Paper 288-28

PROC MIGRATE: How to Migrate Your Data and Know You've Done It Right!

Diane Olson, SAS Institute, Cary, NC
David Wiehle, SAS Institute, Cary, NC

ABSTRACT

Migrating your data to a new version of SAS can be relatively simple or complex depending on the attributes of your data libraries. Some issues to consider when adopting SAS 9 include the version of SAS in which your data currently reside, what member types exist in your libraries, and whether you must move members from 32-bit libraries to 64-bit libraries. To address these issues, SAS 9.1 includes a new utility procedure, PROC MIGRATE. PROC MIGRATE streamlines the process of moving libraries forward to a new release. This paper introduces PROC MIGRATE and compares it to migrating with the traditional methods – PROC COPY, PROC CPORT/PROC CIMPORT, and PROC CATALOG. SAS also provides tools that can help you validate the content and attributes of your data after conversion. We discuss ways to use PROC DATASETS, PROC CONTENTS, and PROC COMPARE to ensure the integrity of your data.

GET THE LATEST INFO ON PROC MIGRATE

Development work on PROC MIGRATE was ongoing as this paper was being written. See the Version 9 Resources page at http://www.sas.com/rnd/base/index-v9-resources.html to access the latest update to this paper and the associated validation macros.

WHY A NEW PROCEDURE?

When a new version of SAS is installed, customers may need to upgrade their files from an earlier version of SAS to the current version. This has traditionally been called conversion and accomplished with the COPY, CPORT/CIMPORT and CATALOG procedures (hereafter referred to as the *copy* procedures), or some combination of them. Each of these procedures has many options that may be used when moving library members; when using these procedures, you must consider whether or not to keep indexes, integrity constraints and attributes such as compression. Should you preserve the original creation date? Should you convert the entire catalog or just convert the "wrapper" and allow each entry to be converted as it is accessed? Should you copy the entire library or individual files and sacrifice your referential integrity constraints?

These options are necessary for a multitude of reasons, but none of the *copy* procedures were designed specifically to migrate a SAS library to a new release. New for SAS 9.1, PROC MIGRATE provides a simple interface for migrating a SAS library from an earlier version to the current version. It allows new functionality needed for migration that the other *copy* procedures do not.

The new procedure makes it simpler for the customer to migrate a library and removes potential performance impact and unnecessary code complexity in the *copy* procedures.

PROC MIGRATE

INTENDED USAGE

Unlike other procedures, PROC MIGRATE is designed to process a library only once for each release. After a library is converted to the current release, PROC MIGRATE will most likely not be needed again for that library. The only intended usage of PROC MIGRATE is to move a Version 6.12 or later library forward to the current release; it is not designed to have the flexibility of one of the *copy* procedures. For example, you are not allowed to migrate individual files from a library. This restriction is necessary for several reasons, the most important being the retention of referential integrity constraints and maintaining data set generation groups.

PROC MIGRATE is not designed to move libraries from one machine architecture to another. Migrating libraries across machine architectures is strongly discouraged, as results are not guaranteed. Use PROC MIGRATE for that task at your own risk. However, if you need to migrate members that were created using a 32-bit platform in order to use them on a 64-bit homogeneous platform, you can use PROC MIGRATE. For more information, see *Migrating 32-bit libraries to 64-bit libraries* later in this paper.

PROC MIGRATE SYNTAX

PROC MIGRATE IN=*source-libref* OUT=*target-libref* <BUFSIZE=*bufsize*> <MOVE> <SLIBREF=*remote-libref-for-32-bit-migration*>;

Required arguments are the IN= and OUT= librefs. *source-libref* references a Version 6.12 or later library. *target-libref* references a library assigned to the current release. Not only must you migrate forward to the current release, but (as with the *copy* procedures) you must also migrate to a different physical location. There is no provision for migration in-place. Therefore, the source and target librefs must reference different physical locations. Also, the target library must be assigned to the current version's BASE engine. This can be accomplished by:

```
libname outlib BASE 'valid-path';
```

or

```
libname outlib 'valid path';
```

if the *valid-path* currently contains no SAS files.

You can use the BUFSIZE option to specify the bufsize for members of the target library. The default value used is the bufsize of the source library SAS member, potentially different for each member.

