ABSTRACT
From a programming perspective, there are many datasets, summary tables, listings and graphs to produce in order to prove the safety and efficacy of a drug in a clinical study. We know that as the study progresses, the number of programs in the study increases proportionally. Within a clinical study, after a dataset program is updated, we then need to re-run that dataset program and all of the associated table, listing and graph (TLG) programs that depend on this particular dataset. In this way, we will update all of the TLG reports affected by the dataset changes. When done manually this process can be time consuming and error prone, but we can automate the process by using a Makefile.

To create a Makefile one has to follow strict syntax, which the make utility on UNIX can understand. This rigid syntax causes mistakes to become very hard to debug. Thus, one has to be very careful while creating and editing a Makefile. This paper will discuss how to automate making of a Makefile using a SAS ® program. This can save a lot of time and resources, and removes human errors caused by manual creation of Makefile.

This paper will introduce users to the basic options of the make facility, the syntax of Makefile, automating the making of a Makefile, and applying the Makefile in the clinical trials environment. The program discussed in this paper will create a Makefile by scanning a standard header of all SAS programs stored in a particular directory.

INTRODUCTION
A Makefile is actually a script file stored as ASCII text. It is in itself a program. This is similar to how a SAS program is stored as a set of instructions in a text file. The Makefile works with the make facility which acts upon the instructions specified within the Makefile. To invoke make facility "make" command is issued at Unix prompt. It has many options, which are specified through lettered parameters similar to other UNIX commands such as 'ls' or 'more'.

One of the main goals of the Makefile is to manage the dependencies between files such as SAS programs, output and datasets. It can execute commands such as submitting a SAS program. This method of refreshing output files can be triggered through the dependencies defined in the Makefile.

CLINICAL TRIALS EXAMPLE
A typical clinical trial is shown below in figure 1.
Assume that all of our programs (datasets and TLG) are stored in the \programs directory, Analysis datasets are stored in the \outdata directory, TLG reports are stored in the \results directory and datasets extracted from Oracle Clinical are stored in the \sasport directory as shown in figure 1. In our example, dataset programs use the data extracted from Oracle Clinical to output analysis datasets. These analysis datasets are used by TLG programs to create TLG reports for the study. Also in our example, we will assume that we have dataset programs d_stratvar.sas and d_ae.sas, which use the ocpinfo.sas7bdat and ocae.sas7bdat datasets respectively to create analysis datasets such as stratvar.sas7bdat and ae.sas7bdat. These analysis datasets are in turn used by TLG programs like t_ae.sas to create reports like t_ae_s.out and t_ae_f.out for our clinical trial.

CREATING A MAKEFILE MANUALLY

Let’s simplify the process of creating a Makefile by dividing it into three parts. The first part gives the order of execution by creating dummy targets and dependencies. Targets are desired outputs and dependencies are inputs that are required to create those targets (outputs). The second part of the Makefile will associate these dummy targets with actual outputs. The third part gives details on the relationships between the targets, dependencies and command in updating these targets. As a general rule of thumb, targets and dependencies are separated by a colon, and a backslash (\") symbolizes continuation of line.

![Figure 5: Example of three parts of the Makefile](image)

<table>
<thead>
<tr>
<th>First Part</th>
<th>Second Part</th>
<th>Third (TOP, MIDDLE &amp; BOTTOM) Part</th>
</tr>
</thead>
</table>
| system: \ \strata: \
  ../outdata/stratvar.sas7bdat |
| strata \ outdata \ tables |
| outdata: \ ../outdata/ae.sas7bdat |
| tables: \ ../results/t_ae_s.out \ ../results/t_ae_f.out |
| |
| d_ae.sas |
| sas9 d_ae |
| ../outdata/stratvar.sas7bdat: \ ../sasport/ocpinfo.sas7bdat \ d_stratvar.sas |
| sas9 d_stratvar |
| ../outdata/ae.sas7bdat: \ ../sasport/ocae.sas7bdat \ d_ae.sas |
| sas9 d_ae |
| ../results/t_ae_f.out \ ../results/t_ae_s.out: \ ../outdata/ae.sas7bdat \ ../outdata/stratvar.sas7bdat \ t_ae.sas |
| sas9 t_ae |

As shown in figure 5 (first part), system is a dummy target, and strata, outdata and tables are dummy dependencies. They are called dummy because we don’t have actual outputs or inputs with that name. These dummy dependencies will be executed in the order they are specified. So in our example, strata will be updated before outdata, and outdata will be updated before tables. In the second part, first-part dependencies will become targets, and actual outputs will become the dependencies for those targets. The third part of the Makefile can be divided in three parts itself. The top part contains the program outputs (also known as targets), the middle part shows the program inputs (also known as dependencies), and the bottom part displays the command used to run the program.

