PS 027
Understanding what ‘make’ can do for you

David J Garbutt

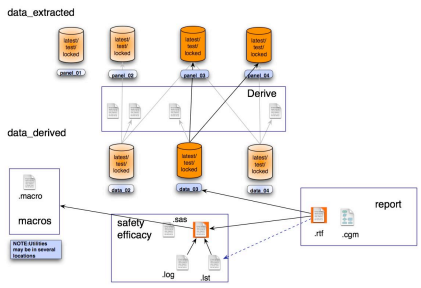
BSI AG, Täfernstrasse 16A, Baden-Dättwil, CH- 4504 d.garbutt@bsiag.com

Introduction

- This poster shows more information on make than could be included in the talks AS09 & RC05 'Implementing CFR21 Part 11 for SAS without tears or joins’
- Current practices for large deliveries are inefficient, stressful and prone to errors.
- The (Unix) make program can assist in the timely and fulfilment of reporting deadlines. Automation of processes is a key part of innovation. Why should this not apply to programming?
- Creating Project deliverables - AUTO-magically
- We will see how it is possible *and useful* to use standard UNIX tools to create and maintain up-to-date All the project deliverables By managing the content of one file - the makefile.
- make is effectively a tool for writing custom batch files that do the minimum needed to finish the job. To make this possible make needs to know what is required to create all the objects it is controlling.

What is a dependency?

- A dataset *depends on* the program that created it. If a program creates a set of data then if the program is changed then the dataset should be recreated.
- Analysis datasets are built from the raw extracted datasets. Often they are built from several of these. An analysis dataset *depends on* the program that created it and also on the extracted datasets it reads data from.
- Outputs such as RTF files are built by programs using Analysis datasets and system macros.
- These are all dependency relationships and they form a complicated network of relationships that is hard to grasp mentally.
- Diagram of typical dependencies



This diagram shows the a typical network dependencies traced back from an RTF file back to the extracted datasets. Areas enclosed in blue rectangles represent directories. The convention used is that dependencies point upward., so that objects derived from others are below them. Notice that the data flow takes the opposite direction and is therefore downwards, from extracted data to formatted report.

How does make work?

Part of the brilliance of make is to let the program itself build the network and let the user specify only the dependency pairs. The information on pairs is relatively easy to determine. 'make’ reads the dependency pairs and links them to build a network, which it uses to calculate which commands to execute.

How do we describe dependencies?

- Take as an example a program with the same dependencies as in the diagram.

1. July.rtf is built by july.sas using derived.data_03.
2. Analysis dataset data_03 is built by data_03.sas from panel_03 and panel_04.

- To describe a dependency to make we write at least two lines:

```
july.rtf : july.sas data_03.sas7bdat
    sas july.sas
```

- The dependency second is specified like this (\ means ‘no line break’)

```
Data_03.sas7bdat:data_03.sas \
panel_03.sas7bdat panel_04.sas7bdat
    sas data_03.sas
```

Note: *The piece in blue is what we want to create (the target), those in orange are the ingredients we need, the ':' (colon) is the operator 'is built from' and the line in green is the the command that must be executed to build July.rtf from its ingredients.*

Running a makefile

- These commands are put in a file called Makefile in the same directory as the files mentioned. The work is done by typing:

```
make July.rtf
```

- Then if any of the data_03.sas, panel_03 or panel_04 files have altered since july.rtf was modified then make will issue the commands:

```
sas data_03.sas
sas july.sas
```

- If only july.sas, or the data_03 dataset have been altered then only the first of those two SAS commands are run. If everything is up-to-date then make does nothing.
- Any target in the make file can be passed as an argument to the make command.
- It is also possible to use special targets. 'Special’ because they have no associated commands to run. Special targets allow make to be used to structure the work needed into convenient pieces.
- If we follow standard practice our files will normally be in different directories. This can be handled several ways: One way is to add the files’ path to each use in the make file. Another way is add make macros (in SAS terms macro variables) to use as standard directory paths. This results is simpler maintenance and shorter targets to call.
- It also means anyone can generate all the studies’ outputs without detailed knowledge of what is needed. This is a tremendous advantage.

Problems with make

Although make files are typically created once for a project there is some maintenance overhead. Perhaps worse is the need to protect against changes that make the makefile outdated (e.g., a new program is added to the output list), because if we forget to maintain the makefile the new output will not be updated when necessary. So make files need to be validated in some sense.

We need to put *all* the dependencies in the make file - this includes macros called, include files, utilities. This is not so easy because dependencies may be hidden within utility or reporting macros that we do not expect the SAS programmer to know about. Worse, the dependencies can also be dependent on parameters passed with sysparm to SAS at runtime.

These issues can be ameliorated by writing a system to manage the programs and their dependencies (and other useful information such as completion status) which can generate an updated make file.

There are also problems because 'make’ uses modification dates to make build decisions and these are not reliable for SAS binary files.

ClearCase and clearmake

There is another way - use the ClearCase version control and configuration management system. ClearCase comes with version of make (clearmake) that solves the lost dependencies problem.

Clearmake can do this because of two features:

1) ClearCase keeps special metadata for objects that clearmake builds. These objects are called derived objects and the metadata is held in a Configuration record.

2) Build decisions are made by version number, not by modification date.

This is possible because clearmake can look in the configuration record.

