Installing and Building Relational DataScript (rds)

Harald Wellmann <HWellmann@harmanbecker.com>

Revision History

Revision 0.1 28 September 2006

Initial version describing the build process.

Revision 0.2 3 December 2006

Installation description added. Updated the Building from Source section.

Revision 0.3 2 February 2007

Corrected Jar files for Ant build. Additions for Windows XP. Corrected SVN repository URL.

Revision 0.4 8 February 2007

Changes for Java 1.6.0. Added -doc option.

Revision 0.5 23 May 2007

Java 1.6.0 is now default. Adapted to rds 0.8. New command line options. Renamed dstools-antlr to rds. Hints on Linux support. Remark on Eclipse dependency.

Revision 0.6 28 Jun 2007

Added new options -ext and -xml.

Revision 0.7 21 Sep 2007

Adapted to rds 0.14. JET is no longer used. Support for Eclipse 3.3 and Linux.

Revision 0.8 29 November 2007

Adapted to rds 0.18. -ext option dropped. -java_e option documented. New section on stand-alone Ant build.

Table of Contents

1. Introduction

1.1. Purpose

This is a short guide for installing and running the DataScript tools, for getting the source from a release package or from Subversion and for building the tools from source.

1.2. Package Overview

This guide relates to the packages available from the `dstools` `[http://dstools.sourceforge.net]` project at Sourceforge. There are the following packages:

- `rds-bin`: Binary distribution of Relational DataScript, our current baseline version for a DataScript parser, including relational extensions and HTML generation.
- `rds-src`: The source package corresponding to `rds-bin`.
- `dstools-bin`: Binary distribution of a DataScript parser and code generator, directly derived from the reference implementation of Godmar Back (obsolete). This was the starting point of the dstools project, which has now been superseded by `rds`.
- `dstools-src`: The source package corresponding to *dstools-bin*.

1.3. Platform Support

The development platform for DataScript is Windows XP, but the packages should run on all platforms supporting a Java SE 1.6.0. To use the relational extensions, you need a JDBC driver for SQLite which depends on a platform-specific native library. `rds` itself, however, does not use native code. For convenience, a JDBC driver and the native libraries for Windows and Linux are included in the `rds-bin` and `rds-source` packages.

This guide uses Windows syntax for command lines and path names, trusting in the ability of Unix users to silently make the necessary changes.

Java versions 1.5.x or earlier are not compatible due to the use of generics and of `java.util.ServiceLoader` in the `rds` implementation. The JAR files provided in the `rds-bin` package are compiled with Java SE 1.6.0.

`rds` also runs under Linux (tested on openSuSE 10.2).

2. Installing and Running

To install the DataScript tools, download the `rds-bin` package and unzip it to an installation directory `%RDS_HOME%`. `rds` is implemented in Java. You need a Java VM 1.6.0 or higher to run it.

`rds.jar` is an executable Java Archive which can be run by

```
java -jar %RDS_HOME%\rds.jar
[-c] [-doc] [-pkg <output package>] [-xml [<file name>]]
[-java_e]
[-out <output path>] [-src <source path>]
<input file>
```

<*input file*> is an absolute or relative file name for the top-level DataScript package to be parsed. If this package contains imports, e.g. `import foo.bar.bla.*`, `rds` will convert this package name to a relative path name and try to read the imported package from `foo\bar\bla.ds`.

The `-src` option defines the root directory for the input file and all imported packages. If this option is missing, the default value is the current working directory. Example: If the source path is `C:\datascript` and the input file is `com\acme\foo.ds`, `rds` will try parsing `C:\datascript\com\acme\foo.ds`. If `foo.ds` contains the declaration `import com.acme.bar.*`, `rds` will try parsing `C:\datascript\com\acme\bar.ds`.

Currently, only one source directory can be specified. A list of directories as in the Java `CLASSPATH` is *not* supported.

