PharmaSUG 2015 - Paper AD09

The Dependency Mapper: How to save time on changes post database lock

Apoorva Joshi, Biogen, Cambridge, MA
Shailendra Phadke, Eliassen Group, Wakefield, MA

ABSTRACT

We lay emphasis on building a perfect roadmap to database lock. To name a few, we keep all stake-holders in-sync, nail down the specs and simulate dry-runs pre-database lock. This helps us produce the final deliverable such as tables, listings, figures and data sets in a seamless fashion post database lock. In our race to create these deliverables in record time, we usually forget to account for one aspect – human error.

Changes to any part of the deliverable post database lock can be frustrating and could potentially delay timelines. What could be the impact of a small change, such as correcting a label typo in ADLB to the impact of something more severe, such as a new derivation in ADSL? By creating a dependency map – an extensive map of data sources and their dependent programs – one can greatly reduce time consumed on changes post database lock. This paper explores the various advantages of creating a dependency map and explores different methods to create such maps programmatically. With sample user cases the various time saving advantages of dependency maps are explained in detail. While detailed code is not provided in this paper, important snippets of code necessary for someone to get started are provided.

INTRODUCTION

This paper provides a simple approach to many a time complicated process of identifying and managing changes post database lock. Before jumping in, the reader would do well to familiarize himself with database lock procedures and have some familiarity with SAS® programming, an understanding of SDTM and ADaM data sets and how tables, listings and figures reference these data sources. The following figure gives a high level overview of the basic concept of this paper

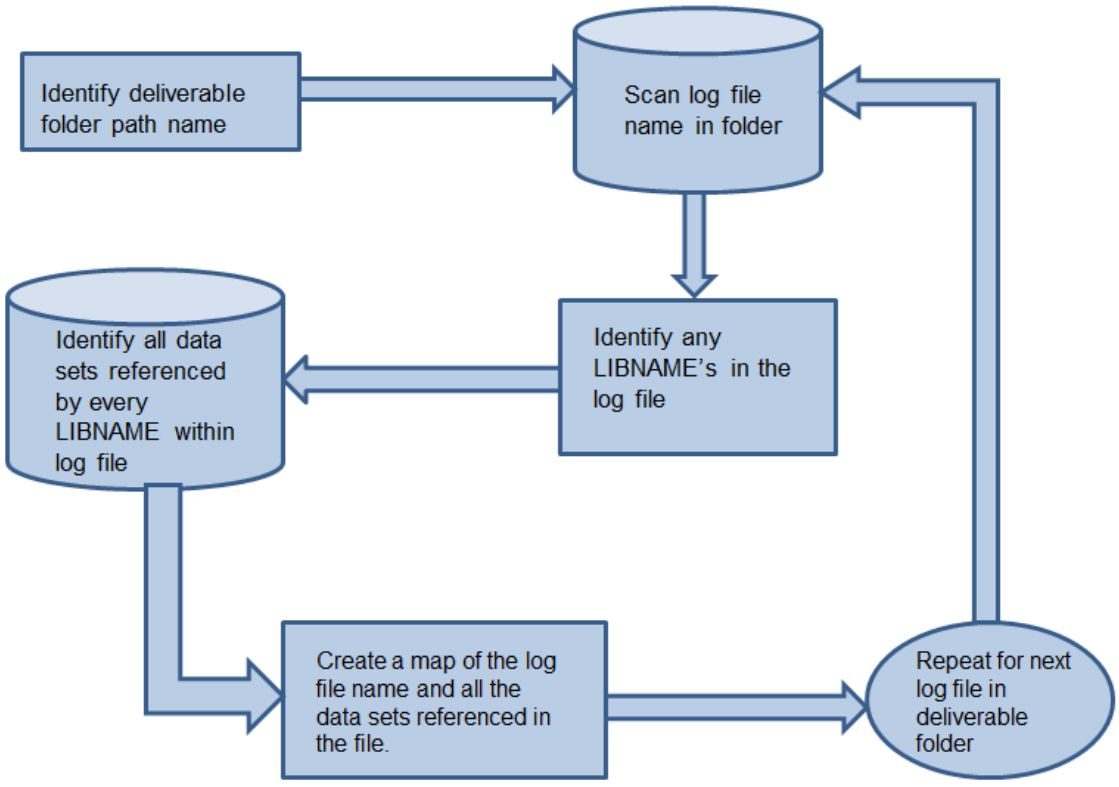

Figure 1. Basic Concept

THE BASIC CONCEPT

Imagine a 100% validated deliverable comprised of four ADaM data sets and hundred tables, listings and figures (TLF) as shown in Figure 2.

