PhUSE 2008

Paper DM05

Migration of clinical data from IBM MVS to HP Unix

Fronke Gerken, Boehringer Ingelheim, Ingelheim, Germany

ABSTRACT

The harmonisation of the clinical data environment established HP with UNIX (HP-UX) as the common base at Boehringer Ingelheim. Therefore the migration of approximately 20 GB data from IBM MVS was necessary. Data were stored in SAS® data libraries, programs either in SAS data libraries or in operating system files. SAS data libraries were to be transformed into a transport file format using PROC CPORT. The transport files were to be copied from the MVS to the UNIX environment using an FTP program. On the UNIX side the transport files had to be imported to SAS data libraries using PROC CIMPORT.

INTRODUCTION

Since the early 1980s biometrics and data management (BDM) at Boehringer Ingelheim's (BI) location in Ingelheim, Germany ran SAS software on IBM mainframe with operating system MVS for nearly 20 years. Until the end of 1998 clinical data were stored in SAS version 6 data sets organised in SAS data libraries. A SAS data library was stored in one IBM MVS file. Programs were stored either in SAS catalogs, also organised in SAS libraries, or directly in operating system text files. Output for clinical trial reports were stored in SAS catalogs. The SAS software could be used interactively under the Time Sharing Options (TSO) subsystem as well as in batch mode, with jobs written in Job Control Language (JCL). Users were able to log on only from 5:30 am to 10:00 pm. After 10:00 pm the system administrators ran jobs like backup procedures.

The harmonisation of the clinical data environment across all BI BDM locations established UNIX on HP machines (HP-UX) as the common platform at BI. Therefore the migration of approximately 20 GB of data and programs from the IBM machine to the HP machine was necessary. Both machines were connected by a permanent line. This paper describes the planning, programming and conducting issues of this migration project.

REQUIREMENTS

No project without requirements:

1. All data were to be completely transferred in a documented and validated way.
2. The production, statistician's and data manager's work, was not to be interrupted by migration activities.
3. It was to make sure that after the transfer of an object (a SAS data set or a program) no BDM user was able to make changes to this object on the IBM machine any more.
4. The transferred programs should produce the same results on the HP machine as on the IBM machine and the stored output should look the same if routed to a printer after the migration as if routed to a printer before the migration.

SPECIFICATIONS

The requirements were translated into the following specifications.

To fulfill the first requirement, it was looked after tools that were needed to perform the task. On the IBM machine, the SAS CPORT procedure and on the HP machine the SAS CIMPORT procedure were to be used. As communications software a program called FTP was to be used, which was able to copy a file from the IBM machine to the HP machine via the permanent line. To check the completeness a list of all files used by the BDM users was to be produced.

The IBM MVS operating system did not allow the BDM users to work later than 10:00 pm in the evening. To fulfill the second and third requirement, it was specified that the data were to be transferred during the night and that the write access rights were to be revoked thereafter.

Only a time slot of 3.5 hours, from the termination of back-up procedures on the IBM machine (mostly 2:00 am) until the start of the interactive user mode (5:30 am), was made available by the IBM system management. But he whole amount of 20 GB of data and programs was too much to be transferred during one night. Therefore it was necessary to split the whole migration into several transfer units which could each completely be executed during one night. To be able to put IBM MVS files of suitable size together for each transfer unit, it was decided to build up a SAS data set that worked as a database for IBM MVS

files storing the following meta data: name, type (SAS data library, text file), size, and the users who were working on it. This database then was to be used
- to select the IBM MVS files for a transfer unit in one night
- to choose the suitable transfer options (ascii, binary) dependent on the type
- to translate the hierarchical structure as defined in the name of the IBM MVS files into the directory structure in UNIX, and
- to contact the responsible users in case of problems and to inform them about the completed transfer.

A transfer unit consisted of several transfer jobs, one individual transfer job for each IBM MVS file.

