AntBuilder

This document is aimed at AntBuilder 1.21

AntBuilder helps you construct and maintain an Ant[1] `build.xml` file for your project. Yes, Maven is gaining in popularity, and supposedly Ant is on the way out, but Maven is also a more complicated beast than Ant. In the interest of simplicity, I believe Ant (and by extension, AntBuilder) still have their place.

AntBuilder is a very simple GUI tool and requires Java 1.5 or later to operate. It should be relatively obvious to use. For the tl;dr[2] crowd the following summarizes the bare necessities:

- Whenver you click "Accept" ...
  - AntBuilder generates a `_build.xml` file (note the leading underscore). After you have tested this file, move it over the default `build.xml` file.
  - AntBuilder writes the `.antbuildrc` file (note the leading dot, this makes it a hidden file on Unix-type operating systems). This file contains all the settings. AntBuilder loads it next time you start it. If you delete this file you'll have to reconfigure AntBuilder next time you run it.
- Whenever you click "Cancel", AntBuilder simply quits. It does not write any files whatsoever.
- Tooltips on most fields should provide hints and even education.
- Run "`ant -p`" to view all the available targets (this is a standard Ant feature). Mostly, "`ant clean`", "`ant compile`", and "`ant jar`" should get you started.

Introduction

If you are not using a heavy-weight development environment that handles all the building for you, then you may be relying on Ant to build your project. Perhaps you find the process of building your own `build.xml` file challenging, even enlightening; perhaps you are copying and pasting pieces from other projects. The idea with AntBuilder is that it can construct a fairly decent and functional `build.xml` file, allow you to change your mind on any of the options, and quicly rebuild it (without ever overwriting your existing `build.xml` file.) It's convenience in a box. I hope you like it, because I sure do!

Antbuilder saves all settings in the current working directory, in a file named `.antbuildrc` and reloads this file when next you start AntBuilder. You can delete this file, causing AntBuilder to forget everything about your current project. AntBuilder will only save this file if you press the "Accept" button.

AntBuilder generates the `_build.xml` file (note the leading undersore) which you should test first so that you do not lose an existing (and presumably working) `build.xml` file. You test the `_build.xml` file by invoking Ant with the "`ant -f _build.xml`" command (add what options you want at the end of that); Once you are satisfied, move the `_build.xml` file over the (existing) `build.xml` file to eliminate the need to use the `-f` option.

Ant's GUI is organized into four sections:

Main — The options most commonly of primary interest

Other — Options of secondary importance, especially for more advanced use

Paths — The paths that AntBuilder will manage for you

JAXB — An experimental, and unmaintained option for building JAXB (Java XML Binding) stuff.

1 Ant is a project build tool published by the Apache Foundation. Although Ant is most often used for Java projects, it is perfectly capable of describing the build process for any other type of system, as well.

2 tl;dr = too lazy; don't read.

Main Options

Project name — This is intended for human-readable stuff elements of your project, and may include any text, including spaces and "special" symbols across the entire Unicode character set.

Executable — The name of the executable to build. You should omit a `.jar` or `.war` extension as that will be added by AntBuilder dynamically (see the Type option on the "Other" tab).

Main Class — The (fully qualified) name of the class that contains the `public static void main(String[])` method. This will affect the generated Manifest in the JAR file.

Java Source Version — If you use Generics, you must set this at least to "1.5"; if you use the diamond operator, try-with-resources, Strings in switch/case statements, and the like, then you must set this at least to "1.7"; if you are using Lambda expressions or JavaFX 8.0 then you need to set this at least to "1.8".

Java Target Version — You should set this to the smallest number possible, but no smaller than the Java Source Version, in order to make your software accessible to the most people. Just because you are running a bleeding edge pre-release compiler doesn't mean everybody does.

Create skeleton (source code) — This option allows you to specify what type of source code you want AntBuilder to generate for you, provided that the files it wants to build do not yet exist. In other words, AntBuilder will not overwrite existing source files! The purpose is to provide you with something you might find useful as the start of a project. Note that AntBuilder will always rebuild these files if they do not exist, so long as this option remains selected.

This value has (currently) three options:

1. None — Do not generate any source code at all.
2. Swing (Java 1.5+) — Generate a simple Java 1.5+ Swing GUI application. It will compile, but do precious little.
3. JavaFX (Java 1.8+) — Generate a simple Java 1.8+ JavaFX GUI application. It will compile, and even look somewhat pretty, but it does nothing. Note that JavaFX 2 made it into Java 7, but AntBuilder doesn't care about that (I never much bothered with earlier JavaFX, I'm afraid).