The MOVE option is for customers who are short on storage space and want the source library's members to be deleted once the migration of those files to the target library is completed without errors. This is for space-constrained customers only, as it is preferable to verify the migration of the member before it is deleted (see the *Validation of Your Library's Migration* section of this paper). The validation process is more effective if the source library member exists until its migration can be validated.

The SLIBREF option is only necessary for customers who wish to migrate 32-bit catalogs to a 64-bit SAS version. These 32-bit catalogs are inaccessible from a 64-bit SAS session unless this migration occurs. For more information, see *Migrating 32-bit libraries to 64-bit libraries*.

ADVANTAGES OF PROC MIGRATE

The goal of PROC MIGRATE is to produce members in the target library that are exactly the same as the source library, except that the target library's members are in the current SAS version's format.

With PROC MIGRATE,

- Migrated SAS members retain the created date/time and modified date/time of the source library's members, as PROC COPY does when the DATECOPY option is specified. For more information on the DATECOPY option, see the DATASETS Procedure in *Base SAS Procedures Guide*.
- All deleted observations are retained in migrated data sets.
- Audit trails are migrated.
- Indexes and all integrity constraints are automatically retained in a migration. When migrating an indexed data set, SAS migrates the data set first, and then the index is applied. If errors occur while indexing a migrated data set, the data set will be migrated without the index; this will be noted in the SAS log. However, if errors occur when applying either integrity constraints or initiating the audit trail on a migrated data set, the data set will be deleted from the target library, and an error will be written to the SAS log. The reason for this difference in error behavior is best explained in terms of data integrity. A data set that is migrated without its index may lead to a performance issue, but its data integrity remains intact. However, a data set migrated without its integrity constraint or audit file creates a data integrity exposure.
- Generation data set groups are migrated.
- Compressed files remain compressed, and encrypted files remain encrypted.
- Passwords are retained, though you do not need to specify them at migration time.

Another advantage to PROC MIGRATE is its simplicity. When using PROC MIGRATE, you do not need to consider all the possible permutations of options available with the *copy* procedures. Data attributes and auxiliary files, such as indexes and audit trails, are automatically migrated to the target library. The target library becomes a clone of the source library, in the format of the current version.

Since PROC MIGRATE provides functionality necessary for migration that the other *copy* procedures do not, the results from PROC MIGRATE are different from the *copy* procedures. For example, PROC MIGRATE retains all deleted observations in migrated data sets. The *copy* procedures clean up the data sets, such that deleted observations are removed and disk space recovered. This restructuring has its advantages, but results in a data set that is not historically accurate if you are tracking changes through an audit trail. Because the *copy* procedures do not keep deleted observations, the transactions noted in the audit trail would not match the observations in the copied data set due to the removal of deleted observations. PROC MIGRATE, however, migrates deleted observations and audit trails.

SUGGESTED USAGE

It is highly recommended that the target library be empty before your migration begins. If a member of the same name currently exists in the target library, an error will be printed to the log and that member will not be migrated from the source library.

If errors are encountered when migrating a member of the source library, and the cause of the errors is eradicated, a second PROC MIGRATE invocation will result in migrating only those members not already in the target library. In other words, PROC MIGRATE does not overwrite existing members in the target library.

DETAILS ABOUT SPECIFIC MEMBER TYPES

DATA

The only differences between the *copy* procedures and PROC MIGRATE for data files are the retention of deleted observations, automatic retention of all integrity constraints and migration of the audit trail. See Table 1 below.

***Copy* Procedures**	**PROC MIGRATE**
No deleted observations retained	Deleted observations retained
No audit trails copied	Audit trails migrated
General integrity constraints copied only when option specified; referential integrity constraints as well if entire library copied	All integrity constraints migrated by default

Table 1

When an audit trail is migrated, you will see an additional entry in the audit trail indicating that the data set was migrated. The audit trail is migrated in its present state. For example, if the audit trail was suspended, the resultant audit trail is also suspended; after the "migrated" record is written, an additional "suspended" record is added to the audit trail data set. The "migrated" record is

always added; a “suspended” record will only appear if the source library’s audit trail is suspended prior to migration. See Example 1.

```
Obs reason_code     _ATMESSAGE_
 1  Added record
 2                  SUSPEND
 3                  MIGRATE
 4                  SUSPEND
```

Example 1 PROC PRINT of Suspended and Migrated Audit Trail (keep=reason_code _ATMESSAGE_)

VIEWS

There are three categories of views to be considered: DATA step views, SQL views and SAS/ACCESS engine views. Each uses an entirely different format and has different migration considerations.