As shown in Figure 2, the top and middle portions are separated by a colon (:). Dependencies can be datasets, macros or the program name itself. This is followed by a tab character and the command used to execute this program, in our case it is “sas9 d_ae”. This has to be done for all the programs to be included in Makefile.

![Figure 2: Pseudo code and an Example for a portion of Makefile](image)

| Target: \
  Dependency1 \ Dependency2 \ Dependency3 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo Code</td>
<td>..outdata/ae.sas7bdat: \ ..sasport/ocae.sas7bdat \ ..outdata/stratvar.sas7bdat \ d_ae.sas</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>sas9 d_ae</td>
<td></td>
</tr>
</tbody>
</table>

Once we get all these commands in the Makefile, we can execute it and make will recursively check the modification times of the files; whenever a dependency becomes newer than the target (in other words, is modified), make will execute the program to update the target. In the example shown above, suppose our dataset program is updated. Make will then execute the dataset program, which will give us updated analysis datasets (AE). Make will then run all of the TLG programs that depend on this datasets, because the dependency for TLG using AE dataset has been updated.
To manually create a Makefile as described above, we will have to type in each and every command individually. Manual creation of Makefile is a time-consuming and error-prone process, which also has the potential for introducing human errors. Debugging a Makefile is not a user-friendly process. To find a forgotten colon or tab can take lot of time, depending on the size of the Makefile. Automating creation of a Makefile eliminates the need for multiple developers trying to access the Makefile to add their programs to it, and as a result, overwriting each other’s data. So to overcome all of these problems, let’s see how we can write a SAS program that will create a Makefile automatically.

CREATING A MAKEFILE USING A SAS PROGRAM

To create a Makefile using a SAS program, we will assume that each program in our study has a standard header as shown in figure 3. It is important to have the folder name and input/output names aligned properly, as column input is used to read these values. Also, we can specify whether we want this file to be included in the Makefile by specifying ‘yes’ or ‘no’ in front of the \texttt{make} option in our header. If a program is still being developed, or is an ad-hoc program, which is not part of the study, it can be omitted from the Makefile by specifying “no” in the include option of the header. The program to read this standard header is shown in the appendix.

| 1---+---| 10---+----| 20---+---- |
| **************************************************** |
| *** input :                                   |
| *** from: outdata: stratvar.sas7bdat          |
| *** ae.sas7bdat                               |
| *** output:                                   |
| *** to: results: t_ae_s.out                  |
| *** t_ae_f.out                                |
| *** include: make: yes                       |
| ****************************************************;

**SECTION LIBNAME IO**

1. The PIPE option in the filename statement retrieves the output from a UNIX \texttt{ls} or Windows \texttt{dir} command, which return the names of all SAS programs stored in the specified directory. The Makefile file reference points to the destination where the Makefile will be stored, which is the current directory from where the program is executed.

\begin{verbatim}
filename sasfiles pipe 'ls -1 *.sas';
filename Makefile "Makefile";
\end{verbatim}

2. The filename coming from the \texttt{ls/dir} command is going to have a variable-length size, so in order for us to read all records, we need to specify the TRUNCOVER option in our INFILE statement. The TRUNCOVER option tells SAS, in effect, to read whatever is there, even if the end of line (EOL) comes before it is expected. This will store a list of all SAS programs in the specified directory in our output dataset.

\begin{verbatim}
data all_files;
infile sasfiles truncover;
input pgmname $50.;
run;
\end{verbatim}

