument both compiles the application catalog and starts the server.)

4.2. Running the Example Applications on a Cluster

Often the next step after seeing that you can run the sample applications on your local machine is to move to a cluster to fully experience the scalable nature of VoltDB performance. This is also easy to do.

To run a VoltDB database on a cluster, you run the server process on all nodes of the cluster making only three changes to the startup procedure:

1. Specify the number of nodes in the cluster `deployment.xml` file by changing the `hostcount` attribute from "1" to the number of servers you plan to use. For example, assuming you plan on using three nodes (ServerA, ServerB, and ServerC), change the deployment file as follows:

   ```
   <cluster hostcount="3" sitesperhost="2" kfactor="0" />
   ```

2. Select one of the nodes as the "host" and specify its address on the command line when starting each of the server processes. Note that all nodes specify the same host. So, if we select ServerA as the host from our previous example, you include **host ServerA** in the command line when starting all three nodes.

   You can do this by editing the run.sh script that comes with the sample application and changing the definition of the environment variable LEADER at the beginning of the file, like so:

   ```
   VOLTDB="../../bin/voltdb"
   VOLTCOMPILER="../../bin/voltcompiler"
   LICENSE="../../voltdb/license.xml"
   LEADER="ServerA"
   ```

3. Finally, change the run.sh so the client application makes connections to all of the nodes in the cluster rather than just one. This helps maximize the throughput by avoiding any network bottlenecks. You specify the nodes to connect to as an argument to the client application. Edit the run.sh, then find and replace all references to the `--servers` argument with a list of server nodes. For example:

   ```
   --display-interval=5 \
   --duration=120 \
   --servers=ServerA,ServerB,ServerC \
   --port=21212 \
   --pool-size=100000 \
   ```

4.3. Using the Example Applications to Learn VoltDB

The examples can be very helpful in learning how to develop applications in VoltDB. In particular:

- **Performance** — The primary focus of VoltDB is on best-in-class throughput performance. All of the samples in the examples folder are designed to demonstrate outstanding performance. These samples use convenience classes and methods to produce an application that integrates benchmarking and automated throttling, which helps optimize throughput and latency. See the `README` file for each application for additional information on running the application in different configurations.

- **Build Process** — The run.sh file for each example shows how the application is constructed from its various parts. If you plan on using shell scripts to build your applications, you can use the examples as a template for writing your own script file.

- **Sample Code** — Each example comes complete with source code for the stored procedures, client application, database schema, and project definition file. In addition to the example applications, the doc/tutorials folder contains source code for other examples (Hello World and Auction) that demonstrate use of the basic VoltDB Java API.

Chapter 5. Using VoltDB with Eclipse

As you can see from the Hello World tutorial and the sample applications distributed with VoltDB, there are several components to a VoltDB application. Managing the source files, their location, and the details of how to build them effectively is not a trivial exercise. This is why VoltDB provides a script to help generate a template application, including the appropriate Ant build file to manage these details for you.

In addition to managing the build process, it is often helpful to use an integrated development environment (IDE) to manage the editing of the source files as well. IDEs provide, as the name implies, a complete environment to assist in software development, including "smart" editors that validate code, resolve errors, and execute test builds through a simplified interface.

Eclipse is one of the most common Java IDEs available. (Eclipse supports many other programming languages as well, so is useful beyond just its application to VoltDB.) The following sections explain how to set up a new VoltDB project within Eclipse. These sections are not intended as a tutorial in Eclipse; they assume you are already familiar with software programming practices, the Java programming language, and Eclipse (or a similar IDE).

There are other Java IDEs available, such as NetBeans, which operate in much the same way as Eclipse. The choice of which IDE to use is up to you, the user. But for the purpose of this guide, we will describe how to set up Eclipse for use with VoltDB and leave the set up of other IDEs as an exercise for those readers who prefer the alternatives.

5.1. Installing Eclipse

Eclipse is available as an installation package for most Linux-based platforms. So you can often use your operating system package manager (such as Synaptic) to install Eclipse. If not, you can find installation kits and instructions on the Eclipse web site: `http://www.eclipse.org/`.