SQL VIEWS

SQL views retain data in a transport format; therefore there are no problems to be concerned about when migrating an SQL view. The view will work the same in the target library as it did in the source library, even for the 32-bit source library and 64-bit target library.

DATA STEP VIEWS

In Version 8.0, DATA step views gained the ability to store the source used to create the view. Using that source, DATA step views are migrated to current versions of SAS. The first time the newly migrated DATA step view is accessed, the source is automatically recompiled. In this way, the execution of the DATA step view will take advantage of fixes and upgrades available in the current release. DATA step views previous to Version 8.0, or DATA step views not containing their source, cannot be migrated. They must be recreated from the source. An error message is written to the log in those cases.

SAS/ACCESS ENGINE VIEWS

PROC MIGRATE makes use of a new procedure, PROC CV2VIEW, to migrate SAS/ACCESS engine views written with the Oracle, Sybase or DB2 engines. PROC CV2VIEW messages may be seen in the log during the migration. Views from Version 6.12 and later will be migrated. For more information, see the CV2VIEW Procedure in *Base SAS Procedures Guide*.

CATALOGS

PROC MIGRATE makes use of PROC CPORT and PROC CIMPORT to migrate catalogs. (You may notice CPORT or CIMPORT notes being written to the log while PROC MIGRATE is running.) The result of using the CPORT/CIMPORT combination is that catalogs are fully converted to the current version.

PROC COPY does not provide this functionality. Instead, it changes the outer "container" catalog to the new version, but leaves the entries as they are, to be converted as they are updated.

You could use PROC CPORT and PROC CIMPORT to convert your catalogs, but PROC MIGRATE provides one-step conversion for your entire library.

MDDBS

PROC MIGRATE will migrate MDDBs to a SAS 9 library. The only exceptions are Version 7 MDDBs, which cannot be accessed with any version other than Version 7. If you attempt to migrate a Version 7 MDDB, you will get an error message in the log.

SAS 9.1 supports a new OLAP storage format. If you are interested in OLAP processing, you may want to investigate this new product. It has enhanced features and improved performance. See the SAS OLAP Server Administration Guide for more information.

PROGRAM FILES

Stored compiled DATA step program cannot be migrated. They must be recreated from the source.

ITEM STORES

There are no special concerns for the migration of item stores, unless you are migrating from a 32-bit library to a 64-bit library. In that situation, item stores cannot be migrated.

Note that item stores cannot be stored in a sequential library. This is important only if you save a copy of your library to tape before using the MOVE option in PROC MIGRATE.

MIGRATING 32-BIT LIBRARIES TO 64-BIT LIBRARIES

If you were running Version 8 or previous versions of SAS on AIX, Solaris or HP/UX, then you have 32-bit members in your libraries. Those platforms have been changed to 64-bit-only access starting in SAS 9. Note that Linux platforms still have 32-bit access in SAS 9, so there is no 64-bit conversion necessary for that platform.

For the 64-bit platforms, you can still access your 32-bit data sets and MDDBs. You must migrate other member types in order to access them. As previously noted, DATA step views created before Version 8.0 or DATA step views not containing their source will need to be recreated. All stored DATA step programs will need to be recompiled from source. Item stores must be recreated.

PROC MIGRATE will automatically convert the members of the IN= library to 64-bit versions in the OUT= library. If you have catalogs that you need to migrate, you must have access to a 32-bit version of SAS that can be used during PROC MIGRATE. If you have licensed SAS/SHARE© or SAS/CONNECT© software, you specify a remote libref assigned to access the source library through a 32-bit SAS server as the *libref* for the SLIBREF option. Your catalogs will be read through that connection and then written out in 64-bit format in the target library. If you do not have either SAS/SHARE© or SAS/CONNECT© software, you will need to migrate your catalogs by hand using CPORT on a 32-bit SAS invocation, and CIMPORT in the current version of SAS to place a migrated copy in the OUT= library.

For more information, see Processing 32-Bit Version 6, 7, or 8 SAS Files with 64-Bit SAS 9 at http://www.sas.com/rnd/base/topics/convert9/ .