A check was to be made as to whether an individual transfer job had run successfully, firstly by reference to the log files, and secondly in the case of transfer of a SAS library by the use of a special validation job. This involved the following seven steps.
1. All SAS data sets of each SAS data library to be transferred were to be submitted to PROC CONTENTS on the IBM machine.
2. The PROC CONTENTS output was to be written into a SAS data set that also had to be transferred from the IBM machine to the HP machine.
3. On the HP machine, all SAS data sets of the recreated SAS data library were to be submitted to PROC CONTENTS.
4. The PROC CONTENTS output was to be written into a SAS data set.
5. Both, the SAS data sets created by PROC CONTENTS on the IBM machine and on the HP machine, were to be electronically compared using PROC COMPARE.
6. A comparison outcome of "all values equal" was to be taken as sufficient indication that the entire SAS data library has been successfully transferred.

The fourth requirement led to the study of character sets and translation tables. A translation table had to be built up for usage by the SAS procedures CPORT and CIMPORT.

PROGRAMMING ISSUES

Some aspects, which can play also a role in other than migration projects, are presented in this section.

INPUT PROGRAM

The use of INPUT statements within SAS data steps to read formatted text files is well known. Unusual it is to read the output of operating system commands.

In order to build up the SAS data set containing the database of the IBM MVS files to be transferred, a list of the relevant IBM MVS files was created using the IBM MVS command LISTD. Unfortunately, a direct interface to the SAS system was not available, by which a SAS data set could be written with the LISTD output. So the output of a set of LISTD commands was routed to a text file on the IBM machine that was read in by a SAS program.

The following is an excerpt of the LISTD output. A line with three points (...) means that either lines are skipped because of non relevance or repeated text of the same structure as before:

```
...
READY
  LISTD 'BI197' LEVEL
BI197.PO.PROG.TEMP132
--RECFM-LRECL-BLKSIZE-DSORG
  FB    132   5148    PO
--VOLUMES--
  BIKB03
BI197.PO.PROG.TEMP80
--RECFM-LRECL-BLKSIZE-DSORG
  FB    132   5148    PO
--VOLUMES--
  BIKB03
...
READY
  LISTD 'BIB21' LEVEL
...
READY
END
```

The structure looks easy to survey. After a "READY" line, a line repeating the command was given, in this case "LISTD 'BI197' LEVEL". The term "BI197" was a userid and according to a company rule all file names had to start with the userid. The keyword "LEVEL" forced IBM MVS to list all files starting with "BI197" and therefore all files of the user "BI197" were listed.

The first file had the name "BI197.PO.PROG.TEMP132". Usually five lines were given per file.
1. the file name ("BI197.PO.PROG.TEMP132"),
2. constant text to describe the values in the third line ("--RECFM-LRECL-BLKSIZE-DSORG"),
3. values from which type and file size could be determined ("FB"/132/5148/"PO"),
4. constant text to describe the values in the fifth line ("--VOLUMES--"),
5. name of the volume where the file was stored.

One LISTD command produced output for at least one IBM MVS file. Each LISTD command ended with a "READY" line, followed by a new LISTD command or by an "END" line.

The LISTD output consisted of lines of different length. The possibility to read lines with varying length was very helpful:

```
INFILE file-specification LENGTH=linelen;
LENGTH line $ 132;
INPUT line $VARYING132. linelen;
...
```

The INFILE statement forces SAS to write the length of a line into the variable specified by the LENGTH= option whenever a new line is read from the file. The $VARYING format uses the value in the variable linelen to stop reading at the end of the line.

The first lines of the LISTD output were out of interest. Their interpretation was skipped in the following program lines:

```
* It was sure that the first line of the LISTD output was not equal to 'READY'.;
DO UNTIL (line='READY');
  INPUT line $VARYING132. linelen;     * Reads the line but does nothing.;
END;
* Variable LINE contains the value 'READY';
```

