ASTROPHYSICS and SPACE RESEARCH GROUP
DEPARTMENT OF PHYSICS AND SPACE RESEARCH
UNIVERSITY OF BIRMINGHAM
Asterix System Note
ASTERIX System Note 002
SYS/002
David J Allan
7 Jan 1995

The GEN_make Procedure

1 Introduction

This document describes the `GEN_make` Makefile processor – the Tcl script which generates ASTERIX `Makefiles`. This script exists because editing the ASTERIX make files is too error prone to be done by hand. It is better to reduce the varying information to control files and put the invariant rules on how to build the various components of the system in proto-make files. The reduction in volume thus achieved is about a factor of 40.

In addition to the script itself there are only 3 files involved when `GEN_make` is run. They are named according the the *class* of directory for which the `Makefile` is being constructed. They are,

class.config Contains the configurations of the `Makefile` which are peculiar to the directory to which it refers. It might, for example, contain the names of the programs to be built.

class.make The proto-type make file. This is the file which contains the skeleton of the `Makefile`. Things which vary from one instance of a directory class to another are parameterised using GEN_make tokens, which have the form *colon_identifier_colon* (eg. `:FILES:`). Proto-type include files may also contain `#include` directives which cause file inclusion. This enables entire sections common to different classes of make files to be shared, further reducing the number of potential errors.

class.base_config This file, in the same format as the **class.config** file, contains default definitions for all the tokens used in the proto-make file. This serves two purposes. It enables developers to realise the tokens which can be used, and it serves to provide default behaviour which may or may not need to be overriden.

2 How It Works

Consider a directory in which a file called `etc.config` exists in the directory `$AST_ROOT/dev/etc`. How does GEN_make go about creating a `Makefile`? To get it started the user types,

```
% GEN_make etc.config
```

The file extension may be omitted. GEN_make then follows the following algorithm,

1. The class is derived from the name of the configuration file. In this case it is `etc`.

2. GEN_make clears its token substitution array. It defines a number of standard tokens which are available for use by all proto-make files. They are listed below (without their delimiting colons), with their values in parentheses where relevant,

   CLASS The name of the class (*etc*)

   ROOT The relative file specification to get to `$AST_ROOT` (*../..*)

   HERE The location of the `Makefile` relative to `$AST_ROOT`. (*dev/etc*)

   SUBSYS The is full name of the subsystem. Same as the HERE token, but with slashes converted to underscores. (*dev_etc*)

   MODULE The leftmost directory name in the HERE list (*dev*)

   MODSUB The rightmost directory name in the HERE list (*etc*)

   DATE The date in `14 Jul 1789` format

   CREATOR The version of GEN_make used to create the `Makefile`

   HOST The machine on which the `Makefile` was made (*xun9*)

   PKG The package name. Hardwired at the moment (*ASTERIX*)

   INDENT A joker this one. Essentially a string of spaces whose length is $min(1, 2*depth)$. Useful for formatting recursive `Makefile` text.

3. GEN_make loads the value of the `AST_MAKE_PATH` variable. This is a colon separated list of the directories which will be searched for proto-make files.

4. The script checks that the *class*.**make** and *class*.**base_config** files exist somewhere in `$AST_MAKE_PATH`.

5. The token translations in the files **global.base_config**, *class*.**base_config** and *class*.**config** files are loaded in that order. See the next section for a discussion of the format.

6. The proto-make file is processed to remove all `#include` directives – the file name argument following the `#include` specifies a file which *must* lie somewhere in the `$AST_MAKE_PATH` directory list. This file flattening process is performed for speed's sake for a C tool which lives in the kernel called `at_strm`.

7. GEN_make now reads this flattened file looking for occurrances of its tokens. When it finds one it substitutes the token value defined by the configuration files and/or its built in tokens. This process continues until each line is free of tokens. The only further complication is the `#foreach` directive. This is a control structure within the proto-make file for generating multiple copies of the same bit of make text. An example is,

```
#foreach &X& :TASKS:
$(BIN)/&x&: &x&.f
    alink $? `ast_link_adam`
    mv &x& $@

#end
```

This directive expands *zero or more* times, depending on the number of space separated values in the translation of the token specified as the second argument. The first argument is the loop variable name – it must be enclosed in ampersands. This apparently silly choice of ampersands and colons was done deliberately to make GEN_make stuff stand out from normal `Makefile` text, and ensure `make` will break if you leave any such tokens untranslated. Upper case instances of the loop variable expand to upper case copies of the elements of the second and subsequent arguments, and lower to lower case copies.

The utility of this directive can only be appreciated after a thorough understanding of the differences of OSF, Solaris and Linux 'make' syntax. The above text of a good example of why it useful. The final target is in another directory and we only want to rebuild it if it is out of date with respect to its source code.

The (entirely) blank line before the `#end` marker is required to generate correct `Makefile` source if there is more than one repetition of the directive.

The `#foreach` directives cannot be nested.

8. The output file is renamed to be `Makefile`. Any pre-existing file of that name is deleted.

3 Configuration File Syntax

To recap the behaviour of GEN_make is quite simple. It translates named tokens in an input file to token values in an output file. The job of the configuration files is to manipulate the GEN_make token translation table. GEN_make is written in **Tcl** and the token translation table is just an associative array called `Subs`.

GEN_make loads configuration information by `source`ing them, ie. a configuration file is itself a Tcl script! This cunning trick is a generally useful technique and I commend it to you. There are currently two configuration file commands, called `files` and `const`. These are there for historical reasons, in the current implemention there effect is similar except for the fact that `const` can have only one argument, whereas `files` can have many. The syntax is,

```
files TOKEN values

const TOKEN [value]
```

In the `const` case the value defaults to the null string. In addition to simply listing file names there are some 'specials'. These have the following forms,

```
files NEW_TOKEN %remext OLD_TOKEN

files NEW_TOKEN %reverse OLD_TOKEN

files NEW_TOKEN %upper OLD_TOKEN

files NEW_TOKE %lower OLD_TOKEN

files NEW_TOKEN %prefix <prefix> OLD_TOKEN
```

```
files NEW_TOKEN %suffix OLD_TOKEN <suffix>

files NEW_TOKEN %presuf <prefix> OLD_TOKEN <suffix>
```

The use of these is hopefully clear! They add or subtract bits from file name to create new tokens, or reorder the elements of a token. If the bit you are adding contains characters special to Tcl (eg dollars) then you should enclose the bit in braces, eg.

```
# The task names

files TASKS  hdir hcreate hretype

# The Fortran files

files FORTRAN %suffix .f TASKS

# The executables

files EXECS %prefix {$(BIN)/} TASKS
```

It is important to realise that all token translation is delayed until the proto-make file is processed, ie. it is quite valid to use one of the specials above before `TOKEN2` is defined.

The ability to do this is extremely useful. We only have to name the tasks *once* – all the rest is rule based. In Starlink make files task names often appear in 4 or 5 places.

4 Adding a Directory Class

To add a directory class you must create a `.base_config` and a `.make` file in the `dev/etc` directory. Then edit the `etc.config` file in that directory, build it, and then you are ready.