Images — Include image files in the JAR. This directory is created and deleted (only if empty) as necessary to match this option. See the 'Paths Options' section for more information.

Help files — Include help files in the JAR. This directory is created and deleted (only if empty) as necessary to match this option. See the 'Paths Options' section for more information.

Miscellaneous — Include miscellaneous files in the JAR. This directory is created and deleted (only if empty) as necessary to match this option. See the 'Paths Options' section for more information.

Sources — Include source code files in the JAR. Yes, this causes your project's source code to be included in the executable JAR. Read this paragraph again. If you writing Free/Open Source Software (FOSS) then this may be a really convenient option for you. If your company lives and dies by the secret sauce in your code, you probably don't want to check this option. It's always a good idea to check the contents of the JAR file before you ship it[3].

Package — If you want to build a WAR file, rather than a JAR, then check this option. Most of the time you are probably going to leave this option unchecked (i.e. build JAR files, rather than WAR).

Note that I have not built a WAR file with AntBuilder in quite a while, so I don't know how well this part of the code still functions. If you need to build WAR files instead, or even EAR files, and really want AntBuilder to do this for you, then let me know what you need it to do, and if

3 You can treat a JAR file like a ZIP file so long as you are careful. If your tools are dumb, then rename the file to have a .zip extension while you are inspecting or even manipulating it, so long as you restore the .jar extension afterwards.

you provide good example Ant XML for it, I'll do my best to teach AntBuilder the requisite skills. :)

Other Options

Libraries — There are three mutually exclusive options here, and your choice will affect the availability of the "signing" related options (see below):

- Merge JAR/ZIP files into executable — This extracts all your libraries into the executable JAR file.: You end up with a single executable JAR that has no external dependencies, your program is self-contained. If you sign your executable, the merged libraries will be signed with your key (this also loses any signatures on the libraries).

  You should be careful about licenses of external libraries, as you may not be able to (legally) merge your code and the libraries in this manner!
- Reference JAR/ZIP files in Class-Path — This lists all of your libraries in the "Class-Path" entry of the JAR's Manifest, effectively recording the libraries as external dependencies of your project. You must ship these libraries with your project (in a subdirectory that matches the name in the "Library" directory under the "Paths" tab) or in some other way ensure that these libraries are installed where your JAR can find them.

  When you pick this option, you also enable one to "Re-sign libraries" (see below)**.**
- No explicit libraries are used — This indicates that your software is soft-contained and uses no external libraries.

Sign JAR — Sign the executable JAR with a digital signature (see 'Signing properties file') to declare the origin of the file. Although signing a JAR does not imply that the software itself can be trusted to be safe or free of bugs, it does provides end-users with a means to verify that the person (or entity/company) who signed the application is the one that the user expected it to be, rather than an impostor.

Applets and applications delivered by Java Webstart need to be digitally signed.

Re-sign Libraries — At one time (perhaps to this day) Java Webstart would check only one digital signature and wanted that one to match all signatures in all JAR files; a mismatch caused the application to fail. The "Re-sign libraries" option is intended to overcome that limitation (provided the problem still exists) by throwing away the digital signatures of all the libraries, and applying your own digital signature to them, instead. **You need to ensure that you do not violate the licenses of the libraries by re-signing them!**

Signing property file — This names a simple property file (containing key=value pairs on separate lines) that provide four pieces of information:

1. keystore-file = Name of the Java Keystore file. Unless you really know what you're doing, you should ensure that this file is *not located in your project directory, and not checked into your repository, either.* The cryptographic keys in this file allow you to sign your code and thereby assert that it was you who build it. If someone else gets hold of your keys, they can sign whatever they like as if you were the one who build and certified it. Every developer should have his/her own keystore, so specify a file that looks something like "`../.keystore`" (also note that the leading symbol ~ (tilde) is equivalent to ${HOME} and will be replaced with the value of the user home directory, as reported by the Java Runtime.
2. keystore-pass = The password for the keystore. The presence of this password (as well as the key-pass, see below) is why you should be *very concerned* if the 'Signing property file' or the keystore are located inside your project directory, or even checked into the (same) repository as your project sources.

3. key-alias = The alias for the key to be used for signing applications. Keys could be addressed by their hexadecimal identifier, but an alias is easier to use, so that's what AntBuilder wants from you. The 'keytool' can help you manage aliases on keys.
4. key-pass = The password for the key matching the 'key-alias'. Again, the presence of this password is reasons for keeping the signing property file (and the keystore) out of the project directory, and out of the project's repository.