From here on to the end of the LISTD output all lines were of interest. The program follows the structure seen in the LISTD output. Since all further program lines were within a DO WHILE loop, further explanation of the program is presented as a SAS comment.

```
* The next line contains either the value 'END' or a LISTD command.;
INPUT line $VARYING132. linelen;

DO WHILE (line NE 'END');
  * The current line contains a LISTD command.;
  * Therefore, a so-called LISTD block follows.  A LISTD block contains information about;
  *    IBM MVS files or is empty.  A LISTD block is closed by a "READY" line.;

  * The next line contains either the value 'READY' or the name of an IBM MVS file.;
  INPUT line $VARYING132. linelen;

  DO WHILE (line NE 'READY');
    * The current line contains the name of an IBM MVS file.;
    * Therefore, a so-called file information block follows.  A file information block;
    *    contains information about one IBM MVS file.  A file information block is closed;
    *    by the name of the next IBM MVS file or by the end of the superior block, in;
    *    this case a "READY" line.;

    * Input of all information about the current IBM MVS file.;
    * The current line contains the name of an IBM MVS file.;
    dsname=SUBSTR(line,1,linelen);

    * The next line contains the constant text '--RECFM-LRECL-BLKSIZE-DSORG'.;
    INPUT line $VARYING132. linelen;

    * The next line contains the wanted values for the IBM MVS file.;
    INPUT line $VARYING132. linelen;
    recfm  =SUBSTR(line, 3, 5);
    lrecl  =SUBSTR(line, 9, 5);
    blksize=SUBSTR(line,15, 7);
    dsorg  =SUBSTR(line,23, 5);
```

```
      * The next line contains the constant text '--VOLUMES--'.;
      INPUT line $VARYING132. linelen;

      * The next line contains the name of the volume on which the IBM MVS file is stored.;
      INPUT line $VARYING132. linelen;
      volume=SUBSTR(line,3,7);

      * The information of one IBM MVS file is completely read.;
      OUTPUT;

      * The next line contains either the value 'READY' or the name of an IBM MVS file.;
      INPUT line $VARYING132. linelen;
    END;

    * The next line contains either the value 'END' or a LISTD command.;
    INPUT line $VARYING132. linelen;
  END;
```

Any structure in the input data can be read by a SAS data step. Of course, the complexity of a SAS data step grows with the complexity of the structure in the input data.

CHARACTER SETS AND TRANSLATION TABLES

The BDM users had used special characters in their programs in order to improve the readability of the output on paper. For example,

```
put 132*'BF'X;
```

forced the IBM MVS printer to print out a solid line. The output of a single program, which was part of a package for a study, was stored in a SAS catalog. From there all output entries were routed to a printer to produce final tables and listings for a clinical trial report.

The fourth requirement was that in the example above also a solid line was printed if the transferred output entry was routed to a HP-UX printer. This condition required the development of an individual translation table.

Data to be transferred consisted of numerical and character values in SAS data sets and program and output text in SAS catalog entries or external text files. Generally, programs are written with characters out of the ASCII character set. On the IBM machine the IBM specific character set EBCDIC with some extensions was used. On the HP machine the ASCII character set with some extensions was used. The FTP program took this fact into account and provided an option for code conversion from EBCDIC to ASCII during the transfer operations. This option is called the ascii mode. Text files could be directly transferred using FTP in ascii mode. Within the ascii mode, FTP transformed each incoming code to an outgoing code according to a translation table. The translation table guaranteed that each incoming EBCDIC code was transformed to its logical representation as ASCII code. For example, the incoming EBCDIC code for the character 'A' ('C1'X) was transformed to the ASCII code for the character 'A' ('41'X).

The SAS system provides its own transport format for most of its objects. The CPORT procedure writes SAS data sets, SAS catalogs, or SAS data libraries to files in the transport format, while converting characters from the entire character set - in our case EBCDIC - to a SAS specific character set. The FTP program had to transfer the SAS transport file byte for byte without any changes or conversions. The option provided for this FTP behavior is called the binary mode. On the receiving machine - in our case the HP machine - the CIMPORT procedure read the SAS transport file and extracted the SAS data sets, SAS catalogs, or SAS data libraries converting characters from the SAS specific character set to the character set of the target machine - in our case ASCII. Similarly to FTP, the SAS system used translation tables on both, the IBM and the HP machine.

The IBM machine as well as the HP machine used different extensions of the character sets for national characters and for screen or printer presentation. The default translation tables used by FTP and by the SAS system supported only the transfer from one particular extended character set on the IBM machine to another particular extended character set on the HP machine. Unfortunately, these extended character sets supported by default were not those, which were used by the BDM users. Therefore an own translation table was defined.

In a first step a table was produced by a SAS program on the IBM machine with all possible character set codes in one column and the corresponding character beside in another column and printed out. In a second step the same was done on the HP machine. Thirdly both tables were merged by the characters. The column with the IBM MVS character set codes and the column with the HP-UX character set codes formed the logical translation table.

