Version 6 Data Library Engines in the CMS Environment
Ronald O. Page, SAS Institute Inc., Cary, NC

ABSTRACT
This paper discusses the concept of the data library engine as it relates to applications in Version 6 of the SAS® System. The structure of the Multiple Engine Architecture (MEA) is described. The data library engine is examined, with emphasis placed on the types of engines and examples of their use. Special considerations are made in relating these topics to the CMS environment. Also discussed is the introduction of the LIBNAME statement as it relates to the association of an engine to a physical data library.

INTRODUCTION
Version 6 of the SAS System has introduced the Multiple Engine Architecture (MEA). This redesign of the SAS I/O layers has created the concept of the data library engine. Being the lowermost layer of the architectural structure, the engine is able to shield the upper layers, which consist of the engine supervisor and application from the peculiarities of a particular library format. When a single engine is employed, it provides a method of extracting data from a physical library for analysis and presentation by the SAS System. When multiple engines are used, the SAS System also becomes a connectivity tool, linking two or more physical libraries whose formats are engine-specific. The power of the SAS System may be used in the exchange of data between these external sources without the overhead of data conversion.

The following is a list of terms used in this paper:
host
any operating system on which Version 6 of the SAS System is implemented. CMS is considered a host.
SAS file
a member of a SAS library containing a filename and a filetype. There are many different varieties of SAS files. Some of them include data, catalog, access, view, and index.
SAS data set
a collection of data values, created by a native SAS engine and usually containing information such as observations and variable values. A SAS data set can be seen as a table in which the observations are the rows and the variables are the columns.
data set format
the format of a SAS data set as it resides on a particular host.
data set index
an auxiliary data set created in addition to the genuine data set to provide fast access to records or observations within a data set by a variable or key.
SAS data library
a collection of SAS files that share a common library name. This name is dictated by the user within a SAS application. The default library name is WORK.
data library engine
a set of routines that provide the services necessary to access a physical file as if it were a SAS data set. There are two types of engines, native and interface.
native engines
data library engines that create and access data in a format that is maintained by SAS Institute. Examples of the native engine are the base engine and the compatibility engine. The base engine is the default engine that is shipped with a particular release of the SAS System. The compatibility engine is responsible for accessing an earlier form of SAS data sets.
interface engines
data library engines that access external files as if they were SAS data sets. These external files are typically created by other vendors’ products.
engine supervisor
The upper layer of the I/O system that accepts an application’s requests for I/O services and dispatches these requests to the appropriate engine. It is the job of the engine supervisor to handle SAS conventions such as BY processing, that are not associated with the physical file format. This allows the engine to be as fundamental as possible.
VIO access method
a performance enhancement in Version 5 of the SAS System under CMS. By default, WORK library members are placed in memory rather than on disk, alleviating the overhead of doing I/O to a physical device.

MULTIPLE ENGINE ARCHITECTURE (MEA)
Overview
The Multiple Engine Architecture (MEA) is a multi-layered approach, each layer accepting its share of responsibility to perform the data set I/O. These layers are generically designed to facilitate the addition of new data library engines after the SAS System has been installed. For instance, user-written access methods can be developed and implemented to operate on other vendors’ files with no modification to the SAS System. A top-down examination of these layers will better explain the Multiple Engine Architecture (MEA) approach.

Figure 1 Layers of the MEA
The Four Layer Design

At the highest level, the user invokes the SAS System creating requests for reading and writing data. For example, invoking the DATA step, the PROC step, or any one of the many SAS applications constructs a request for data to be read or written to a particular data library member. One of the benefits of the MEA is that the user sees only the application layer and is unaware of the layers that reside beneath the application. An added benefit is that the syntax of the SAS application is effectively the same whether the user is invoking the base engine, the compatibility engine, or an interface engine to perform the needed i/o.

The next layer of the MEA is the application. SAS applications might range from an invocation of the DATA or PROC steps to highly interactive SAS applications such as SAS/ASSIST™ procedures. Regardless of the size, all applications require i/o services such as opening a library member, reading its contents, and so on. These requests for i/o services ultimately are relayed to the engine supervisor.