Similarly, the `-out` option defines the root directory for the generated Java source files. Specifying `-out C:\java` in our example, we will find the generated code in `C:\java\com\acme\foo` and `C:\java\com\acme\bar`.

The `-pkg` option specifies the Java package name for types that do not belong to a DataScript package. The files will be created in a subdirectory <*output package*> of the output path. Any DataScript source file should contain a package declaration, so this option is rather obsolete.

If the `-doc` option is present, HTML documentation will be generated into a subdirectory `html` of the current working directory.

If the `-xml` option is present, `rds` will dump an XML representation of the syntax tree of all input files to <*output path*>`\datascript.xml`. The default name of this output file can be overridden by supplying a file name argument to the `-xml` option.

If the `-c` option is present, `rds` checks the structure of its internal syntax tree, which may be useful for `rds` devlopers, but certainly not for `rds` users.

If the `-java_e` option is present, the equals() methods generated by the Java extension will throw an exception instead of returning false when the argument is not equal. This is mainly intended as a debugging aid for easier detection of a mismatch in a complex type hierarchy.

The `-ext` option specifies the path for `rds` extension libraries. `rds` will load and execute all extensions in this directory. The default value of this option is `ext`. If you do not run `rds` from `%RDS_HOME%`, you will have to set `-ext` explicitly. Each code generator (for Java, HTML, XML and optionally C++) is implemented as an extension. If `rds` cannot find any extensions, it will not do anything except parsing the input.

The Java code generated by `rds` depends on the classes contained in `rds-runtime.jar`. To use `rds`-generated code in some other project, you will have to

add `rds-runtime.jar` to its class path or to include the classes from this JAR into some other JAR of your project.

3. Building from Source

`rds` supports stand-alone Ant builds or builds within Eclipse 3.3 where Ant cooperates with the Eclipse Java builder. If you know what you are doing, you will be able to build `rds` in other environments, but these two options are the only ones that the authors will document and support.

The following instructions refer to the `rds-src` package. For the `dstools-src` package, some path and target names need to be adapted.

3.1. Stand-alone Ant Build

Using any Subversion client, fetch the `rds` sources to your local workspace. From the root of this workspace, invoke `ant` as follows:

```
ant -lib lib [target]
```

The default Ant target is `jar`. Running this target will build the rds JARs and their dependencies in `build\jar`.

Target `cleanall` cleans the results of the build. Target test.run runs the JUnit test suite for `rds`.

3.2. Eclipse Build

3.2.1. Installing Eclipse

- Install JDK 1.6.0 from `http://java.sun.com`.
- Install Eclipse 3.3 from `http://www.eclipse.org`.
- Start Eclipse and set the proxy options in **Window | Preferences | Install/Update**.
- Select **Window** | **Preferences** | **Java** | **Installed JREs** and make sure that your JDK 1.6.0 VM is listed there.

3.2.2. Installing the Subclipse plugin

Subclipse is a Subversion client plugin for Eclipse. Using this plugin, you can directly access Subversion repositories from Eclipse. This step is recommended if you wish to build rds directly from a given revision in the Subversion repository. Subclipse is not required for building rds from a source package.

- Goto **Help | Software Updates | Find and Install**. Select **Search for new features to install** and click **Next**.

- Click on **New Remote Site...** Enter name `Subclipse` and URL `http://subclipse.tigris.org/update_1.0.x`.
- Select the Subclipse Site and click **Finish**. The Search Results should display a feature named **Subclipse**.
- Select the Subclipse feature and click **Next**. Accept the license terms and click **Next**. Click **Finish**.
- There will be a warning **You are about to install an unsigned feature**. Simply click **Install**.
- You will be prompted to restart the workbench. Click **Yes**.
- Select **Window | Preferences | Team | SVN** and activate **SVN Interface SVNKit (Pure Java)**.
- Select **Window | Open Perspective | Other... | SVN Repository Exploring**.
- Select **Window | Show View | SVN Repository**.
- If you are forced to use a proxy for HTTP and HTTPS, you have to edit a configuration file so that Subversion will use your proxy. Using any text editor, open the file `servers` in the folder `%APPDATA%\Subversion`. (`%APPDATA%` is a Windows environment variable referring to a folder with user-dependent application settings, which translates to something like `C:\Dokumente und Einstellungen\HWellmann\Anwendungsdaten`.) If this folder does not exist, make sure you did not miss any of the preceding steps. The folder gets created when you first open the SVN Repository view.