3. We then need to read the headers of all files, whose names are in the \texttt{ALL_FILES} dataset. As we are going to read multiple external files, we need to use the \texttt{FILEVAR} option in the second INFILE statement. As the value of \texttt{FILEVAR} variable changes, the file read by the \texttt{INPUT} statement changes. The delimiter is specified as “:\”, as we don’t want the input statement to consider the default space as a delimiter. A single “@” is used with the input statement to hold our input stream, and check if we are reading the header portion of the program. Once confirmed, the program reads the section, library name and input/output files for this program. This is done for all the programs in the specified directory.

\begin{verbatim}
invalue pgminfmt
"d_" = 1 "t_" = 2
"l_" = 3 "g_" = 4;
if pgmname ^= "d_stratvar.sas" then
pgmtype = input(substr(pgmname, 1,2), pgminfmt.);
else pgmtype = 0;
\end{verbatim}

4. A \texttt{PGMTYPE} (program type) variable is created, depicting whether a program outputs datasets, tables, listings or graphs. This variable is used to give the proper order to the Makefile, so that dataset programs are run before other programs. We get the value for this variable by applying the \texttt{PGMINFMT} informat to the first two letters of the program name. As the \texttt{STRATVAR} dataset has to be run before any other dataset, it is given a value of 0, so when the dataset is sorted by the \texttt{PGMTYPE} variable, it comes out at the top of the list.
5. After we read all headers for all programs, the following values are populated in our resulting datasets. Counter variable counts number of outputs and inputs used or created by each program.

<table>
<thead>
<tr>
<th>OBS</th>
<th>SECTION</th>
<th>LIBNAME</th>
<th>IO</th>
<th>PGMNAME</th>
<th>PGMTYPE</th>
<th>IOTYP</th>
<th>COUNTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>output</td>
<td>outdata</td>
<td>ae.sas7bdat</td>
<td>d_ae.sas</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>input</td>
<td>sasport</td>
<td>ocae.sas7bdat</td>
<td>d_ae.sas</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>input</td>
<td>outdata</td>
<td>stratvar.sas7bdat</td>
<td>d_ae.sas</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>output</td>
<td>outdata</td>
<td>stratvar.sas7bdat</td>
<td>d_stratvar.sas</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>input</td>
<td>sasport</td>
<td>ocpinfo.sas7bdat</td>
<td>d_stratvar.sas</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>output</td>
<td>results</td>
<td>t_ae_s.out</td>
<td>t_ae.sas</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>output</td>
<td>results</td>
<td>t_ae_f.out</td>
<td>t_ae.sas</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>input</td>
<td>outdata</td>
<td>stratvar.sas7bdat</td>
<td>t_ae.sas</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>input</td>
<td>outdata</td>
<td>ae.sas7bdat</td>
<td>t_ae.sas</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

6. Depending on the INCLUDE option specified in the header, only those programs which have “yes” specified (figure 3), are included in the Makefile, the others are dropped. A report is generated showing the files included and excluded, for documentation purposes.

7. We select distinct program types, assigning the ORD format (shown in the figure 5) to the program type, and concatenate a dummy system target to get the first part of Makefile. This gives the order of execution to the Makefile.

---

**Figure 4: Dataset with information about Target & Dependencies**

**Figure 6: Final MAKEFILE**

```plaintext
# SYSTEM DEFINITION ============================
system:   
  strata   
  outdata  
  tables

#STRATA DEFINITION ============================
strata:   
  ../outdata/stratvar.sas7bdat

#OUTDATA DEFINITION ==========================
outdata:   
  ../outdata/ae.sas7bdat

#TABLES DEFINITION ===========================
tables:   
  ../results/t_ae_s.out  
  ../results/t_ae_f.out

# DATASETS DEFINITION ==========================
../outdata/ae.sas7bdat:   
../sasport/ocae.sas7bdat  
../outdata/stratvar.sas7bdat  
  d_ae.sas
  sas9 d_ae

../outdata/stratvar.sas7bdat:   
../sasport/ocptinfo.sas7bdat  
  d_stratvar.sas
  sas9 d_stratvar

# TABLES DEFINITION ===========================
../results/t_ae_f.out  
../results/t_ae_s.out  
../outdata/ae.sas7bdat  
../outdata/stratvar.sas7bdat  
  t_ae.sas
  sas9 t_ae
```
8. The second part of the Makefile lists all outputs created by our programs, grouped by program type, as shown in figure 5. To get this list, we subset the dataset in figure 4 to records where SECTION = "output", and apply the ORD format to get the dummy targets.