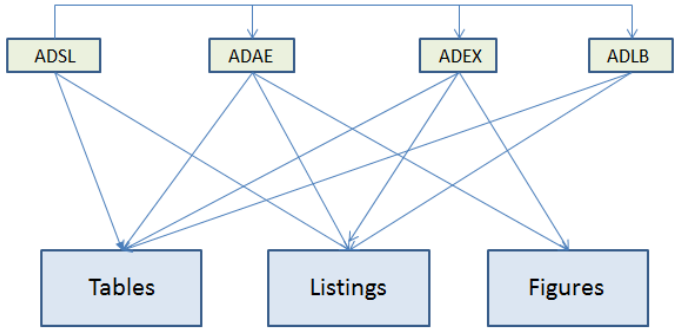

Figure 2. Dependencies within Deliverable

Each data set references ADSL and various TLF's reference various ADaM data sets, creating a web of dependencies. We asked a random sample of programmers and statisticians how much time it would take to rerun and QC the above deliverable, assuming no data changes. We received answers from 2 hours to 2 days, based on type of QC techniques involved.

NOTHING IS CONSTANT, BUT CHANGE.

Once database locks, we rerun and QC our deliverable and everything looks great. Production and QC programmers high five each other and are ready to go home by 5 in the evening. Welcome to our imaginary world.

Raise your hand if you have ever been in the following situation. Database has locked, data sets have passed QC, TLF's have been run and QC passed and sent over for senior review. Then you receive an email saying a variable derivation needs to be changed in ADLB because of a data issue which was not identified pre-database-lock. While the programming for this derivation modification might not be time consuming, the ordeal that follows is every programmer's nightmare.

Once data sets have been updated – and usually one would like to rerun all data sets to maintain similar timestamps – all the TLF's need to be updated and passed through QC. We asked the same set of statisticians and programmers to estimate time taken to modify, rerun and QC the deliverable after the change to ADLB. The two hours – two days range had changed to two days to a week. Most of the pool we interviewed believed the entire deliverable should be rerun and all TLF's should be QC passed.

This got us thinking that any change post database lock is beyond our control. What can be controlled are which parts of the deliverable need to be passed through QC and which can purely be rerun without worrying about the change to ADLB affecting them. If out of the 100 TLF's, only 15 reference the ADLB data set, why should we be worried about checking the remaining 85?

DEPENDENCY MAPPING

A dependency mapper identifies dependencies within a deliverable, namely the various data sets and output TLFs. If we can get a comprehensive map identifying the dependency between every TLF and data set, we could limit our QC activities to programs which reference ADLB. Many work under the assumption that a change in one data set requires QC of the entire deliverable. But if one can prove that the change affects only a subset of the deliverable we could avoid QC of the entire deliverable and cut down turn-around time.

We explored various methods of dependency mapping as listed below

1. Build a parser which scans the log files for any library name (LIBNAME) references and identify data sets which are referenced through these libraries.
2. A slight variation of the above method. Create a macro which is used in the programs to read any data set. Then create a parser which scans for certain keywords which are displayed in the log each time the macro is invoked. Text which follows these keywords is the data set name and will be read in by the parser to create a dependency map.
3. Manually scan through each program and create a list of referenced data sets.

4. Ask programmers to keep track of data sets referenced as they program an output and create a consolidated list of programs and the data sets referenced.

Each of the options above has its pros and cons. The third option is extremely labor intensive, prone to human error and not worth spending any time considering. The fourth option, while quick and requiring least amount of application development, is highly prone to human error and could turn out to be a maintenance nightmare. The second option is close to fool-proof but might be prone to human error (like directly referencing the data set without the macro call). The first option is fool-proof but requires taking care of large number of permutations and combinations during application development. While a programmer might choose any of the techniques above, we decided to focus our efforts on the first option.

IMPLEMENTATION

Discussed below is the implementation of the first method, which is, build a parser which scans the log files for any LIBNAME references and identifies data sets which are referenced through these libraries. We develop a macro with an option to input the following parameters

a. Path name of the deliverable location for which you want to check for data dependencies. It is important that the log files of the programs are present in this location.
b. Path name of the folder location where you want to save the output excel file displaying all data dependencies.
c. List of different libraries from where data is being used. This particular option is optional as the program will automatically read all library references.
d. List of the outputs (data sets and TLFs) for which you want to check the data dependencies. This is option but can come in handy when one would want to check a specific set of the deliverable.

ACCESS INPUT FOLDER AND CREATE LIST OF SAS LOG FILES

Using the following code, we access the input folder defined in the macro call and read in the log file names.

```
DATA dsn(keep=nfile nnext);
       /*inpath is the macro parameter with the path of the deliverable
       Assign filref for the directory given by &inpath*/
       rc=FILENAME('dir',"&inpath");
       dirid=DOPEN('dir');
       /*numsel will have number of members in a directory*/
       numsel=DNUM(dirid);
       DO i=1 TO numsel;
              /*nlst will have each member name with the extension*/
              nlst=DREAD(dirid,i);
              nfile=SCAN(nlst, 1, '.');
              nnext=UPCASE(scan(nlst, 2, '.'));
              /*if the member is log file then output*/
              IF INDEXW(upcase("log"),upcase(nnext)) THEN OUTPUT;
       END;
      /*after each log file is output in dsn then close the directory*/
      rc=DCLOSE(dirid);
RUN;
```

As an example, to better explain the output of the above code snippet, if the deliverable folder contains few SDTM program log files, ADaM program log files and table program log files, the "dsn" data set will be as follows.

