EBuild

COLLABORATORS			
	TITLE : EBuild		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 22, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

EBuild

1.1 EBuild.guide

```
EBuild
The Make clone
Copyright 1997 Glauschwuffel, Wouter, Rob

  Introduction
  Invoking EBuild
  Build Files
  Misc
  History
  The Authors
```

1.2 EBuild.guide/Introduction

```
1 Introduction
**************

   EBuild is a "Make" clone, and it functions likewise. It is a tool
that helps you in recompiling necessary parts of a large application
after modification.

   You write a file `.build' in the directory that contains the sources
of your project. The file contains info about which sources depend on
which, and what actions need to be performed if a module or exe needs
to be rebuilt.

   EBuild checks the dates of the files to see if a source has been
modified after the last compilation, and if the source uses modules
that also have been modified, it will compile these first.
```

1.3 EBuild.guide/Invoking EBuild

2 Invoking EBuild
****

EBuild can be run from any shell. It's arguments are:

```
TARGET,FROM/K,FORCE/S,VERBOSE/S,NOHEAD/S,CONSTANTS/S:
```

If you supply a 'TARGET', this way build will start with another target. 'FROM' allows you to use another file than '.build', and 'FORCE' will rebuild everything, regardless of whether it was really necessary.

'VERBOSE' makes the program print each action it executes. 'NOHEAD' doesn't print the heading line and the 'CONSTANTS' switch forces EBuild to tell you what symbolic constants are there.

1.4 EBuild.guide/Build Files

3 Build Files
****

Symbolic Constants

Build files are normally named '.build'. This is the file EBuild looks for when it is run.

The syntax of build files equals that of unix-make. In general, '#' precedes lines with comments, and:

```
target: dep1 dep2 ...
  action1
  action2
  ...
```

'target' is the resulting file we're talking about, in most cases an exe or module, but may be anything. Following the ':' you write all files that it depends upon, most notably its source, and other modules.

The actions on the following lines are normal AmigaDos commands, and need to be preceded by at least one space or tab to distinquish them from targets.

```
bla: bla.e defs.m
     ec bla quiet
```

This simple example will only recompile 'bla.e' if it was modified, or if the 'defs.m' which it uses was modified.

If you type 'build' with no args, build will ensure the first target in the file to be up to date.

A longer example:

```
     # test build file

     all: bla burp

     defs.m: defs.e
      ec defs quiet

     bla: bla.e defs.m
      ec bla quiet

     burp:  burp.e
      ec burp quiet

     clean:
      delete  defs.m bla burp
```

   This build file is about two programs, 'bla' and 'burp', of which 'bla' also depends on a module 'defs.m'. An extra fake target 'clean' has been added so you can type 'build clean' to delete all results.

   It's okay to have fake targets, however, these cannot be used as module dependencies.

   Other dependencies and actions are easily added. For example, if your project uses a parser generated by E-Yacc:

```
     yyparse.m: parser.y
                eyacc parser.y
                ec yyparse quiet
```

   Or incorporates macro-assembly code as often used tool module:

```
     blerk.m: blerk.s
              a68k blerk.s
              o2m blerk
              copy blerk.m emodules:tools
              flushcache tools/blerk
```

1.5 EBuild.guide/Symbolic Constants

```
3.1 Symbolic Constants
======================
```

   EBuild v3.3 has symbolic constants. Before writing the rules you can set a symbol to any value. Those symbols can be used in rules and actions. The text of a symbol will be inserted wherever the symbol is found.

   Example:

```
     options=IGNORECACHE LINEDEBUG DEBUG
     test:  test.e
      ec test.e $(options)
```

```
   or even

     testfile=bla
     $(testfile):$(testfile).e
      ec $(testfile).e
```

   There is one special symbol in EBuild: 'target'. It holds the name of the target the current action belongs to. In the example above we can tell EC to compile the target instead of writing the actual name:

```
     options=IGNORECACHE LINEDEBUG DEBUG
     test:  test.e
      ec $(target).e $(options)
```

   This may seem to be not too useful, but take a look at this example:

```
     options=IGNORECACHE LINEDEBUG DEBUG
     test:  test.e
      ec $(target) $(options)
      if warn
        echo "Error: compile failed"
      else
        echo "Compiled OK... running"
        $(target)
      endif
```

   It's largely equivalent to the old code below, but allows more.

```
     options=IGNORECACHE LINEDEBUG DEBUG
     all: test
      echo "ok, running:"
      test

     test:  test.e
      ec -q test $(options)
```

   All symbols except '$(target)' may be used in rules as well as in actions. '$(target)', however, may only be used in actions. It's safe to have it in rules, EBuild just aborts with a message that tells you that it doesn't know this symbol.

1.6 EBuild.guide/Misc

```
4 Misc
******
```

   Once you get to know build, you'll discover you can use it for more purposes than just this. See it as an intelligent script tool.

   If you want to find out the details of what build can do, read the documentation of some unix-make, as build should be somewhat compatible with this. What it doesn't do for now, is:

```
   - allow "\" at the end of a line for longer rules

   When EBuild discovers a cyclic dependancy it just aborts, i.e. this
won't be executed:

     bla: defs.m blurp.m bla
      ec $(target).e

   since the target 'bla.e' has the dependancy 'bla.e'. EBuild used to
crash with an infinite loop on this one.
```

1.7 EBuild.guide/History

```
5 History
*********

   For v3.1 it was updated by Jason Hulance, to fix the bug that
executed actions in reverse order. Also he introduced the local variable
$target in actions.

   EBuild was updated for v3.3 by Gregor Goldbach to support symbolic
constants and to stop on cyclic dependancies. The $target behaviour was
expanded to match other symbols: $(target) is legal, too.
```

1.8 EBuild.guide/The Authors

```
6 The Authors
*************

   Wouter van Oortmerssen is the creator of E. He has studied computer
sciences and lives in England where he occasionally destroys monitors.

   Gregor Goldbach loves E, starts studying computer sciences very soon
and lives in Germany. He met Wouter but his monitor is still running.

   Jason R Hulance is an Englishman and they say he has met Wouter
several times. He coded some tools for E of which one will work
together with EDBG in the next E release.

   Rob is just Rob.
```