The engine supervisor is responsible for dispatching the application’s i/o requests to the appropriate engine. Aside from this duty, the engine supervisor also supports SAS conventions such as BY processing and the OBS, FIRSTOBS, and NOREPLACE data set options. Whenever possible, the engine supervisor relieves the individual engines from these duties, thus avoiding the needless repetition of code throughout all engines. By expanding the scope of the engine supervisor, the services that are provided by a particular engine can remain as primitive as possible. From a standpoint of flexibility, the engine interface provided by the engine supervisor is generally designed so that new engines can be developed and implemented to accommodate present and future library formats.

At the lowest end of the chain is the data library engine. As mentioned before, the services provided by an engine are somewhat primitive. The engine is responsible for reading from or writing to a given library member and is the only layer of the MEA that truly understands and interprets the format of the external file. For example, the Version 5 SAS compatibility engine has the ability to use Version 5 SAS data sets without the imposition of data set conversion. The compatibility engine is invoked automatically when accessing a Version 5 format library member, and reads the member directly, interpreting its contents for use by SAS applications.

The advantages of MEA design are obvious. Fundamentally, the SAS System will allow the user to continue to use Version 5 SAS data sets without the burden of file conversion. Naturally, users will ultimately want to do data set conversion to take advantage of new Version 6 SAS data set features like indexing and record compression. However, this task can be done at the user’s convenience. Secondly, the design is very flexible. Development and implementation of new engines can be done in the field with absolutely no change to the internals of the SAS System. This characteristic makes the SAS System a very competitive force in the area of database interfacing. The MEA design provides a sound foundation for the implementation of new interface engines as they are needed.

DATA LIBRARY ENGINE

Data Library Engine Overview

As mentioned previously, a data library engine provides only the most primitive services necessary to read and write to an external file. The engine supervisor insulates the engine from knowledge of SAS conventions such as BY processing and data set options in the DATA step. Likewise, the data library engine is the only layer of the i/o structure that knows and interprets the format of the external file. Thus, the engine insulates the engine supervisor from the burden of dealing with a particular file format.

Components of the Data Library Engine

There are a few basic services that a data library engine must provide for the engine supervisor to function. These functions are described as follows:

library services

- provide the means of logically assigning and deassigning logical libraries to a physical library. These services also include checking for library existence, resetting a library to be empty, parsing engine-dependent options, and returning engine-dependent information about a library.

directory services

- are necessary for tasks such as returning a list of members within a library for processing. Also included in directory services are routines that rename or delete members of a data library.

data level services

- are responsible for opening, closing, reading, and writing data library members. Data set options parsing also takes place at this level.

variable services

- are present to allow variable definition and selection depending on whether a library member is being created, updated, or read. Variable services also provide a method of obtaining a list of variable names and types within a library member.

catalog services

- are responsible for opening, closing, reading, and writing catalog files. Catalog services are optional in the development of engines. However, catalog services must be present if the catalog is to be shared among multiple users with SAS/SHARE® software.

index services

- provide a method of opening, closing, reading, and writing index files. Some of the index services include obtaining key definitions, defining and creating an index, removing and deleting an index, and getting statistical data for a particular index. As with the catalog services, index services are optional in an engine. For example, the Version 5 SAS base engine contains indexing while the Version 5 SAS compatibility engine does not.

Aside from the services mentioned earlier, a data library engine contains the services necessary to initialize itself, thus readying itself for processing. Likewise, an engine contains routines to give back unused resources when processing has terminated. Despite its primitive nature, a data library engine is an exclusive set of routines that are contained within an independent executable image. This allows the engine supervisor to load the engine into memory only when it is needed, and unload it whenever processing has completed.

The following diagram describes the services of the engine supervisor and also depicts the various interfaces of the MEA.
Engines that are created and maintained by SAS Institute Inc. to access some form of SAS data set. There are two basic types of engines. First, there are native engines that are designed to read and write other vendors’ files and are included in Version 6 of the SAS System. Following is a list of native engines that contain a number of new features such as indexing. The benefits from these new features are ease of use and improved performance.

The compatibility engine is responsible for processing old format SAS data sets. The compatibility engine is loaded automatically when a user accesses an old format SAS data set. The benefit of the compatibility engine is that it allows the user to continue to access Version 5 SAS data sets with no change in syntax. This will greatly ease the transition from one release of the SAS System to the next. While users are still encouraged to convert their data libraries to Version 6 SAS format, this conversion can take place at the user’s convenience.