Eclipse 3.0 and later comes bundled with Ant, which you need to manage the build process for your application. So you do not need to install Ant separately. However, it is a good idea to make sure Eclipse is using the correct version of the Java runtime environment (JRE). Choose **Preferences...** from the **Window** menu and look under **Java > installed JREs** to see which JRE Eclipse is using. Eclipse must be using Java 1.6 or later to operate properly for VoltDB applications,

5.2. Managing VoltDB Development With Eclipse and Ant

It is possible to start a new VoltDB project from scratch within Eclipse. However, it is often easiest to import an existing project to ensure the build environment is set up correctly.

Ant is a software build automation tool designed specifically for Java applications that integrates well with Eclipse. If you use Ant as your automation tool, the easiest way to create a VoltDB project in Eclipse is to import an existing project, using your current Ant build file as the target of the import. If you do not have an existing project, you can create the preliminary project structure, including folders and stub files, using Ant to describe the structure. Then import the Ant build file into Eclipse.

5.2.1. Importing the Ant Build File into Eclipse

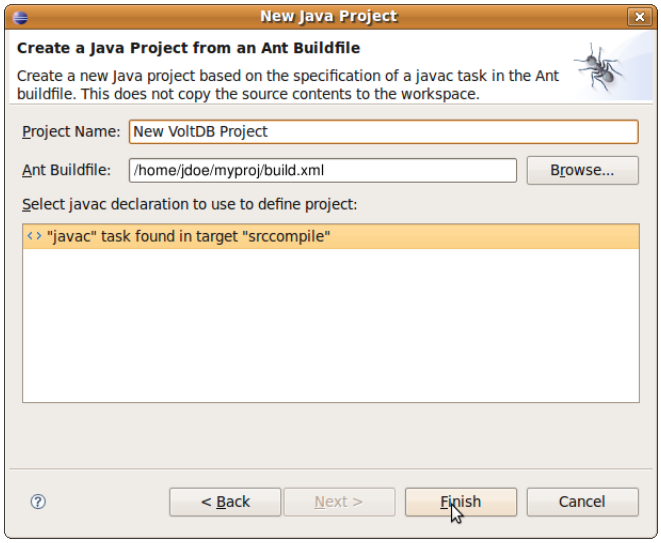
To import a project into Eclipse using Ant, do the following:

1. Start Eclipse.

2. Select **New... Project...** from the File menu.

3. In the **New Project** dialog box, expand **Java** in the list of wizards and select **Java Project from Existing Ant Buildfile** and click **Next**.

4. In the next dialog box, enter the `build.xml` from the directory where you created your template project as the Ant build file, and enter a descriptive project name. Then click **Finish**.



Eclipse creates a new project, leaving the source files where they are in your template project directory. You can now edit, build, and debug the source files from within Eclipse.

5.2.2. Managing Your Project in Eclipse

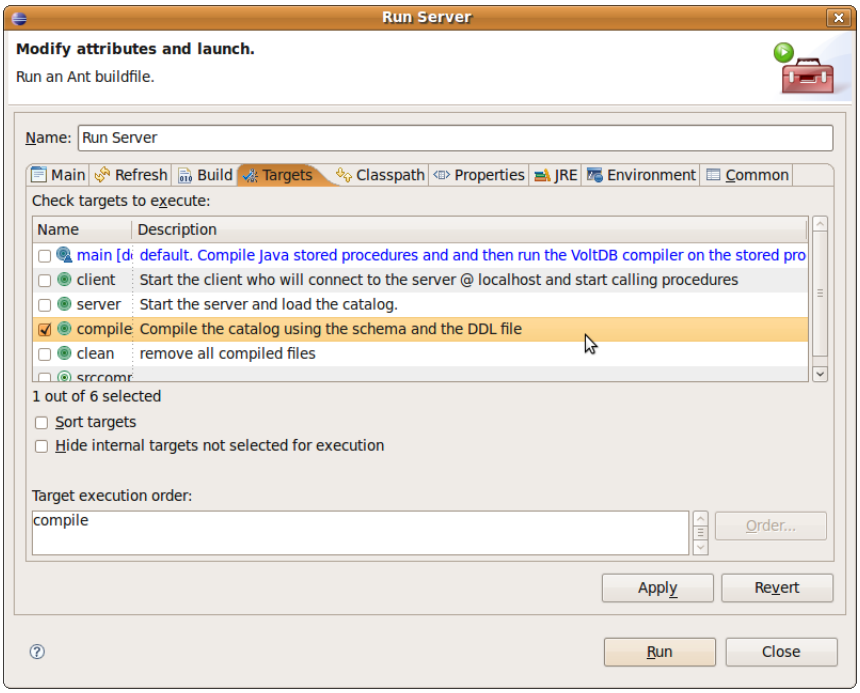
Once you import the project into Eclipse, you can manage, edit, and build your Java sources within Eclipse. Expanding the project name in the navigation window, you can double click on Java source files to open them in the editor. Selecting **Build** from the **Project** menu lets you compile the sources into class files.

