ABT User Manual

Maxime Biais

Artenum

November 26, 2003

1 Introduction

ABT automates the building of projects by executing the minimum set of commands to update the project files like the famous *Make*[1]. ABT is given a file, called the recipe (filename is abt-recipe.xml), that describes how the files that make up the project are related (how they are dependent upon one another) and how to update (build) those files. ABT checks the dates on the files to see if any need to be updated (when a target is older than one of its sources). If there are files that need to be updated, ABT executes the rule actions specified in the recipe for that purpose. ABT have also testing features and os dependant command execution.

2 ABT recipe

To write your own compilation process you need to write a file understandable by ABT, this file is called a recipe (abt-recipe.xml). The recipe is based on XML file format and is cut in 2 different part:

- the project decription part (between the `<project></project>` elements)
- the makefile part (between the `<makefile></makefile>` elements)

[1] http://www.gnu.org/software/make/make.html

2.1 Project Part

Here is a simple commented example of a project description:

```
<project>
  <!-- The name of the project -->
  <name>abt</name>
  <!-- The version of the project -->
  <version>0.3</version>

  <!-- Binary dependencies -->
  <!-- We need python version >= 2.1 -->
  <dependence_bin name="python" version="2.1" />
  <!-- We need jython version >= 2.1 -->
  <dependence_bin name="jython" version="2.1" />

  <!-- Library dependencies -->
  <!-- We need the jython-vtk library used with the jython interpreter -->
  <dependence_lib name="jython-vtk" compiler="jython" />
  <!-- The jython-picup library used with the jython interpreter is optional-->
  <dependence_lib name="jython-picup" compiler="jython" option="yes" />

  <!-- Here we are asking the user to enter a value for the INSTALLDIR
       variable used later in the makefile part -->
  <dependence_ask name="INSTALLDIR" version="enter the install directory" />
</project>
```

2.2 Makefile Part

This part describe a Makefile-like formated in XML. Like old Makefile you define variables (also called MACROs) and write rules (also called target).

2.2.1 Variables

A variable is declared with the `<var></var>` element `<var name="POULET">a poil</var>` (define POULET="a poil"). Variables can be used in rule name, action content or variable content.

There are several builtins variables:

PROJECT_NAME, name of the project

PROJECT_VERSION, version of the project

BUILDER_MINIMAKE, path to minimake.py file (used to call the building process without calling binary and library tests)

BUILDER_RM, path to the rm builtin file (os independant rm-like written in python)

BUILDER_CP, path to the cp builtin file (os independant cp-like written in python)

BUILDER_CWD, full path to the directory where abt have been invoked (corresponds to the result of the pwd command on unix)

BUILDER_INTERPRETER, interpreter running the builder (if python is used, then this is the full path to the python interpreter, if jython is used then this variable is set to 'jython')

INSTALLDIR, install directory (default: "/usr/local") see 3 for more informations to set the INSTALLDIR variable.

For example if you want to re-run the building process as a rule you can write in an action or a variable content:

```
$(BUILDER_INTERPRETER) $(BUILDER_MINIMAKE)
```

All the *tested* interpreters/compilers/librairies are also transformed in builtins variables for the example `jython, python, java` binaries can be find with the corresponding variables name. for example if jython have been found by the tester you can use the variable $(jython) pointing to the full path of the jython interpreter (for instance `/usr/local/jython-2.1/jython`) in an action, a rule name or a variable content.

2.2.2 Rules

Here is a commented example of a makefile rule:

```
<!-- defines the distclean rule -->
<rule name="distclean">
  <!-- the documentation string wants to describe the rule (it's optional) -->
  <doc>Removes generated files</doc>
  <!-- distclean depends on the clean rule-->
  <prerequisite>clean</prerequisite>
  <!-- the following action will be executed on all OSes and will be muted -->
  <actions os="all" mute="yes">
    echo "distclean rulez !"
  </actions>
  <!-- the following action will be executed only on unix and will
       not be muted -->
  <actions os="unix">
    rm -rf *.class
    rm -rf *.pyc
    rm -rf $(PROJECT_NAME)-$(PROJECT_VERSION).tar.bz2
  </actions>
</rule>
```

- os = all|win|unix
- mute = yes|no (optional and default is no)

3 Invoke ABT

Usage: `abt [OPTIONS] [RULE]`

`OPTIONS`

-h print a short help describing invokation of abt and exit.

-v print all documented rules and exit.

-f filename use filename recipe instead of the default abt-recipe.xml.

-p path tell abt to search binaries in the PATH environment variable plus the path string (you can specify several path separated with ":" under Unix and ";" under Windows).

-i path set the abt INSTALLDIR variable to path (usefull when you write installation rules in your recipes).

-c Force checking and rewrite abt.cache.

`RULE` specify the rule to run, if not precized the `start_rule` defined in the recipe is executed.

4 Miscellaneous

4.1 Cache

Abt checked data are cached in the file abt.cache (binary file generated using pyhton/jython pickle module). The purpose of caching variables and detected (or not) binary or library is to increase abt speed. If you want to force the check you can pass "-c" option to abt (see 3)

5 ABT testing library

Binaries or libraries without version check can't be tested (sometimes the programs does not permit user to get its version). For the version shipped with this manual the supported binaries are:

Name	Version checking
Binaries	
python	Y
jython	Y
jythonc	N
java	Y
javac	N
gmsh	N
medit	N
tetgen	N
Libraries	
jython-vtk	N
jython-picup3d	N