The sequential engine is provided to create data libraries on sequential devices. The sequential engine lacks some of the overhead of the base engine since sequential access to information within a member is much simpler to implement than random access. As with past releases of the SAS System, the sequential file format is common among the IBM® hosts. Therefore, sequential libraries may be shared among the CMS, MVS, and DOS/VSE operating systems.

The transport engine is provided in order to produce machine-independent files that may be shared among any host running Version 5 or later of the SAS System. In Version 5 of the SAS System, PROC XCOPY was used by the IBM hosts (CMS, MVS, and DOS/VSE) to produce transport files. To exploit the benefits of the MEA, a transport engine was fashioned to accomplish the same task as PROC XCOPY. One might note that the TRANSPORT data set option was used by Version 5 of the SAS System under the VMS®, AOS/VS and ng PRIMOS® operating systems. The transport engine provides a common syntax for all hosts running Version 6 of the SAS System.

The remote engine is a communication link that interfaces with a SERVER to allow the exchange of data within SAS/SHARE software in Version 6 of the SAS System. The remote engine uses communications software to exchange data with a detached server. The remote engine will allow the reciprocity of data between users of the same system, or with users of other systems as well.

The view engine is responsible for creating and storing a view previously defined by an SQL™ operation. Views may be limited to a single SAS data set, or they may span heterogeneous data sources by some logical connection.

Interface Engines

Aside from native engines, Version 6 of the SAS System supports a number of interface engines. The number of interface engines available will surely grow as the need arises to interface with new and existing database products. Currently under CMS, there are plans to implement the following interface engines:

- SQL/DS™
- ORACLE®

As previously mentioned, interface engines are responsible for reading from and writing to external files that are in a format that is not a SAS format. These files may have been created by another vendor’s software, or the data may simply be in a format that is unknown to the SAS System. Whatever the case, an interface engine can be written to read and interpret the data directly without the overhead of conversion.

As a result of the interface engine, the SAS System becomes a connectivity tool linking two or more external databases. For example, the engine supervisor dispatches /O requests to two interface engines, each engine having the knowledge to read and write the file format of the external file that isn’t in SAS format.

A simple invocation of the COPY procedure would transfer all of the data in the input data file to the output data file. Since the engine is the only layer of the architecture involved with the actual format of the file, no time is lost in the conversion process.

USER INTERFACE TO ENGINES

Thus far, this paper has explained the MEA and described the data library engine. This section will further explain the syntax involved with the use of data library engines in Version 6 of the SAS System.
One of the goals of the multiple engine architecture is to provide a user interface that is as simple and user-friendly as possible. Whenever possible, little or no changes were made to the existing SAS syntax. Most of the changes that Version 6 of the SAS System introduced are additions and refinements to an existing syntax. This is quite a challenge considering the size and complexity of the SAS System. One area that is particularly indicative of this challenge deals with the transparent access of native data libraries.

Transparent Access Using Native Engines

Transparent access is a term that is used to describe the ability of the SAS System to recognize a particular native library format and to allow the user to access the data sets within the library as if they were in the Base library format. The user does not need to make any changes in order to access a Version 5 format data set using Version 6 of the SAS System. Conversion of Version 5 SAS format data sets to Version 6 SAS format can take place at the user's convenience. This will ease the transition from one release of the SAS System to the next.

To further explain the concept of transparent access, the following CMS example is offered. Suppose a user has several Version 5 SAS format libraries on his A disk. He invokes Version 6 of the SAS System and then invokes the DATASETS procedure on one of the Version 5 SAS format libraries. Remembering that library name translates into the filetype of the member, the SAS System interrogates all files on the A disk having the given filetype. As soon as the SAS System identifies the format of the native library, the searching is complete. The SAS System loads the appropriate engine into memory and the engine is called upon to process the requests of the DATASETS procedure. As with previous releases of the SAS System, mixed-format libraries are not allowed and an error will occur if a library member is encountered that is not of the chosen library format.

Transparent access ultimately affords the user the ability to access any native format data library without burdening him with the task of identifying the release of the SAS System that created the library.