Note that Eclipse doesn't know how to compile the VoltDB catalog directly. However, this task should still be a target within the Ant build file. To compile the catalog from within Eclipse:

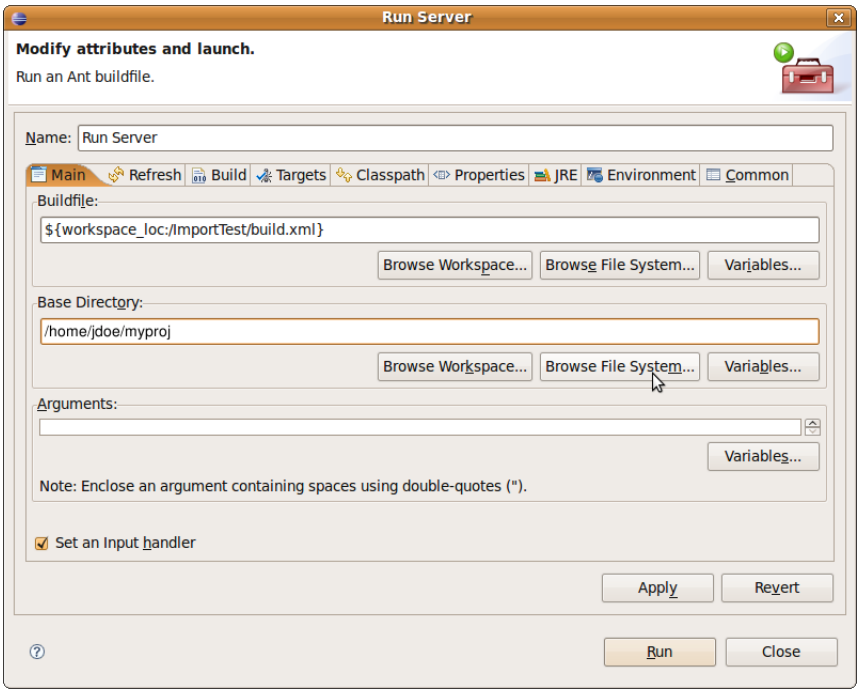
1. Expand the project in the Navigation window until you can see the `build.xml` file in the project contents list.

2. Right-click on the `build.xml` file and select **Run As... Ant Build...** (item 2 from the popup menu).

3. In the resulting dialog box, Make sure the **Targets** tab is selected and check the appropriate target within the build file for compiling both the stored procedures and the VoltDB application catalog.



4. Switch to the **Main** tab and use the **Browse File System...** button to change the Base Directory to point to the root directory of your template project.



5. Click **Apply** and then **Run**. (After applying your changes, you should not have to perform steps #3 and 4 again.)

Finally, there are two components of the VoltDB project that Eclipse does not recognize from the Ant build file. These are the project definition (XML) file and the database schema (SQL DDL) file. To manage the project properly, you should import these files as well so they are visible within the project package and you can edit them by double-clicking on them.

To import the files:

1. Right click on the project name in the navigation window and select **New... File**.
2. Make sure the correct project is selected in the selection box, then click on the **Advanced** >> button.
3. Check the box for **link to file in the file system**, browse to your template project and select the file you want to import. Click **Finish**.

Once the files are imported, you can double-click on them in the navigation window to open them for editing.

Note

Eclipse can perform smart editing on XML and DDL files. However, Eclipse does not recognize .sql as a file type for database schemas and will start the default text editor to edit such files. If, before importing, you rename the schema to have the file type .ddl, Eclipse will recognize it and provide its own editor. However, you will need to edit the project.xml file as well to make the corresponding change to the project definition.

5.2.3. Running VoltDB Applications in Eclipse

Eclipse helps you compile your source files and the VoltDB catalog, as described in the preceding section. Once you have successfully built your application, you can run both the client and the server from within Eclipse as well.