VALIDATION OF YOUR LIBRARY'S MIGRATION

INTRODUCTION

Since PROC MIGRATE is a new procedure, we needed to develop methods and tools to validate its behavior. The software validation motto, "Know your requirements, and know your data" is certainly familiar to most readers. This concept played a central role in defining the methodology we developed to perform this task. Keeping this concept in mind may help explain why we chose certain data checking methods above others. The validation macros discussed later grew out of our validation choices and is the tool we used to validate our results. We realized that customers could also use it to ensure the correctness of their own data migration. This tool alone is not enough to provide that certainty, however. It must be used in conjunction with your knowledge of the contents of your data libraries.

PROC MIGRATE VALIDATION STRATEGY

The migration validation strategy can be conceptualized in two basic parts. First, you must determine the expected behavior of the migration of a particular library. Then, after the migration, you must prove that the migration produced those expected results. In other words, what members and member attributes were in the source library prior to the migration? What members and member attributes appear in the target library after the migration?

SOURCE LIBRARY MEMBERS

Unfortunately, knowing your data is not as simple as knowing what member types exist in the source library. Consider the hidden complexities of SAS data sets. Do your data sets have indexes? Integrity constraints? Password protection? How about permanent formats? What are the lengths of your variables? Are your data sets labeled? Do your data sets have data representation and encoding values different from the default data representation and encoding of the operating system? In order to validate the migration, you must document all these details before the migration to determine whether or not the migrated data sets contain expected attributes. PROC CONTENTS prints the contents of a SAS data set to the listing, including the data set attributes described above. Besides documenting this information, it is also necessary to understand what attributes are expected to change when a data set is migrated from the source to the target library. For example, the data sets in the source library were created with a different engine than that of the target library, so it's expected that the *Engine* in the listing of a PROC CONTENTS of the migrated data set will differ. Certain differences from the source library are the desired result. Another example of this is the *Encoding* value in the PROC CONTENTS output; this is a new field added in SAS 9, such that pre-SAS 9 data sets will display "Default" for that value. In the migrated member, you will see a different, more specific value.

PROVE PROC MIGRATE PRODUCED EXPECTED RESULTS

Customers who are familiar with PROC COMPARE will recognize the value of having PROC COMPARE perform the "heavy lifting" when comparing two data sets. However, early in our planning process, we determined that the output generated for validation must be reasonable. Whatever method we chose had to strike a balance between providing too little output to adequately validate the results, and providing too much output to review in a reasonable amount of time. With the use of the proper options, PROC COMPARE produces very little output when there are no differences between two data sets, but it can produce a huge amount of output if there are any differences. We decided that PROC COMPARE is best suited for situations in which data sets are expected to compare exactly; unexpected differences will be immediately recognizable. Situations in which data sets were not expected to compare exactly required a new comparison technique.

USING ODS TO VALIDATE PROC MIGRATE RESULTS

Consider this example of default PROC CONTENTS output which displays the attributes of two data sets in the source and target libraries:

```
Data set in source library
Data Set Name        SOURCE.IC_TWO
Member Type          DATA
Engine               V8
Created              May 1, 1999
Last Modified        May 1, 1999
Protection
Data Set Type
Label
Data Representation  WINDOWS
Encoding             Default


Data set in target library
Data Set Name        DESTIN.IC_TWO
Member Type          DATA
Engine               BASE
Created              May 1, 1999
Last Modified        December 1, 2002
Protection
Data Set Type
Label
Data Representation  WINDOWS
Encoding             wlatin 1
```

A review of these two listings will quickly reveal differences in *Data Set Name, Last Modified* and *Encoding*. (Note that the *Last Modified* time changes because of an integrity constraint being recreated after the data set was migrated.) Besides this attribute data, PROC CONTENTS also reports variable information and engine/host data.

If you used PROC CONTENTS output to validate the results of PROC MIGRATE, the output generated from each data set in the source library would have to be manually compared to those in the target library. This would be a time-consuming but manageable task if you were comparing ten data sets in a library. But what if your library contained a hundred data sets? Or a thousand?