  Go to the `[global]` section at the end of the file, uncomment and edit the lines `http-proxy-host` and `http-proxy-port` to reflect the proxy settings at your site.

3.2.3. Creating a local project from the Subversion repository

- In Eclipse, go to the SVN Repository view in the SVN Repository Exploring perspective.
- Select **New | Repository Location** from the context menu.
- Fill in the URL `https://dstools.svn.sourceforge.net/svnroot/dstools`. Click **Finish**.
- When prompted for accepting a digital certificate, click **Accept Permanently**.
- Expand the repository tree and select the subnode `trunk/rds`.
- Select **Checkout...** from the context menu of this node.
- Enter a project name. If you are expecting to work on multiple versions in parallel (e.g. trunk and development), make sure to select a meaningful name, e.g.

`rds-trunk`. Click **Finish**.

3.2.4. Repository Structure

Following Subversion conventions, the repository has the following folders:

- `branches`: Development branches for tasks that should not interfer with main-line development on the trunk.
- `tags`: Release tags. To create a release, a given version of a trunk subfolder is simply copied to a new subfolder of the tags folder.
- `trunk`: The main development line.

The trunk has the several subfolders or packages:

- `dstools`: Sources for the `dstools-src` package (obsolete).
- `rds`: Sources for the `rds-src` package.
- `www`: Content of the project homepage [http://dstools.sourceforge.net]. Thanks to a cron job running on the Sourceforge server, any commits to this folder will be visible on the homepage within an hour.

3.2.5. Setting up the project properties

- Switch to the Java perspective and select your new project `rds-trunk`.
- Select the Ant build file `build.xml` from the root directory and open **Run As | 2 Ant Build...** from the context menu. This will open the **External tools** dialog.
- Select the **Refresh** tab and activate the checkbox **Refresh resources upon completion** and select **The project containing the selected resource**.
- Select the **Build** tab and deactivate **Build before launch**.
- Select the **Classpath** tab and the **User Entries** tree root. and click **Add JARs...** Add `lib/junit.jar` and `lib/antlr.jar` from your project `rds-trunk`.
- Select the **JRE** tab and activate the radiobutton **Separate JRE**. Make sure to select a 1.6.0 JRE. Using a separate JRE ensures that stand-alone Ant builds from the command line will also work. (This is a new feature of rds 0.14. Before, due to dependencies on Eclipse JET templates, it was required to run Ant in the Eclipse VM.)
- Click **Apply** and **Close**.
- Select **Window | Show View | Ant**.

- Goto the Ant view and select **Add Buildfiles...** from the context menu.
- Select `rds-trunk/build.xml` and click OK.

3.2.6. Building the rds project

Go to the Ant view, open the `rds` node and double-click on the `compile` target. This will run an Ant build that writes diagnostic messages to a console tab of the Eclipse workbench. For a stand-alone Ant build from the command line, simply invoke `ant` in the project root directory.

The build produces class files in `build/classes`. To build an executable JAR file `rds.jar` in `build/jar`, use the `jar` target. This JAR file depends on `rds-runtime.jar`, `antlr.jar`, `commons-cli-1.1.jar`, `freemarker.jar`. To do anything useful with `rds`, you also need (some of) the extensions `rds_javaExtension.jar`, `rds_htmlExtension.jar`, `rds_xmlExtension.jar` which will be located in the `build\jar` folder after the build.

The `test.run` target builds and runs JUnit tests for the project.