- To run the application server, right-click on the `build.xml` file, choose **Run As...** and specify the appropriate target for starting the server process. Eclipse will start the server and display any messages from the server process in the console window.
- To run the application client, select **Run...** from the **Run** menu and specify your client application class as the Java application to run. Again, Eclipse displays messages from the client process in the console window.

There are limitations to running VoltDB applications in Eclipse. First, Eclipse can only run the client and server if the project is defined as a single server running on localhost. Also, the messages from one process (such as client) will replace those of the other (server) in the console window. However, running the application from within Eclipse can still be a very effective tool for testing and debugging the completed application locally before a full-scale build and test.

Appendix A. The Completed Hello World Application

The following examples contain the source code for the completed Hello World application.

Example A.1. project.xml

```
<?xml version="1.0"?>
<project>
    <database>
       <schemas>
           <schema path='helloworld.sql' />
       </schemas>
       <procedures>
           <procedure class='Insert' />
           <procedure class='Select' />
       </procedures>
    </database>
</project>
```

Example A.2. deployment.xml

```
<?xml version="1.0"?>
<deployment>
   <cluster hostcount="1"
            sitesperhost="2"
   />
   <httpd enabled="true">
      <jsonapi enabled="true" />
   </httpd>
</deployment>
```

Example A.3. helloworld.sql

```
CREATE TABLE HELLOWORLD (
   HELLO VARCHAR(15),
   WORLD VARCHAR(15),
   DIALECT VARCHAR(15) NOT NULL,
   PRIMARY KEY (DIALECT)
);

PARTITION TABLE HELLOWORLD ON COLUMN DIALECT;
```

Example A.4. Insert.java

```
import org.voltdb.*;

@ProcInfo(
    partitionInfo = "HELLOWORLD.DIALECT: 2",
    singlePartition = true
)

public class Insert extends VoltProcedure {

  public final SQLStmt sql = new SQLStmt(
      "INSERT INTO HELLOWORLD VALUES (?, ?, ?);"
  );

  public VoltTable[] run( String hello,
                          String world,
                          String language)
      throws VoltAbortException {
          voltQueueSQL( sql, hello, world, language );
          voltExecuteSQL();
          return null;
      }
}
```

Example A.5. Select.java

```
import org.voltdb.*;

@ProcInfo(
    partitionInfo = "HELLOWORLD.DIALECT: 0",
    singlePartition = true
)

public class Select extends VoltProcedure {

  public final SQLStmt sql = new SQLStmt(
      "SELECT HELLO, WORLD FROM HELLOWORLD " +
      " WHERE DIALECT = ?;"
  );

  public VoltTable[] run( String language)
      throws VoltAbortException {
          voltQueueSQL( sql, language );
          return voltExecuteSQL();
      }
}
```

Example A.6. Client.java

```
import org.voltdb.*;
import org.voltdb.client.*;

public class Client {
```

```
    public static void main(String[] args) throws Exception {

        /*
         * Instantiate a client and connect to the database.
         */
        org.voltdb.client.Client myApp;
        myApp = ClientFactory.createClient();
        myApp.createConnection("localhost");

        /*
         * Load the database.
         */
        myApp.callProcedure("Insert", "Hello", "World", "English");
        myApp.callProcedure("Insert", "Bonjour", "Monde", "French");
        myApp.callProcedure("Insert", "Hola", "Mundo", "Spanish");
        myApp.callProcedure("Insert", "Hej", "Verden", "Danish");
        myApp.callProcedure("Insert", "Ciao", "Mondo", "Italian");

        /*
         * Retrieve the message.
         */
        final ClientResponse response = myApp.callProcedure("Select",
                                                            "Spanish");
        if (response.getStatus() != ClientResponse.SUCCESS){
            System.err.println(response.getStatusString());
            System.exit(-1);
        }

        final VoltTable results[] = response.getResults();
        if (results.length == 0 || results[0].getRowCount() != 1) {
            System.out.printf("I can't say Hello in that language.\n");
            System.exit(-1);
        }

        VoltTable resultTable = results[0];
        VoltTableRow row = resultTable.fetchRow(0);
        System.out.printf("%s, %s!\n", row.getString("hello"),
                                       row.getString("world"));
    }
}
```