Fortunately, SAS offers an alternative to this tedious chore. The Output Delivery System (ODS) allows

customers flexibility in choosing what output is produced by a SAS procedure and flexibility in choosing what that output contains. For example, by combining ODS output statements and a simple DATA step, you can direct PROC CONTENTS output to a data set (similar to the OUT= option) and produce a listing of only those data set attributes that are different in the source and target libraries.

```
attribute        source library
Data Set Name    SOURCE.IC_TWO
Encoding         Default
Engine           V8
Last Modified    May 1, 1999

attribute        target library
Data Set Name    DESTIN_IC_TWO
Encoding         wlatin1
Engine           BASE
Last Modified    December 1, 2002
```

This output is considerably easier to review than the default PROC CONTENTS output, but it can be refined even further. *Data Set Name* (because the libname is included in the data set name) and *Engine* will generate insignificant differences. These data set attributes can be excluded from the comparison. Once the PROC CONTENTS output is directed to a data set, you can use data set options to keep only those variables (i.e., data set attributes) that are deemed significant for the comparison.

```
attribute        source library
Encoding         Default
Last Modified    May 1, 1999

attribute        target library
Encoding         wlatin1
Last Modified    December 1, 2002
```

A SAS program that produces output similar to this can be seen in Appendix 1.

WHY OUR VALIDATION SOLUTION USES THE MODULAR APPROACH

Our PROC MIGRATE validation solution is composed of two parts: a migration validation program template and a series of macros that perform specific validation tasks. The validation program template is a sample program that contains a generic PROC MIGRATE step. It also contains instructions indicating which validation macros should be run, and whether they need to be run before or after PROC MIGRATE. We decided to use a number of smaller macros separated by function instead of a single large macro primarily because we wanted to allow maximum flexibility in the PROC MIGRATE step.

For example, customers who are migrating from a 32-bit platform to a 64-bit platform may add an SLIBREF option to their MIGRATE step that is not required for other customers. If we integrated the PROC MIGRATE step into the same program that performed the validation, we would increase the amount of input required by the validation macros. By performing the PROC MIGRATE step separately from the validation macros, we can make the validation macros straightforward.

We also determined that since PROC MIGRATE was new, a certain amount of trial and error could occur as a normal part of the migration process. We wanted to design a solution that would prevent the need to restart the validation process from scratch should there be a need to rerun a portion of the migration. For example, if PROC MIGRATE encounters a member in the target library with the same name and memtype as a member in the source library, then SAS outputs an ERROR in the log and does not migrate the member. If the member in the target library is renamed or deleted, the PROC MIGRATE step could be repeated, and the member would be successfully migrated. The modular solution allows customers in this situation to simply pick up the validation process where they left off.

INTRODUCTION TO THE VALIDATION MACROS

We determined that customers are likely to have a large number of members in a source library, as well as having different member types. It seemed logical to automate the validation process using SAS Macro as much as possible.

Once you compile the macros either by copying the macro code into an interactive session and submitting the code, or by saving a copy of the validation macro program and using “%include”, all that is required is to define three librefs: a source library, a target library, and a library to contain ODS output data sets.

```
%include 'location of validation macro
    program';
libname lib1 [engine] 'valid path
   referencing location of source files';
libname lib2 BASE 'some other valid path
   referencing migration target library';
libname ods 'still another valid path
   referencing destination of ods output
   data sets';
```

Note that the first libname statement includes an optional engine assignment. It is required only if you have a mixed library, i.e. one containing members created using different versions of SAS, such as Version 6 and Version 8. In that case, the engine assignment in the libname statement specifies which version’s members you wish to migrate.

DOCUMENTING THE CONTENTS OF THE SOURCE LIBRARY BEFORE THE PROC MIGRATE

Once the librefs are assigned, the next step is to capture information about the source library before the migration. What members are in the source library, and what are their attributes?

%mig_in_lib is a macro which creates a data set in the ODS library containing the name and memtype of each member in a library.

```
/*create a data set which contains the
 members in the source library before the
 PROC MIGRATE*/
%mig_in_lib(lib=lib1);
```

As discussed earlier, files of different memtypes require different comparison methods. The data stored in data sets can be compared directly using PROC COMPARE, while the data stored in catalogs, item stores, audit files, etc. must be compared indirectly by some other means. Since customers may need to migrate large numbers of the same memtype, we created macros that can be used to compare results for large numbers of them. These memtype comparison macros will be discussed later in this paper.