Sample Configuration record

```
cleartool> catcr "X:\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398"
Derived object:
\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398
Target m:/djg_view/proj_a/PROTOCOLxxxx/export/pgm_eff/x7107_7_01.rtf built by
ccadmin.CLEARCASE_Users
Host "hhtga-dt-42" running NT 5.0 (i586)

Reference Time 04-Jul-03.10:20:02, this audit started 04-Jul-03.10:20:02

View was hhtga-dt-42:d:\ClearCase_Storage\views\ch\ccadmin\djg_view.vws
Initial working directory was Y:\proj_a\PROTOCOLxxxx\export
-----------------------------
MVFS objects:
-----------------------------
\proj_a\PROTOCOLxxxx\data_01\diawky.sas7bdat@@\main\1  <26-Jun-03.15:20:38>
\proj_a\PROTOCOLxxxx\data_01\formats.sas7bcat@@\main\1  <03-Jul-03.10:43:59>
\proj_a\PROTOCOLxxxx\data_a\_vnslock.sas7bdat@@03-Jul.17:13.2377
\proj_a\PROTOCOLxxxx\data_a\a_ident.sas7bdat@@\main\2  <03-Jul-03.15:17:21>
\proj_a\PROTOCOLxxxx\data_a\formats.sas7bcat@@\main\1  <03-Jul-03.17:19:40>
\proj_a\PROTOCOLxxxx\data_a\TrakData.csv@@\main\2   <03-Jul-03.15:17:16>
\proj_a\PROTOCOLxxxx\data_a\vcontrol.sas7bdat@@\main\1  <01-Jul-03.17:37:26>
\proj_a\PROTOCOLxxxx\export\lst2rtf_bat@@\main\2      <03-Jul-03.17:12:27>
\proj_a\PROTOCOLxxxx\export\pgm_eff\x7107_7_01.rtf@@04-Jul.10:20.2402
\proj_a\PROTOCOLxxxx\pgm_a\_autorun.sas@@\main\4     <02-Jul-03.17:23:40>
\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.log@@04-Jul.10:20.2398
\proj_a\PROTOCOLxxxx\pgm_eff\l_eff1.sas@@\main\2     <03-Jul-03.15:19:07>
\proj_a\PROTOCOLxxxx\pgm_eff\x7107_7_01.lst@@04-Jul.10:20.2401
\proj_a\PROTOCOLxxxx\pgm_eff\x7107_7_01.tmp@@04-Jul.10:20.2400
\proj_a\PROTOCOLxxxx\util\_dropsuffix.sas@@\main\1   <30-Jun-03.18:06:35>
\proj_a\PROTOCOLxxxx\util\_last_in_path.sas@@\main\1  <30-Jun-03.18:07:45>
\proj_a\PROTOCOLxxxx\util\_setup.sas@@\main\2       <03-Jul-03.15:19:19>
\proj_a\PROTOCOLxxxx\util\_superscript.sas@@\main\1  <30-Jun-03.18:08:38>
\proj_a\PROTOCOLxxxx\util\fmtadlib.sas@@\main\1      <30-Jun-03.18:09:15>
\proj_a\PROTOCOLxxxx\util\formats.sas@@\main\1       <30-Jun-03.18:10:04>
\proj_a\PROTOCOLxxxx\util\make.csv@@\main\1          <02-Jul-03.16:31:52>
\proj_a\PROTOCOLxxxx\util\xcontrol.rmt.log@@04-Jul.10:20.2399
\proj_a\util\pgm\macro\ispgmvns.sas@@\main\1         <02-Jul-03.14:33:34>
\proj_a\util\pgm\macro\postrep.sas@@\main\2          <03-Jul-03.15:19:17>
\proj_a\util\pgm\macro\prerep.sas@@\main\2           <03-Jul-03.15:19:18>
\proj_a\util\pgm\macro\xcontrol.sas@@\main\1         <03-Jul-03.15:25:18>
\proj_a\util\pgm\macro\xgetctl.sas@@\main\1          <03-Jul-03.11:41:45>
\cc_tools\tools\bin\awk.exe@@\main\1                  <02-Jul-03.09:56:16>
\cc_tools\tools\bin\cygwin1.dll@@\main\1              <02-Jul-03.09:49:19>
\util\vns\data\template.sas7bitm@@\main\1             <27-Jun-03.11:13:06>
\util\vns\sasmacro\deldset.sas@@\main\1               <22-Jan-01.13:40:42>
\util\vns\sasmacro\dsetnobs.sas@@\main\1              <23-Nov-00.13:06:18>
\util\vns\sasmacro\lowcase.sas@@\main\1               <08-Aug-01.08:23:08>
\util\vns\sasmacro\try___r.sas@@\main\1               <08-Aug-01.07:03:10>
\util\vns\vnsmacro\maclist.sas@@\main\1               <24-Oct-01.12:37:20>
\util\vns\vnsmacro\patches.sas@@\main\1               <15-Feb-02.14:05:26>
\util\vns\usermacro\cnv2rtf@@\main\2                 <03-Jul-03.17:13:29>
\util\vns\usermacro\vnsinit.sas@@\main\3              <02-Jul-03.17:23:34>
-----------------------------
Variables and Options:
-----------------------------
CMD_PAR=cd m:/djg_view/proj_a/PROTOCOLxxxx/pgm_eff; sas.exe l_eff1.sas ; echo -
-----------------------------
Build Script:
-----------------------------
        cd m:/djg_view/proj_a/PROTOCOLxxxx/pgm_eff; sas.exe l_eff1.sas ; echo -
-----------------------------
cleartool>
```

Conclusion

Using make, and especially, clearmake, to automate production of of SAS Clinical report deliverables offers many advantages over current practices, especially for larger projects.

Acknowledgements

Thanks to the wonderful 'Unix Power tools’ for providing the original motivation to use make.