Speedy Make

A replacement for make and makefiles
by Kim Haskell
and Denis Sureau

Home page: http://www.speedymake.com
This page: http://www.speedymake.com/speedymake.html

Table of Contents

Overview

Speedy Make replaces make and its makefile format, by the simplest parser and makefile format possible.
XML is used as makefile and some predefined tags are sufficient to give tasks to the interpreter.
It is a small engine that reads an XML file, and with just a few rules, turns it into a list of commands to process.
Speedy Make is free and open source under the GPL license.

Overall structure of a SpeedyMake's makefile:

```
<?xml version="1.0" ?>
<makefile name="">
    <binary>
    </binary>
    <sources>
    </sources>
    <compile>
    </compile>
</makefile>
```

The root is **makefile**.

The **binary** tag holds the name of the binary executable that will be built.
Sources holds the list of source files.
Compile holds the command for the compiler.

How it works

Main algorithm

- Evaluates options at command line and set the flags.
- Reads the XML makefile.
- Check the conformity of the file.
- Convert the makefile to internal tree.
- Execute the top action.
- Execute dependencies.

Top action

- Get the list of actions from the content of the tag.
- Execute each action in the list in order.

Referenced actions

- Recursively from the top action, search for all dependencies.
- The content is made of actions and variables.
- If an action holds variables, replace the variable by its value.
- Actions are made of actions or are terminal actions.
- For each terminal action, build a command line.
- Scans the command line for embedded variables. Processes variables prefixed by $ and postfixed by [].
- Build a list of commands.

Referenced variables

- Get the element.
- Process the content as a list of file. Replace extension if needed.
- If action is "parse", recursively extract include statements, and build dependencies.
- Check existence of each file.
- Build a string from the full list of files.
- Replace the variable by the final string.
- If the variable is not a list of files, replace the variable by the content packed into a string.

Final commands

- Executes each command.

Unreferenced tags

- Unreferenced tags are ignored. A list of ignored tags is built, and is displayed in verbose mode only.

Speedy Make makefile

A Speedy Make file, is an XML document written into a pure text file, with any extension,
but preferably the ".sm" extension.
It is designed to be the simplest XML document possible, and just some rules define a makefile.
1) Components are variables or actions.
2) The name of a tag is not significant.
3) The role of the tag depends upon the "action" attribute.
4) Attributes are commands for the interpreter. The interpreter reads the content of the tag to perform this command.
5) The top action called at start is the tag whose name is not in the content of any other action.

Names of tags are freely chosen by the user. Only attributes are predefined.

Tag

The interpreter recognizes tags according to the attribute. If no attribute is given the tag is an action and the content is a list of actions.

These internal actions are recognized:
- display: display the content of the tag.
- parse: make a list of the content of the tag, and scan the file for include statements.
- run: build a command. A tag is a terminal or not terminal.
- A terminal tag holds a command and has a content used by the command.
- A non-terminal tag holds a list of tags (terminal or not terminal).

Syntax inside quoted strings

$varname designates a variable.
[] is an extension
$varname[.ext]
means that files that are the content of the tag "varname" has their extensions replaced by ".ext"
Example:

```
<source>
 mysource.cpp
 main.cpp
</source>

<action>
 program = "link $source[.o]
/>
```

is translated into:

```
link mysource.o main.o
```

No more syntax rules required.

Variable

A variable is a tag and its value is the content.

```
<myvar>
  value
</myvar>
```

a) Using a variable

To put the variable into a command, use this syntax:

```
command $myvar options
```

The string "$myvar" is replaced at runtime by the content of the tag.
At command line, a variable may be assigned a value with this syntax:

```
-myvar="othervalue"
```

This replaces the initial content of the tag written inside the XML file, by "othervalue".
The initial content thus, is the default value.