In an early version of our tool, we simply strung together a series of these memtype comparison macros. If at least one file with a certain memtype was present, the memtype comparison macro would produce output. If no files of a certain memtype were present, the SAS macro produced a "file not found" ERROR. It soon became clear that we were outputting a great deal of unnecessary ERROR messages to the log. We decided before we attempted to compare the contents of files of a particular memtype, we first needed a way to determine whether or not files of a particular memtype existed in the source library.

%mig_source uses the "source library" data set created above to create "memtype flag" variables, which indicate the presence of files of a particular memtype in the source library.

```
data _null_;
set ods.lib1m;
     select (memtype);
        when ("CATALOG") do;
          %let catalog_here=Y;
        end;
        when ("DATA") do;
          %let data_here=Y;
        end;
     otherwise;
     end;
```

Once these memtype flag variables are set, it's a simple matter to surround the memtype comparison macro calls with conditional code:

```
%if &data_here=Y %then %do;
    %checkdata;%end;
```

%mig_source also creates macro variables which drive the individual memtype comparison macros. We wanted our validation solution to require as little input as possible, and decided to let SAS "catalogue" the source library for us. %mig_source also uses the "source library" data set to create macro variables containing the total number of files of each memtype in the source library, as well as a macro variable containing the name of each file in the source library. For example, consider a source library that contains the following files:

Name	Memtype
BTIMES	DATA
CARDS	DATA
FORMATS	CATALOG
STORE	ITEMSTOR

There are files of three different memtypes in the source library, so %mig_source will create three "total" variables, &lib1tdata (with a value of two), &lib1tcatalog and &lib1titemstor (each with a value of one). %mig_source stores the names of each file in a macro variable according to the table below:

Name	Memtype	Macro variable
BTIMES	DATA	&lib1data1
CARDS	DATA	&lib1data2
FORMATS	CATALOG	&lib1catalog1
STORE	ITEMSTOR	&lib1itemstor1

Now that we've documented the contents of the source library, we can submit a PROC MIGRATE step. After that, we can then begin validating the PROC MIGRATE.

VALIDATING THE RESULTS OF THE PROC MIGRATE, PART 1: WHAT FILES WERE MIGRATED?

After the files are migrated, %mig_in_lib can be run again on the source library with the "after=y" option to create a second "contents of the source library" data set in the ODS library. This is useful for customers who wish to confirm that the original files remain in the source library after the migration. It is also useful for customers who choose to use the MOVE option, as you will see in a later section of this paper.

```
PROC MIGRATE in=lib1 out=lib2;run;

/*create a data set which contains the
  members in the source library after
  the PROC MIGRATE.*/

%mig_in_lib(lib=lib1, after=y);
```

Our solution includes one final %mig_in_lib macro call, this time run on the target library. This produces a "contents of the target library" data set in the ODS library.

```
/* target library after MIGRATE.*/

%mig_in_lib(lib=lib2);
```

The next step is comparing the source and target libraries. %mig_check_libs prints a side-by-side comparison of the contents of the source library before the MIGRATE with the contents of the target library after the MIGRATE:

```
contents of target library after MIGRATE
(relative to source lib)

name       MemType    result
FORMATS    CATALOG    OK
TESTCAT    CATALOG    OK
C_INDEX    DATA       not MIGRATED
DATA_SET   DATA       OK
ITEMSTOR   ITEMSTOR   OK
DS_VIEW    VIEW       OK
SQL_VIEW   VIEW       not in source library
```

In this example, five members were present in the source library and were migrated successfully to the target library (result="OK"). The last file on the list was "not in source library", presumably because it was already in the target library before the PROC MIGRATE. One file ("C_INDEX") was not migrated, perhaps because there was some problem during the migration. In this situation, you should check the SAS log for errors to determine the reason that this file was not migrated.

%mig_check_source produces a comparison of the contents of the source library before the MIGRATE with the contents of the source library after the MIGRATE. The contents of the source library (created when %mig_in_lib was run on the source library) are automatically checked against the contents of the target library (created when %mig_in_lib was run on the target library). By default, %mig_check_source produces output based on the default behavior of MIGRATE (i.e., original source files remain in source library after migration)

```
***note the move option is absent;
PROC MIGRATE in=source out=dest;run;

%macro mig_check_source;

source library after PROC MIGRATE
(OK indicates member was present in
 source lib before and after MIGRATE)

name       MemType      result
FORMATS    CATALOG        OK
TESTCAT    CATALOG        OK
C_INDEX    DATA           OK
DATA_SET   DATA           OK
ITEMSTOR   ITEMSTOR       OK
DS_VIEW    VIEW           OK
SQL_VIEW   VIEW           OK
```

In this example, all the files that were present in the source library before the PROC MIGRATE are present afterwards, as expected.