Checksum Files — Create MD5, SHA1, and SHA-256 checksum files. If you publish the checksums of your application in a way that "bad guys" cannot subvert (by hacking your site), then end-users can use tools like md5sum, sha1sum, and sha256sum to verify a digital checksum of your application JAR. Providing at least two (2) such checksums is recommended to avoid hash collisions from giving "bad guys" an opening, especially where MD5 and perhaps SHA1 are involved by themselves (one checksum may come out correctly, but if the other one fails, then the software is bogus). Ideally you use checksums in addition to a digital signature that is carried by the JAR itself. If you really want to pick only one, then go with the digital signature rather than the checksums.

Project log sources — If you want a simple way to extract a log file from your repository, then pick which repository you are using. The 'log' target builds the log file, named the same (but with a .log extension) as your executable.

Apologies for those who are using something other than Git or SVN. Let me know, and I'll be glad to include it in a future build.

Paths Options

Source directory — The name of the directory where source files are expected to be built. This is also the directory where skeleton source files will be created if you have that option selected in the Main options panel.

The default is "src"

Library directory — The name of the directory where library files (.jar/.zip) files are expected to be located (if any), provided that you have one of the options selected that ask AntBuilder to build code into the `build.xml` file that handles libraries.

The default is "lib"

Images directory — The name of the directory where image files (such as `*.png`, `*.jpeg`, `*.gif`, etc.) are expected to be located. These images are copied into the executable JAR such as they are available in a directory matching the name of the images directory. Example, if you select "img" as your images directory, then a file "`logo.png`" will be available to your application as a resource named "`img/logo.png`"

The default is "img"

Help directory — The name of the directory where your application's interactive help files and data are located. Just like images, these will be made available to your application as resources inside a directory that matches the name you select here.

The default is "help"

Miscellaneous directory — The directory where miscellaneous files are located. You can store anything in this directory that doesn't really fit into the other categories. Just as with help and image files, these files will be made available to your application as resources in a directory that matches the name you select here.

The default is "etc"

JAXB Options

Include JAXB — Add XML to the `build.xml` that handles JAXB files. Java files generated with JAXB for this purpose will be placed into a subdirectory of your sources directory named 'generated' (i.e. they will be part of the 'generated' package).

The files are generated with the `xjc` command, passing three parameters:

1. The name of the `nodeList.xsd` file (located in the JAXB directory)
2. The name of the sources directory (prefixed with a `-d` flag)
3. the name of the 'generated' directory within the sources (prefixed with a `-p` flag)

NOTE: JAXB is not presently a supported functionality in AntBuilder, as it was needed for a project a long, long time ago and has not been exercised since then. If you find it useful, great; if you need it to do more that what it does now, provide me with the necessary information (example XML) so that I can teach AntBuilder what it needs to know. :)

Organizing your Project

- Sources go into the sources directory, whose name is specified under the Paths Options. There is an option, too, that causes your `build.xml` copy your sources into the executable JAR file. This is intended for Free/Open Source Software.
- AntBuilder adds into the JAR file any `LICENSE.TXT` and/or `README.text` and/or `*.fxml` (Java FXML) files that it finds in your project root directory.
- Libraries that your project needs to into the libraries directory, whose name is specified under the Paths Options. There are two ways to handle libraries: (1) make them an integral part of your executable JAR, or reference them as external dependencies. This is specified under the Other Options panel. Externally referenced libraries are referenced relative to your executable JAR file in a directory that matches the libraries directory.
- Image (picture) files that your application needs should go into the images directory, whose name is specified under the Paths Options. Images are available to your application as resources in a directory with the same name as the images directory. You need to enable inclusion of images with an option in the Main Options panel.
- Help files that your applications wants to use should go into the help directory, whose name is specified under the Paths Options. Help files are available to your application as resources in a directory with the same name as the help directory. You need to enable inclusion of help files with an option in the Main Options panel.
- Miscellaneous files (anything that doesn't match what's described above) should go into the miscellaneous directory, whose name is specified under the Paths Options. Miscellaneous files are available to your application as resources in a directory with the same name as the miscellaneous directory. You need to enable the inclusion of miscellaneous files with an option in the Main Options panel.
- If you want to digitally sign your application (and you really should!) then specify in a properties file the information that the `build.xml` file needs to access your (private/secret) keystore. Also note that you really (really!) do not want to make that keystore part of your project (in other words, do not keep in your project repository, and certainly not in your project directory, either.

Changes to this Document

This documentation was updated for AntBuilder 1.21 on 4-Jul-2013.

The original version of this document is dated 21-May-2010.