Nfile	nnext
DM	LOG
AE	LOG
ADSL	LOG
ADAE	LOG
T-DM-DEMOG	LOG
T-AE-SOC	LOG
T-AE-SOC-PT	LOG

Display 1: Contents of DSN Data Set

Using PROC SQL save the `nfile` variable values in a macro variable separated by "|" and scan each value and use it in the INFILE statement to read the log file.

```
PROC SQL noprint;
       SELECT nfile INTO: prgname separated by '|' FROM dsn;
       SELECT count(DISTINCT nfile) into :cnt FROM dsn;
quit;
```

The prgname macro variable will have the value as shown below in Output 1

```
DM|AE|ADSL|ADAE|T-DM-DEMOG|T-AE-SOC|T-AE-SOC-PT
```

Output 1: Contents of prgname Macro Variable

The cnt macro variable will have the value 7. The following code will select each log file and read it into a SAS data set using INFILE statement.

```
%DO i = 1 %TO &cnt;
%LET log = %SCAN(&prgname,&i,'|');
%LET logfile = &inpath./&log..log;

DATA logscan_&i.;
       INFILE "&logfile." DSD;
       LENGTH item $500;
       INPUT item;
RUN;
%END;
```

SCAN LOG FILES TO CHECK FOR DEPENDENCIES

Before scanning the log file we should know the different library references used for the input data sets. If each programmer defines his own library references, we can first scan for the note generated by the LIBNAME statement to check for the defined library references.

When a library reference is defined (ex: `libname LIBDEMO "/server/drug/study/deliverable"`), the following note will be seen in the log.

```
NOTE: Libref LIBDEMO was successfully assigned as follows:
```

Output 2: LIBNAME Statement Resolved in log.

By using a combination of the INDEX and SCAN function we can capture each library reference.

```
IF INDEX(item,"NOTE: libref") THEN DO;
       ref = SCAN(item,3); OUTPUT;
END;
```

Once we know the library references, use the INDEX function to find the string containing each library reference in the log and capture the substring containing the library reference followed by the data set name. This will be data set which is being used and should be stored as a reference. For example in the log file you might have the following note.

```
NOTE: There were 160 observations read from the data set LIBDEMO.ADSL.
```

Output 3: DATA Statement Resolved in log.

```
IF INDEX(STRIP(ref)||'.',item) THEN DO;
    dataused = SCAN(SUBSTR(item,INDEX(UPCASE(item), STRIP(ref)||'.')),1,' ');
   OUTPUT;
END;
```

Above statement will first check if “LIBDEMO.” string is present in the log and will substring LIBDEMO.ADSL in `dataused` variable. LIBDEMO.ADSL is the data set which is being referenced.

Once each log file is scanned for all the library references, the data sets referenced are output in a SAS data set. The output data set can be identified into analysis data sets, tables, listings and graph data sets for better visual representation.

DISPLAY OF FINAL RESULTS

The final results are displayed in an excel file as shown below in the sample output below shown in Display 2.

Date when Table program was executed	Table	SDTM/ADAM/Other Data referenced
Nov 4 14:07	T-AE-SOC-PT	LIBDEMO.ADAE , LIBDEMO.ADSL
Nov 4 14:07	T-AE-SOC	LIBDEMO.ADAE, LIBDEMO.ADSL
Nov 4 14:49	T-DEMOG	LIBDEMO.ADSL
Nov 4 14:07	ADSL	LIBDEMO.DM
Nov 6 11:01	ADAE	LIBDEMO.ADSL , LIBDEMO.AE

Display 2: Sample Output Identifying Dependencies for the Deliverable

You can also display the date and time when each output was created to check if the order of execution was correct. In the above example ADAE was executed after the tables which would be in violation of standard QA practices.

CONCLUSION

Dependency mapper provides a distinct time saving advantage when we have to rerun and QC changes post database lock. It can also provide additional checks such as ensuring all data is being mapped, chronological check of data and TLF reruns and supporting documentation for program TOC.

ACKNOWLEDGMENTS

The authors would like to thank Daniel Boisvert and Matthew Carlson from Biogen for their time and input in brainstorming the idea of a dependency mapper and providing the required support in implementation of this concept.

CONTACT INFORMATION

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

Apoorva Joshi
Sr. Analyst III, Statistical Data Analytics
Biogen
300 Binney St, Suite 94W26
Cambridge, MA 02142
Work Phone: 617.914.6723
E-mail: apoorva.joshi@biogen.com

Shailendra Phadke
Statistical Programming Contractor
Eliassen Group
30 Audubon Rd,
Wakefield, MA 01880
Work Phone: 617.914.4132
E-mail: sphadke@eliassen.com