For those customers who wish to use the MOVE option with PROC MIGRATE, %mig_check_source can produce validation output if you add "move=yes" to the macro call.

```
***note the move option is present;
PROC MIGRATE in=source out=dest move;run;

%macro mig_check_source(move=yes);

source library after MIGRATE move
(OK indicates member was present in
 source lib and deleted after MIGRATE)

name       MemType      result
FORMATS    CATALOG        OK: migrated
TESTCAT    CATALOG        PROBLEM
C_INDEX    DATA           OK: migrated
DATA_SET   DATA           OK: migrated
ITEMSTOR   ITEMSTOR       OK: migrated
DS_VIEW    VIEW           OK: migrated
SQL_VIEW   VIEW           OK: migrated
```

In this example, six of the files that were present in the source library were deleted after the PROC MIGRATE, as expected. However one file, indicated in the example by result="PROBLEM", remained in the source library after the MIGRATE. You should check the SAS log for errors to determine the reason this file was not deleted. Please note that in order to make the validation macros as flexible as possible, they only produce validation output if the source library is actually migrated. Customers who choose to use the MOVE option when migrating source libraries should do so with great care. It is not possible to use any of the memtype comparison macros discussed below unless the source files remain in the source library after the migration. In other words, using the MOVE option significantly limits the validation tools at your disposal.

VALIDATING THE RESULTS OF THE PROC MIGRATE, PART 2: ARE THE FILES CORRECT?

%checkdata is one of the memtype comparison macros mentioned earlier. It uses the macro variables created by %mig_source to output the following to the listing:

1. a side-by-side comparison of data set attributes between the source and target libraries
2. a side-by-side comparison of data set engine/host information
3. a PROC COMPARE of data set contents.

It should be noted that so long as the %mig_source macro is run prior to running %checkdata, customers do not have to input data set names, or even know how many data sets are expected to be in the source or target libraries to produce validation output from %checkdata.

%CHECKDATA STEP 1: COMPARISON OF DATA SET ATTRIBUTES

The first title line of each %checkdata output displays the name of the data set compared in each library. The current data set number (of the total number of data sets in the source library) appears in the second title line. The third title line indicates only differences in header data are displayed in the report, and the fourth title line indicates that all other header information not displayed is identical. Please note the differences in *Encoding* and *Engine* are expected.

%checkdata sample output 1: differences in header data

```
C_INDEX
Number 2 of 12 data sets in source library
Differences in PROC CONTENTS header data
Note:  all other header data was the same
Number 1 of 3 reports for this data set

attribute    source     target
Encoding     Default    wlatin1
Engine       V8         BASE
```

%CHECKDATA STEP 2: COMPARISON OF ENGINE/HOST INFORMATION

As in the example above, the differences in engine/host information are limited to expected differences, *Host Created* and *Release Created*. The *Host Created* differs between the source and target libraries because the MS Windows operating system with which the source library was created is different than the current operating system. Please note that *File Name* was excluded from this comparison.

%checkdata sample output 2: differences in engine/host data

```
C_INDEX
Number 2 of 12 data sets in source library
Differences in PROC CONTENTS engine/host
Note: other engine/host data was the same
Number 2 of 3 reports for this data set

attribute        source    target
Host Created     WIN_PRO   XP_PRO
Release Created  8.0202M0  9.0100A0
```

%CHECKDATA STEP 3: PROC COMPARE OF DATA SET CONTENTS

The PROC COMPARE used in %checkdata makes use of two COMPARE options; BRIEFSUMMARY prints only a short comparison summary to the listing, and LISTALL lists all variables and observations found in only one data set, if differences exist.

