Installing and Building the DataScript Tools

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.

Table of Contents

1. Introduction

1.1. Purpose

This is a short guide for installing and running the DataScript tools and for getting the source and building the tools from source.

1.2. Package Overview

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

- `dstools-bin`: Binary distribution of a DataScript parser and code generator, directly derived from the reference implementation of Godmar Back. This was the starting point of the dstools project, which is due to be superseded by a reimplementation called rds (Relational DataScript).
- `dstools-src`: The source package corresponding to *dstools-bin*.
- `rds-bin`: Binary distribution of Relational DataScript, our current baseline version for a DataScript parser. The parser is complete, the code generator is still lacking support for relational extensions and for write() methods.
- `rds-src`: The source package corresponding to `rds-bin`.

1.3. Platform Support

The development platform for DataScript is Windows 2000, but the packages should run on all platforms supporting a Java SE 1.5.0. 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.

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

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 1.5.0 or higher to run it.

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

```
java -jar %RDS_HOME%\rds.jar -pkg <output package> <input file>
```

The output package option specifies the package name for the generated Java files. The files will be created in a subdirectory <*output package*> of the current working directory. Nested output package names (e.g. `foo.bar.bla`) are not yet supported.

<*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`.

3. Building from Source

The only supported build environment for the dstools project is Eclipse 3.2. Version 3.1.1 also works, but this guide is targeted at 3.2. If you know what you are doing, you will be able to build dstools in other environments, but Eclipse is the only one 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. Installing Eclipse

- Install JDK 1.5.0 from `http://java.sun.com`.
- Install Eclipse 3.2 from `http://www.eclipse.org`.
- Start Eclipse and set the proxy options in *Window* | *Preferences* | *Install/Update*.
- Goto *Help* | *Software Updates* | *Find and Install* to install the Eclipse Modelling Framework (EMF).
- Select *Callisto Discovery Site*. EMF is in the category *Models and Model Development*.

3.2. Installing the Subclipse plugin

Subclipse is a Subversion client plugin for Eclipse. Using this plugin, you can directly access Subversion repositories from Eclipse.

- 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 | 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.

3.3. Creating a local dstools 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://svn.sourceforge.net/svnroot/dstools-antlr`. Click *Finish*.
- When prompted for accepting a digital certificate, click *Accept Permanently*.
- Expand the repository tree and select the subnode `trunk/dstools-antlr`.
- 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*.

3.4. 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.

- `dstools-antlr`: Sources for the rds-src package. The name is obsolete and will be changed to rds.
- `www`: Content of the project homepage at 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.

3.5. 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*.
- Select the *Build* tab and deactivate *Build before launch*.
- Select the *Classpath* tab and click *Add External JARs...* Add `%ECLIPSE_HOME%\plugins\org.junit_3.8.1\junit.jar`.
- Select the *JRE* tab and activate the radiobutton *Run in the same JRE as the workspace*.
- 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.

3.6. Building the dstools project

Go to the Ant view, open the `ds-antlr` 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.

The build produces class files in `build/classes`. To build an executable JAR file `datascript.jar` in `build/jar`, use the `jar` target.

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