The SAS/IMS-DL/I software product is an interface that allows processing of DL/I data bases through use of SAS® programming statements under IMS/VS DB, IMS/VS DB/Dc, CICS/OS/VS, CICS/DOS/VS, and DL/I DOS/VS systems.

This paper demonstrates how to make the interface user friendly, using the SAS macro facility that is a part of the base SAS software. Introductions to IMS-DL/I terminology, SAS terminology, and the SAS macro facility are provided for clarity.

INTRODUCTION TO IMS-DL/I TERMINOLOGY

What Is DL/I?

DL/I stands for Data Language/I and is IBM Corporation's data base language for IMS/VS, DL/I DOS/VS, and CICS/VS systems. IMS/VS is an IBM data base management system that uses DL/I and consists of IMS/VS DB and IMS/VS DC. IMS/VS DB is the data base manager for managing DL/I data bases in batch mode. IMS/VS DC includes data communications facilities for on-line access and control of DL/I data bases. DL/I DOS/VS is the data base manager for DOS/VSE environments. CICS/VS is an IBM product that permits users to access data bases on-line. CICS/VS executes on both OS and DOS operating systems.

What Is a DL/I Data Base?

A DL/I data base is a large, centralized collection of information composed of one or more physical files that can be accessed by multiple users. A DL/I data base contains records. Each record contains data for one entity or case. The records are composed of related segments that are arranged in a hierarchical structure. Each record contains one root segment at the top level of the hierarchy with possible multiple dependent segments at various levels in the hierarchy below the root segment. A segment that is hierarchically dependent upon a segment one level up in the hierarchy is called a child segment, and the segment that it is dependent on is the parent segment. Segments that share a parent segment are called siblings. Multiple segment occurrences of one segment type with the same parent occurrence are called twins. Each segment may be accessed separately. Segments that are dependent upon one another can be grouped by paths. This allows multiple segments in a path to be accessed at the same time.

The DL/I data base record segments are composed of related fields. There are three different kinds of fields in the segments. The first is a sequence field (or key field). A root segment always contains a sequence field that uniquely identifies the record. A dependent segment may or may not contain a sequence field. In twin segments, the sequence field orders the twin segment occurrences in ascending order and may provide unique identification. A segment that is not defined to DL/I and cannot be used to search for particular values. The format of an undefined field is determined by the program that initially loads the data base.

Figure 1 Diagram of Sample Account Data Base

A Program Specification Block (PSB) is a DL/I control block that defines program views allowed a given program or application. Within the PSB, Program Communication Blocks (PCBs) are defined with a particular program view for a given data base. The PCB may contain all segments in a data base or a subset of segments from the data base. In the SAS/IMS-DL/I interface SAS example programs, the PSBs CUSREAD, ACCREAD, TRANREAD, and ACCUPDT are used.

DL/I data base segments are accessed when a program issues a DL/I call. Information needed for the DL/I call includes a call function, a PCB, an /O area, and segment search argument(s) (SSAs). The call function is the type of call requested, for example, GU for get unique. This call requests a particular segment to be retrieved. DL/I calls are classified as either get calls or update calls. A get call retrieves a segment or segments. An update call performs a write function, such as inserting a new segment or replacing or deleting an existing segment. The PCB contained in the PSB, as explained above, is needed to provide the program view allowed the executing program. An /O area is needed to transfer segment data between DL/I and the executing program.

A segment search argument or multiple segment search arguments can also be used for the DL/I call. A segment search argument (SSA) consists of formatted search criteria passed to DL/I to identify a particular segment or group of segments to be processed. If no SSA is provided for the DL/I call, the call is said to be an unqualified call that does not specify a particular segment. If an SSA or multiple SSAs are provided, the DL/I call is said to be a qualified call. An SSA may specify only the segment type to be accessed. In this case, the SSA is said to be an unqualified call.
SSA. Using unqualified calls or qualified calls with unqualified SSAs allows sequential processing of a data base. Qualified SSAs provide the segment type, a field name (the sequence field for the segment desired as defined in the DBD), an operator containing a comparison operator such as =, >, or <, and a value that is to be compared to the specified field in the segment. Random processing of a data base can be achieved by using qualified calls with an SSA. Multiple SSAs can be specified in hierarchical order to retrieve or update a particular segment at a lower level in a particular path. As many as fifteen SSAs can be specified in one call. The segment specified in the last SSA is the target segment. To access more than one segment in one call, an SSA can include a command code. The command code consists of an * followed by a command code. The *D command code allows a path call. A get path call will return the segments specified in each SSA in hierarchical order to the I/O area.