User-Controlled Library Access

Aside from transparent access, the SAS System provides the user with the ability to control library access through the syntax of the LIBNAME statement. The LIBNAME statement is used to associate a library name, residing at a given location, with a particular engine. Version 6 of the SAS System adds an optional engine parameter to the LIBNAME statement. If the library is a native format, then the engine parameter may be omitted and the SAS System chooses the appropriate engine to use (transparent access). However, if the library is an interface library or if the user is sure about the format of the library, then the engine name can be specified on the LIBNAME statement to override the process of searching and comparing to identify the particular library format.

The format of the LIBNAME statement is as follows:

```
LIBNAME libref <engine> <physical name> <host & engine options>
```

where

- `libref` is an eight-character first level SAS data set name that identifies the library. This libref will be used in referencing the library throughout the SAS System.

- `engine` (optional) is an eight-character SAS name that identifies the engine to be loaded to process the library. There are several reserved engine names (LIST, QUERY, SHOW, CLEAR) honored in Version 6 of the SAS System for upward compatibility from Version 5 of the SAS System. The engine parameter is optional only for SAS native engines. All interface engines will require specification of the engine name on the LIBNAME statement. The default engine used for new native libraries is the base engine.

- `physical name` (optional) is a quoted string that describes the physical location of the library. This string, however, will vary from engine to engine. For example, under CMS the base engine accepts an optional filetype and a required filemode. The compatibility engine, however, accepts a required filemode. Under normal circumstances, the physical name parameter on the LIBNAME statement will be a disk mode letter.

- `host & engine options` (optional) is a list of host and engine options that are parsed and applied to this library. Options are parsed and passed down through the layers of the MEA. The engine supervisor handles common engine options and the individual engine is given the opportunity to handle the option at the engine or host level.

Examples of engine Usage

Examples of engine invocation using the Version 6 SAS LIBNAME statement are provided below. These examples are CMS specific.

1. Assign the base engine to point to the TESTONE library on the B disk. Then, use the CONTENTS procedure to provide information on all of the library members.

   ```
   LIBNAME TESTONE BASE 'B';
   proc contents data=TESTONE..ALL;
   run;
   ```

2. Assign the compatibility engine to point to the V5EXIST library on the A disk. Then, use the DATASETS procedure to list all of the library members.

   ```
   LIBNAME V5EXIST VS 'A';
   proc datasets library=V5EXIST;
   run;
   ```

3. Assign the transport engine to point to an external file that is to be sent to the PC. Then use the COPY procedure to place all of the members of the WCRK library into the external file.

   ```
   LIBNAME TRANSPORT SATIST "C:\"PC A";
   proc copy in=WCRK out=TRANSPORT;
   run;
   ```

4. Through an implicit assignment, use the PRINT procedure to print the contents of an existing Version 5 library member called V6IMPLY.FIRST.

   ```
   proc print in=V6IMPLY.FIRST;
   run;
   ```

Once a library has been explicitly assigned by a LIBNAME statement or implicitly assigned by using a libref without a corresponding LIBNAME assignment, the libref may be used throughout the SAS System and the specified engine will be used to perform the I/O.
Assignment of a libref remains intact until the libref is specifically cleared or reassigned.

5. Deassign the VSEXIST libref and make the libref available for reuse.

LIBNAME VSEXIST CLEAR;

or

LIBNAME VSEXIST;

**CMS SPECIFICS OF THE BASE ENGINE**

At this point, it will be beneficial to expound upon the CMS specifics of the MEA. A review of the underlying structure of the data library under CMS will help to establish a working knowledge of the SAS System in the CMS environment.

**Base Engine Data Set Format**

Although Version 6 SAS data sets under CMS continue to be RECFM=F, LRECL=1024, a more efficient means of determining internal blocking factors has been employed. The SAS System analyzes the projected size of a data set, given the size of an observation. This size is determined by a summation of the variables contained within the observation. A buffer size is then chosen that is within an acceptable range of waste and also a multiple of 1024.

At this point, the SAS System is able to exploit the powers of the CMS i/o services by reading and writing multiple fixed length records within the data set. This ability to read and write multiple records is dependent upon the internal buffer size that was chosen for the data set. For example, if a data set is created having an internal buffer size of 4096, then four records will be read or written each time that a physical write occurs. CMS very efficiently chains these writes and the i/o will be done with only one START I/O. This ability to limit the number of physical writes is very desirable in any i/o bound application.