Example:

```
<var1>
    a.cpp
    b.cpp
    c.cpp
</var1>
<compile>
  mycom $var1
</compile>
```

The following command will be sent:

```
mycom a.cpp b.cpp c.cpp
```

The extension may be replaced, while using the same list of files:

```
<compile>
  mycom $var1[.obj]
</compile>
```

This command will be sent:

```
mycom a.obj b.obj c.obj
```

b) Repetitive variable

A repetitive variable is called with the "*" prefix.

```
command *myvar options
```

In some case, the elements that are the content of the variable must be replaced successively in the command.
For example, the variable $x holds: a, b, c.

```
<var1>
  a.cpp
  b.cpp
  c.cpp
</var1>
<compile action="none">
  mycom *var1
</compile>
```

The command "mycom *var1" will be sent with each element of the list:

```
mycom a.cpp
mycom b.cpp
mycom c.cpp
```

Action

The content of an action is a list of other actions, or a command.
- A non terminal action holds a list of actions, that are name of tags.
- A terminal action is a tag that holds a command, an external program, with options.

a) Terminal action

The "run" attribute denotes a terminal action:

Syntax:

```
<tagname action="run">
  program and options, including variables
</tagname>
```

Before to process an action, the smake interpreter:
- removes extra-spaces, tabs and end of line codes.
- replaces the name of the variable, by the content.

b) Non terminal action

For non terminal actions, the syntax is:

```
<toptagname>
  tagname
  tagname2
  etc...
</toptagname>
```

The content is the name of another tag, or a list of names.

Top action

At run the engine executes the top action defined inside the makefile.
The top action is a simple command when only one action is defined.
If several actions are defined, the upper one is the one that calls other actions and is not called by other actions.
The non terminal action above is the upper action in the example.
If several non terminal actions are defined, you have to defined one that call another and is called by none, to make it the top one. The number of dependencies is calculated by the smake interpreter.

Dependencies

The dependencies of C-like files is denoted by include statements in C sources.
The smake program reads sources to get the include statement and it calculates the dependencies, according to date of modification of each source.
Nothing else is required in the makefile to calculate dependencies.

Once all files are recognized, their dates are compared.
When a source is newer than the binary executable to build, the source is compiled, and all sources that include the header of this file, are compiled also.

List of actions

These actions are recognized by the smake program.

Display

Display the content of the tag.
Indenting is removed:
- tabs are supposed to indent the text and are ignored,
- white spaces are supposed to be a part of the text and are displayed.
For example:

```
     Done.
```

If done is shifted by tabs, it will be displayed as:

```
Done.
```

else, if it is padded with blank spaces, it will be displayed as:

```
    Done.
```

Parse

Apply a special algorithm to scan recursively contained list of files, get all include statements, and extract included filenames.
Cross references are solved and a list of unique filenames is returned.

Exclude

This optional action allows to exclude a list of files from a parsed list, generated at runtime.
The **target** attribute is assigned the name of the variable that holds the list of files in which these files must be excluded.
The syntax is:

```
<tagname action="exclude" target="sources">
   ... list of filenames ...
</tagname>
```

Configure

Not yet implemented. Will be equivalent to the configure program of Unix. It is only possible for now to use external program instead.

Build

The goal of this action is specify the binary executable to build. The date and time of this file are read, and sources are compiled if they have been modified after this time.

None

No action. This tag is not a command, and not an action.
This property is used to exclude the tag of possible top actions.

Run

Execute an external program.

Using Speedy Make

At command line, you can send the name of an action tag or a list of actions tags to execute.
Each of them is prefixed by the "-" symbol.
The makefile to process is not prefixed by this symbol. If no makefile is given, the default "makefile.sm" file if searched in the current directory and processed.
The smake program has its own options, that are prefixed also by the "-" symbol.
Options are one letter and so one letter tag names are rejected by smake to avoid confusing.

Options:

```
-a: All objects files must be rebuild. Dependencies are ignored.
-h: Display the format of the command and these options.
-t: Test and display only, don't execute the commands.
-v: Verbose. Display more infos about the processing.
-d: Debug, display more infos again. Display commands to execute.
```