9. The third part of the Makefile lists targets (top part) which we can get by sub-setting the data in figure 4 to records where SECTION = "output" condition, dependencies (middle part) can be gathered by sub-setting the dataset in figure 4 to records where SECTION = "input" condition. The command (bottom part) can be gathered by program name as shown in figure 5. Also as shown in Figure 5, the structure of the command line of this part is formed by a [TAB] character, followed by compiler name [SAS9], and that is followed by the program name without the ".SAS" extension. Also, the top and middle portion (target and dependencies) are separated by a colon (:). Union Operator is used to concatenate results from each query.

10. We concatenate all three parts to get our final Makefile, and use PUT statement to write it to a file. We also add blank spaces in-between different parts of Makefile to make it more readable. With few more data step manipulations we can add comments to Makefile, so that it is easy to understand.

APPLYING A MAKEFILE
The default method of applying the make facility, uses the make command, which expects the file to be named Makefile. An alternative method is to issue the make command followed by the -f option and an alternative filename for example “make –f mymake.mk”. Option –i ignores error codes returned by commands and is equivalent to the special-function target `.IGNORE:' for example “make –if mymake.mk” will not stop execution of Makefile if an error is encountered. For more information on make options check UNIX man pages.

CONCLUSION
Make requires us to manually track all dependencies between files in the project. This process is error-prone, since a forgotten or an extra dependency might not be immediately obvious, but instead surfaces as subtle bugs in the software. For example, the syntax used by Make includes the use of tab, a whitespace character; however, many editors do not provide very clear visual clues to the presence of tabs rather than spaces, and tab characters are not represented uniformly across editors in any case, with size varying from as little as 2 spaces to 8 spaces. Automating the creation of the Makefile eliminates the need for multiple developers trying to access the Makefile, to add their programs to it, and as a result, overwriting each others data. This can be done by assigning read-write-execute permissions for the SAS program that creates Makefile to a single person on a UNIX system. Thus, by automating the creation of the Makefile, we can save lot of precious time and resources.

REFERENCES
• http://en.wikipedia.org/wiki/Make_(software)
• Managing Projects with Make - O'Reilly
• Unix `make` man page
• SAS ® Documentation.

ACKNOWLEDGEMENT
I would like to thank my manager, Janet Humphrey, for generously providing me with the time to write this paper, I wish to thank Chris Hurley and Nancy Brucken for reviewing my paper and providing well informed suggestions. I wish to thank all of my colleagues who helped me with this paper.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Vikas D Gaddu
I3 Statprobe
2424 Harrodsburg Rd,
Suite 300.
Lexington, KY 40503
Work Phone: 859-825-6957
Fax: 859-825 6979
Email: vikas.gaddu@i3statprobe.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

APPENDIX: SAS PROGRAM TO CREATE MAKEFILE
/*For Windows user*/
/*filename sasfiles pipe 'dir c:\sas\*.sas /b';
filename Makefile "c:\sas\make\Makefile"; */
/*For Unix users*/
filename sasfiles pipe 'ls -l *.sas';
filename Makefile "Makefile";
/*Formats are used to arrange data in proper sequence as required by Makefile syntax*/
proc format;
invalue pgminfmt
    "d_" = 1 "t_" = 2
    "l_" = 3 "g_" = 4;
value ord
    -1 = "system"
    0 = "strata" 1 = "outdata"
    2 = "tables" 3 = "listings"
    4 = "graphs";
invalue iotyp
    "include" = 0 "output" = 1
    "input" = 2;
run;
/*Read the names of all files returned by ls/dir command in a dataset variable*/
data all_files;
    infile sasfiles truncover;
    input pgmname $50.;
run;
/*Read all the headers files of all *.sas files that are in the stored in the specified directory into an dataset*/
data trg_dpd;
    retain sec lib;
    length section $10 libname $11 io $35;
    set all_files;
    /* For Windows users*/
    *file2read = "c:\sas" || pgmname;
    /*For UNIX users*/
    file2read = pgmname;
    infile dummy filevar = file2read dlm = ":" end = done pad missover;
do while(not done);
    sec = section; lib = libname;
    input hmarker $ 1-3 @;
    if hmarker = "***":
        input section $ 4-12 libname $ 13-22 io $23-56 ;
        if section = " " or lowcase(section) in ("from:", "to:") then
            section = sec;
        if libname = " " then libname = lib;
        section = trim(lowcase(tranwrd(section, ":", " ")));
        libname = trim(lowcase(tranwrd(libname, ":", " ")));
        if io ^= " " then io = trim(lowcase(io));
        if pgmname ^= "d_stratvar.sas" then
            pgmtype = input(substr(pgmname, 1,2), pgminfmt.);
        else pgmtype = 0;
        if io ^= " " and index(io, "***")=0
            and section in ("input", "output", "include") and
            libname in ("outdata", "results", "sasport", "rawdata", "make")
            then do;
                if section ^= " " then iotyp = input(section, iotyp.);
                output;
            end;
end; *end of do loop;
drop hmarker sec lib;
run;
proc sort data = trg_dpd ;
by pgmname iotyp;
run;