INTRODUCTION TO SAS TERMINOLOGY

The SAS DATA Step

SAS programs are organized into either DATA steps or procedure (PROC) steps. The SAS DATA step is used for reading and writing SAS data sets, reading and writing non-SAS files, report writing, file management, and information retrieval. The SAS DATA step is compiled, loaded, and executed under the control of the SAS supervisor.

A SAS data set is a file of SAS variables in fixed-format records referred to as observations. A SAS data set contains descriptor information for the data set, such as the type, length, position, and label of each variable in the data set. SAS data sets are created only by SAS programs from data input in the job stream, from data stored in non-SAS files, or from data stored in other SAS data sets. In a DATA step the SAS data set can be read with SET, MERGE, or UPDATE statements.

There are four file-handling statements used to read from and write to non-SAS files in the DATA step. These statements are INFILE, INPUT, FILE, and PUT.

The INFILE statement in the SAS DATA step describes the non-SAS input file. Multiple INFILE statements can be used in one DATA step. The input source is set or reset each time an INFILE statement is executed. Therefore, the current input file is the one named in the most recently executed INFILE statement. The DL/I INFILE statement in the DATA step has options to name the PSB, name a variable(s) for the PCB in the PSB, name a variable for the DL/I call function, name variable(s) containing SSAs(s), and name variables containing information after the DL/I call, such as the status code of the call or the retrieved segment name. The DL/I INFILE statement also includes a DL/I option that signifies this INFILE statement refers to a DL/I data base.

The INPUT statement in the SAS DATA step reads a record from the current input file into a buffer associated with the file. The INPUT statement can also format and move data from the input buffer into SAS variables when the read takes place. A DL/I INPUT statement issues a DL/I get call as formatted by variables specified in the DL/I INFILE statement. The DL/I INPUT statement also places the retrieved segment(s) in the input buffer. If variables are named in the statement, data from the input buffer is moved to SAS variables in the program data vector. After the DL/I INPUT statement executes, it is wise to check the status code, particularly in programs that make random access calls. The status code can be obtained by using the STATUS= option or PCBFL= option of the DL/I INFILE statement. The automatic SAS variable _ERROR_ is set to 0 if DL/I returns a blank status code or codes of CC, GA, or GK. In some programs the values of the _ERROR_, STATUS=, and/or SEGMENT= variables should be checked before moving data from the input buffer to SAS variables in the program data vector. This can be done by using a DL/I INPUT statement with a trailing @INPUT @. This allows the input buffer to be held until a second INPUT statement that specifies the variables is issued.

The FILE statement in the SAS DATA step describes the non-SAS output file. The most recently executed FILE statement determines the current output file. The DL/I FILE statement references a PSB that identifies a DL/I data base and includes a DL/I option to signify this is a DL/I FILE statement.

The PUT statement in the SAS DATA step writes to the current output file that is determined by the last executed FILE statement. If the CALL= option is specified in the DL/I INFILE statement and the CALL= variable is set to the appropriate update call before the DL/I PUT statement is executed, the DL/I PUT statement will write to the DL/I data base by issuing a DL/I update call. If CALL= is not specified, the call function defaults to GN (get-next) and no update calls can be issued. As in the case of the DL/I INPUT statement, it is wise to check the status code returned by DL/I immediately following a DL/I PUT statement.

The SAS Procedure (PROC) Step

The SAS procedure (PROC) step is used to process existing SAS data sets, produce a variety of statistics, format reports, and perform utility functions. PROC steps can also be used to create SAS data sets. Some PROC steps can process non-SAS files.

