Build Documentation

Overview

Avro's build is coordinated by a top level build script, *build.sh* and delegated to each language's build.

Each language is expected to support three phases of the top level build:

- *clean* this target cleans out all file and directory artifacts created by the build.
- *test* this target executes all unit tests for all languages, after compilation or packaging, as needed. It then generates interop test data for all languages, and following that runs interop tests with all languages. Finally, it runs RPC interop tests. Calling *test* immediately after a *clean* is expected.
- *dist* this target builds and packages all artifacts for a release. This includes:
    - Creating a tarball of all source code in the avro tree, after validating that files have acceptabe license headers.
    - Building and packaging artifacts for all languages, including any documentation such as Javadoc.
    - Building the avro website.
    - Creates a tarball of all documentation and a directory per language with language specific artifacts.

Language Specific Details

Java

Prior to Avro 1.5, Avro Java was built using Ant and created two jars. avro.jar contained all of avro's classes, and avro-tools.jar contained that plus dependent jars and a main class for 'java -jar' usage.

Avro 1.5 and beyond use Apache Maven and is composed of multiple modules. Users can choose which subset of avro they need, and individual avro modules can dependon big, complicated things like hadoop without causing users who don't care about those features to pull in dependencies they don't need. Likewise, adding more modules with other dependencies is now easy.
Maven 3.0.2 is recommended, but the build should work with 2.2.1 as well. It has been tested with 3.0.1.

Project Structure

path from lang /java	artifactId	name	artifact type	notes	depends on
/	avro-parent	Apache Avro Java	pom	parent, inherits from Apache master pom, sets common build properties and versions	
/avro/	avro	Apache Avro	jar	avro "core" functionality.	jackson, paranamer
/compiler/	avro-compiler	Apache Avro Compiler	jar	Avro IDL and Specific compiler, including ant tasks	avro, commons-lang, velocity, ant
/maven-plugin/	avro-maven-plugin	Apache Avro Maven Plugin	maven-plugin	Maven mojos for avpr > java; avsc > java; avdl > java;	avro-compiler, maven
/ipc/	avro-ipc	Apache Avro IPC	jar	Avro IPC components, protocols, trancievers, etc	avro, netty, jetty, velocity
/mapred/	avro-mapred	Apache Avro Mapred API	jar	An org.apache.hadoop.mapred API using Avro serialization	avro-ipc, jopt-simple, hadoop-core (provided scope)
/tools/	avro-tools	Apache Avro Tools	jar (with dependencies)	A single jar containing all of Avro and dependencies, with command line tools	avro-ipc, avro-mapred, avro-compiler, avro

Building Instructions: command-line

To clean build all components without testing and install them in your local repository:

```
$ mvn clean install -DskipTests
```

To compile only:

```
$ mvn compile
```

To run tests:

```
$ mvn test
```

To install to local repo, including running tests:

```
$ mvn install
```

Other useful mvn commands:

```
$ mvn clean
$ mvn validate
$ mvn help:effective-pom
$ mvn site
$ mvn generate-resources
```

To download all available javadoc and source of dependent projects into your local repo:

```
$ mvn dependency:resolve -Dclassifier=javadoc
$ mvn dependency:resolve -Dclassifier=sources
```

In general, any maven plugin has documentation online, and available on the command line like so:

```
$ mvn dependency:help
$ mvn javadoc:help
$ mvn versions:help
```

There are a lot of plugins that do almost anything you need. For example, to find out if there are newer versions of any dependencies we use:

```
$ mvn versions:display-dependency-updates
```

to see a dependency hierarchy:

```
$ mvn dependency:tree
```

to create a code coverage report:

```
$ mvn cobertura:cobertura
```

Building Instructions: Eclipse

Use Eclipse 3.6 Helios
Use the m2Eclipse plugin.

- Load the projects into the workspace using the "Import ..." dialog, and select "Existing Maven Projects"
- Select the lang/java directory, and it should show all projects including the parent. Import all of these.
- After the load and first build, it will not completely compile. To fix it up to compile, select all of the projects and right-click. Select **Maven > Update Project Configuration**.
  There is a bug that occasionally appears if you are changing the way the compiler works. The projects that depend on the compiler will break. Restarting eclipse and doing the **Maven > Update Project Configuration** step fixes this.

The Maven Eclipse Plugin is also an option. Any instructions for that are welcome, the tricky part will be that some Avro projects depend on the output of other projects to compile.

More maven information:

These are a good start:

- http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
- http://maven.apache.org/guides/introduction/introduction-to-the-pom.html

For those new and experienced, "Maven By Example" is a very good intro – especially chapters 3+

- http://www.sonatype.com/books/mvnex-book/reference/public-book.html

Apache's maven policies, tips, etc:

- http://www.apache.org/dev/publishing-maven-artifacts.html#inherit-parent

Plugins used include:

- http://mojo.codehaus.org/javacc-maven-plugin/
- http://maven.apache.org/plugins/maven-surefire-plugin/
- http://maven.apache.org/plugins/maven-checkstyle-plugin/
- http://paranamer.codehaus.org/

Other useful plugins:

- http://mojo.codehaus.org/build-helper-maven-plugin/usage.html
- http://mojo.codehaus.org/cobertura-maven-plugin/
- http://maven.apache.org/plugins/maven-shade-plugin/
- http://maven.apache.org/plugins/maven-antrun-plugin/usage.html