data trg_count;
set trg_dpd;
by pgmname iotyp;
if first.iotyp then counter = 0;
else counter+1;
run;

proc sql;
create table trg as
select *
from trg_count as o
where section ^= "include"
and exists ( select io
from trg_dpd as i
where io = "yes"
and o.pgmname = i.pgmname);
title "Following Files will be added in makefile";
select distinct pgmname from trg;
title "Following Files will not be added in makefile";
select pgmname from trg_dpd where io = "no";
quit;

/* Gives execution order to the makefile and forms First part of Makefile */
proc sql;
create table firstpart as
select put(-1,ord.) || ":" as trg length = 150, -1 as pgmtype from trg
union
select distinct put(pgmtype, ord.) || "\" as trg length = 150, pgmtype from trg
order by pgmtype;
quit;

/* Adding dummy target to execution order dummy dependencies */
data firstpart;
set firstpart end = eof;
by pgmtype;
if eof then trg = tranwrd(trg,"\", "");
keep trg;
run;

/* List all the targets for all programs and forms second part of Makefile */
proc sql;
create table output as
select ../" || trim(libname) || "/" || trim(io) || "\" as trg length = 150,
pgmtype, pgmname ,2 as part
from trg
where section = "output"
order by pgmtype, pgmname;
quit;

/* Adding dummy tags to create Second part of the Makefile */
data secondpart;
set output;
by pgmtype pgmname;
if last.pgmname then trg = tranwrd(trg,"\", ");
output;
if first.pgmtype then do;
    trg = trim(put(pgmtype,ord.)) || ":\n";
    pgmtype = pgmtype - 0.5;
output;
end;
run;
proc sort data = secondpart;
    by pgmtype;
run;
/ *
To Create third part of Makefile we concatenate results from following queries
and group them
by program name
1) All outputs created by a program, last output is concatenated by ":\n"
2) All inputs needed by a program
3) Program that creates output (listed in 1) are also listed as dependency
4) Command to update the outputs.
*/
proc sql;
create table thirdpart as
    select "./" || trim(libname) || "/" || trim(io) ||
        case
            when counter = max(counter) then ":\n"
        else "\n"
        end as trg length = 150 , pgmtype, pgmname, 1 as cmd,counter
    from trg
    where section = "output"
    group by pgmname
union
    select "./" || trim(libname) || "/" || trim(io) || "\" as trg length = 150,
    pgmtype, pgmname , 2 as cmd, counter
    from trg
    where section = "input"
union
    select distinct pgmname as trg length = 150, pgmtype, pgmname, 3 as cmd, 0 as counter
    from trg
union
    select '09'x || "sas9 " || tranwrd(pgmname,".sas"," ") as trg length = 150,
    pgmtype, pgmname, 4 as cmd, 0 as counter
    from trg
    order by pgmtype, pgmname, cmd, counter;
quit;
/* Dummy dataset to give bland spaces between each section of Makefile */
data dummy;
    length trg $150;
    trg = " ";
run;
/* Concatenating First, second and third part of Makefile to create final
Makefile */
data make;
    set firstpart(keep = trg) dummy secondpart(keep = trg) dummy thirdpart(keep = trg);
    file Makefile;
    put trg;
run;