The physical translation table was created using the TRANTAB procedure. The SAS system stored the translation table as an entry of the entrytype TRANTAB in the SAS catalog SASUSER.PROFILE. The stored translation table was used by the CPORT procedure. In a similar way a translation table was created for FTP.

The development and the application of the translation tables ensured that all characters existing within the IBM MVS files were correctly migrated to HP machine.

AVOID NUMERICAL PROBLEMS BY NEWLY ROUNDING

Also danger lurked on the side of the precision of numerical values. A 2.1 on the IBM machine was not the same than a 2.1 on the HP machine.

A value of 2.1 does not have the same internal expression over all machines. The IBM machine value of 2.1 was approximated to a value of 2.1+x in the expression made by the CPORT procedure. This value 2.1+x was transferred without changes but then approximated again to a HP machine value of 2.1+x. Although the deviation was small, statements like

```
IF x=2.1 THEN ...;
```

would have failed. Therefore all numerical values had to be newly rounded.

Following macro was applied to each transferred SAS data set.

```
%MACRO round (dsname);
  DATA &dsname.;
    SET &dsname.;

    * The keyword _NUMERIC_ allowed to address all numeric variables automatically;
    *    without any naming conflicts with already existing variables.;
    ARRAY num _numeric_;

    DO OVER num;
      * The DO OVER statement created the automatic variable _i_.;
      IF num(_i_) NE . THEN DO;
        * Rounding only in the case that the value is non-missing.;

        * It was to be found out at which decimal place to round.  Therefore a;
        *   "titration" took place by multiplying the number with the powers;
        *   of ten and comparing the product with the result of the INT function applied;
        *   to this product;
        * An index variable for the exponentials was needed, but it was to avoid to;
        *   define a new permanent variable.  Therefore the automatic variable _n_ which,;
        *   was not needed, was misappropriated.;
        _n_=0;

        * There were not more than four decimal digits in the data.;
        DO WHILE (ABS(10**_n_*num(_i_)-INT(ABS(10**_n_*num(_i_)))/10**_n_ > 10**-4);
          _n_=_n_+1;
        END;

        num(_i_)=ROUND(num(_i_),10**-_n_);
      END;
    END;
  RUN;
%MEND  round;
```

CONDUCTING ISSUES

Generally, an individual transfer job consisted of three steps. The first transfer step was to create a SAS transport file for each SAS data library on the IBM machine using PROC CPORT. The second step was to transfer the SAS transport file in the binary mode from the IBM machine to the HP machine using FTP. The third step was to recreate the SAS data libraries from the SAS transport file on the HP machine using PROC CIMPORT. For text files only the second step was necessary. They could directly be transferred in the ascii mode by FTP.

Since the transfer options depended on the type of the IBM MVS files, the whole transfer was organised on the basis of one transfer job per operating system file. The transfer jobs themselves were created as output of a SAS program using the database of the files to be transferred as input. The output consisted of SAS statements to create the SAS transport files, FTP

statements to create new UNIX directories and to transfer the transport files, log-on statements to the HP machine and start of a SAS batch program on the HP machine, and JCL statements around each of the steps.

All jobs were written by the author as a member of the BDM group, but he was not allowed to run them over night. Only the system administrators had the sufficient access rights to the over night batch queue. Therefore an arrangement was made with the system administrators to store the planned JCL batch jobs that administrated all transfer jobs into a file with an agreed name and to inform them thereafter. The responsible system administrator copied this file and put it into the over night batch queue.

CONCLUSION

What have we learned from this migration project? It had to be thoroughly planned. The deep knowledge of the technical background was necessary. Many functions, BDM users, system administrators and the SAS site representative, were involved either in planning and knowledge transfer or in conducting.

This migration project proofed once more that the SAS system is a very powerful tool. It can also be used in areas not explicitly described in the manuals. But the user should always be aware of what he is doing even if he uses the defaults.

ACKNOWLEDGMENTS

Acknowledgments go after your references. This section is not required.
The author would like to thank Dr. Michael Knößl, Ralf Minkenberg, and Paul Kearney for critical review and valuable advices.

CONTACT INFORMATION

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

Fronke Gerken
Boehringer Ingelheim
Binger Straße 173
55216 Ingelheim
Work Phone: +49-6132-77-8505
Fax: +49-6132-72-8505
Email: fronke.gerken@boehringer-ingelheim.com