In an effort to allow the user to specify the internal buffer size, the BUFSIZE option has also been introduced. This data set option gives the user the ability to override the internally generated buffer size if he needs to do so. As a result of these changes, the base engine has become more skillful in choosing a default internal buffer size. Likewise, the user can override the process of buffering size if he thinks it is necessary.

**Data Library Format**

Significant changes have been made in the area of determining the format of a SAS data library under CMS. As previously mentioned, transparent access is offered for native engines. In determining the format of a library, files containing the given filetype are searched until a native format is found. At that point, searching is terminated and all members of the library are assumed to be in the chosen format. As with previous releases of the SAS System, mixed format libraries are not allowed. The chosen engine is then loaded automatically by the SAS System and the user is free to access the data in the native library.

A user may choose to override transparent access by issuing a LIBNAME statement for a particular library and specifying the engine parameter. A moderate performance gain may be realized by specifying the engine parameter on a LIBNAME statement, but the SAS System will search and identify the particular native engine if it is not specified. In contrast, to access an interface engine, the user absolutely must specify the name of the engine on the LIBNAME statement. The SAS System can identify only native engine formats.

Because all SAS data sets under CMS files are simple CMS files, the SAS System requires that all members of a data library reside on the same minidisk. There is one exception to this rule. Read-only extensions of a single minidisk are considered part of the parent minidisk. In short, the SAS System will look for members of a particular data library on the specified minidisk as well as any extensions of that minidisk.

**The WORK Library and VIOBUF**

Version 5 of the SAS System under CMS introduced the VIOBUF option. VIOBUF, when specified, causes the WORK library to be placed in memory rather than on disk. The VIOBUF option is a performance enhancement because all i/o is done in memory, alleviating all of the overhead of a physical read or write to the disk. Version 6 of the SAS System under CMS continues to support the VIOBUF option. A few changes have been made in the VIO access method to ensure a clean, simple implementation.

The Version 6 SAS implementation of the VIO access method under CMS allocates a memory pool that is specified by the user with the VIOBUF option. All WORK library members are placed in VIO until the allocated storage is used up. At that time, all members of the WORK library are placed on disk and the memory pool is freed. This implementation differs from the Version 5 SAS implementation of VIO under CMS in that under the previous release, the largest member was placed on disk and the VIO access method continued to try to fit members into memory.

Although the previous implementation of VIO was very successful in increasing the efficiency of some applications, there were a few caveats in its usage. The choice of always placing the largest member on disk is somewhat haphazard because the member that is placed on disk might be the most readily accessed member. Second, the process of keeping some WORK library members in VIO storage and some members on disk made for a very complex implementation. Memory fragmentation was also a problem as members were placed in VIO and eventually rolled out to disk, leaving holes in the available storage.

The Version 6 SAS implementation of the VIO access method under CMS attempts to alleviate these problems. All WORK library members remain in VIO storage until the storage is depleted. At that time, all members are placed on disk. Users are encouraged to specify adequate VIO storage and to also encourage the management of VIO storage wisely by deleting unneeded WORK data sets within an application. Statistics can be printed at the end of each program step which will allow the user to monitor the amounts of VIO storage that are available.

As CMS users migrate to a VM/XA™ environment, the ability to define large working environments will become possible. Assuredly, the VIO access method will play an even more important role in enhancing the performance of i/o within the WORK library in a VM/XA environment.

**CONCLUSION**

This paper explored the concept of the data library engine as it relates to applications in Version 6 of the SAS System. A discussion of the MEA design and a description of its component parts was offered. In particular, the concept of the data library engine was discussed in detail and a list of engine types were presented. Also discussed was the syntax of the LIBNAME statement. Several examples of the LIBNAME statement augmented this presentation. The CMS specifics of the base engine were discussed, and explanations were offered about the enhancements and changes to the format of the base engine data set format. Lastly, a look at the VIO access method was taken, providing information about its proper and future uses.
Special thanks goes to Stephen Baatrous, a developer at SAS Institute for his contributions to this paper.

SAS and SAS/SHARE are registered trademarks and SAS/ASSIST is a trademark of SAS Institute Inc., Cary, NC, USA.

IBM is a registered trademark and SQL/DS, SQL, and VM/XA are trademarks of International Business Machines Corporation. ORACLE is a registered trademark of Oracle Corporation. VMS is a trademark of Digital Equipment Corporation. PRIMOS is a registered trademark of Prime Computer, Inc.