%checkdata sample output 3: PROC COMPARE of data set contents

```
C_INDEX
Number 2 of 12 data sets in source library
PROC COMPARE of data set contents
Lib1 is source library, lib2 is target
Number 3 of 3 reports for this data set

The COMPARE Procedure
Comparison of LIB1.C_INDEX with
      LIB2.C_INDEX
(Method=EXACT)
NOTE: No unequal values were found
```

The default behavior of %checkdata is to output only those data set attributes and engine/host data that are different in the source library and the target library. To output a comparison of all data set attributes and engine/host data, submit the following:

```
%checkdata(showall=yes);
```

This produces output similar to the following:

%checkdata(showall=yes) sample output 1: comparison of data set attributes

```
C_INDEX
Number 2 of 12 data sets in source library
PROC CONTENTS header data
Number 1 of 3 reports for this data set

attribute             source     target
Compressed            NO         NO
Data Representation   WINDOWS    WINDOWS
Data Set Type
Deleted Observations  0          0
Encoding              Default    wlatin1
Engine                V8         BASE
Indexes               1          1
```

%checkdata(showall=yes) sample output 2: comparison of engine/host information

```
C_INDEX
Number 2 of 12 data sets in source library
PROC CONTENTS engine/host data
Number 2 of 3 reports for this data set

attribute             source     target
Data Set Page Size    4096       4096
First Data Page       1          1
Host Created          WIN_PRO    XP_PRO
Index File Page Size  4096       4096
Max Obs per Page      126        126
```

%CHECKDATA WHEN USED 32-BIT TO 64 BIT MIGRATION

Customers who are migrating source libraries created with 32-bit SAS to 64-bit target libraries face slightly more complex issues than those customers who are not. These customers must add an SLIBREF statement to their PROC MIGRATE step if their source library contains catalogs. The validation macros require no additional input in this situation, but the LIB1 libref must be assigned to your source library through the remote engine.

Expected differences produced by %checkdata when comparing an data set created on V8.2 32-bit SAS to the version migrated to 64-bit SAS 9.1:

```
C_INDEX
Number 2 of 12 data sets in source library
Differences in PROC CONTENTS engine/host
Note: other engine/host data was the same
Number 2 of 3 reports for this data set

attribute        source    target
Inode Number     967340    1041491
Release Created  8.0202M0  9.0100A0
```

%checkdata is one of a number of macros designed to

assist in the validation of the migration of individual members. Please refer to Table 2 below to determine which macro can be used to validate SAS files of a particular memtype. Full documentation of each of these macros can be found on the Version 9 Resources page at http://www.sas.com/rnd/base/index-v9-resources.html .

Memtype or file	Validation tool
Catalog	%checkcatalog
Data set	%checkdata
Data set with an index	%checkdata and %mig_indexes
Data set with integrity constraint	%checkdata and %mig_indexes
Data set with an audit file	%checkdata and %checkaudit
Generations data set	%checkdata
Itemstore	NONE
Data step view	%checkview
SQL view	%checkview
SAS/ACCESS view descriptor	NONE
MDDB	NONE

Table 2 Memtype Validation Tools

CONCLUSION

PROC MIGRATE is a new tool in the SAS utility procedure toolbox, enabling you to easily upgrade your SAS libraries to the latest version of SAS. Using the procedure in conjunction with the validation macros discussed will allow you to have confidence in your migrated library's data integrity.

CONTACT INFORMATION

If you have questions, please feel free to contact the authors of this paper. For PROC MIGRATE questions, send email to Diane.Olson@sas.com. For validation macro questions, send email to David.Wiehle@sas.com.

APPENDIX 1 EXAMPLE OF ODS "SMART COMPARE" PROGRAM

```
***libname source is source library;
***libname dest is target library;

   ods output attributes=atr1(keep=label1
                              cvalue1
                where=(label1 not in (" ",
               "Data Set Name")));

   ods listing close;
   PROC CONTENTS data=source.ic_two;run;
   ods listing;

   data atr1(rename=(label1=attribute
                     cvalue1=source));
   set atr1;
   run;

   proc sort data=atr1;by attribute;run;

   ods output attributes=atr2(keep=label1
                              cvalue1
               where=(label1 not in (" ",
              "Data Set Name")));

ods listing close;
PROC CONTENTS data=dest.ic_two;run;
ods listing;

data atr2(rename=(label1=attribute
          cvalue1=target));
set atr2;
run;

proc sort data=atr2;by attribute;run;

data atr;
merge atr1(in=in1) atr2(in=in2);
by attribute;
if in1 or in2;
if source ne target then output;
run;

proc print data=atr;run;
```