The SAS/IMS-DL/I interface includes a SAS procedure named PROC DLTEST. PROC DLTEST is an interactive, full-screen SAS procedure that allows the user to format and issue DL/I calls by filing in fields on the DLTEST screen and executing a RUN command. The results of the call are then displayed on the screen, including the PCB feedback data and the data returned to the I/O area. Retrieved segments can be browsed or edited on the DLTEST screen in character mode or hexadecimal mode. An output SAS data set with observations containing the parameters used in each call is created with each interactive execution and as a comparison of multiple executions. The command PSBDIS can be entered to display a screen with the information in the PSB that was specified for the execution. PROC DLTEST can also be executed in noninteractive mode. In noninteractive mode, the procedure builds DL/I calls from values stored in an input SAS data set and executes them. The input SAS data set can be created either by a separate execution of PROC DLTEST or by a DATA step.

SAS/IMS-DL/I Software Options

A number of SAS/IMS-DL/I software options can be specified in job control language (JCL), or in a SAS OPTIONS statement before or during a DATA or PROC step. These options are assigned default values by the installation. The options can also be overridden so that they cannot be overidden in a job execution for purposes of efficiency, security, and data integrity. Most of the SAS/IMS-DL/I software options parallel the DL/I parameters.

INTRODUCTION TO THE SAS MACRO FACILITY

What is the SAS Macro Facility?

The SAS macro facility can be used to construct and edit SAS system text for the DATA step. This processing is done before the text is recognized by the SAS System. The macro facility is controlled by a special macro language that uses the percent sign

152
(** and ampersand (&) to denote macro actions. The macro facility simplifies repetitive data entry and allows packaging of long, detailed programs that can be invoked with a simple command. The SAS macro facility can be used to

- include code that will accept parameters
- accept user input during macro execution
- retain macro variables across SAS steps
- conditionally create SAS statements.

It allows greater flexibility of SAS code.

A SAS macro is stored text containing SAS code and macro language statements that can be referred to by name. The simplest form of a SAS macro is

```
%MACRO macroname;
  macrocall
%MEND;
```

The `%MACRO` statement begins every macro and contains a name for the macro. The `%MEND` statement ends each macro and may or may not contain the name of the macro. The macro text may include any form of constant text. Constant text can be either a character string of SAS variable names, SAS data set names, or all or part of SAS statements. Macro variable references (sometimes called symbolic references) are variables that consist of a name and a single value. The value is a character string that may become part of SAS code when the macro is processed. The value can change at any time in the SAS job. Macro program statements are statements that are preceded by a `%` sign. They are executed by the macro processor and control the macro execution (for example, `%MACRO`, `%MEND`, `%DO`).

A macro expression is a sequence of macro variable names, constant text, and/or macro function names linked together by operators and, where appropriate, by parentheses. A macro call can also be viewed as a macro expression. A macro function reference is a function that assists in processing the text used in macros and in macro variable values (for example, `%QUOTE` that allows quotes to be accepted as part of the text).

A number of SAS macro program statements are available that allow the macro processing to be quite powerful and flexible. Only those needed for the examples are covered here.

A SAS macro is invoked (called) from SAS programs with the %macroname command. A semicolon is not necessary. This allows the invocation to also be used in macros and in macro variable values (for example, `%QUOTE` that allows quotes to be accepted as part of the text).

A SAS macro is invoked (called) from SAS programs with the %macroname command. A semicolon is not necessary. This allows the invocation to also be used in macros and in macro variable values (for example, `%QUOTE` that allows quotes to be accepted as part of the text).

The first use of the macro facility with the SAS/IMS-DL/I interface may be for the input field descriptions of the segment being retrieved. For example, instead of each person accessing the CUSTOMER segment of the ACCOUNT data base with the INPUT statement found in the example above, the CUSTOMER segment can be specified with the macro facility as follows:

```
%MACRO CUSTOMER;
  @1 SSNUMBER $CHAR11.
  @12 CUSTNAME $CHAR40.
  @52 ADDR1 $CHAR30.
  @172 HPHONE $CHAR12.
%MEND CUSTOMER;
```

The SAS program would then invoke the CUSTOMER macro with the following statement:

```
INPUT SSNUMBER CUSTNAME).
```

For an update of the DL/I ACCOUNT data base CUSTOMER segment the CUSTOMER macro above can be invoked as follows:

```
PUT CUSTOMER;
```

For reference in sample programs that follow, examples of the other macros used for INPUT and PUT statements are shown below:

```
%MACRO CHECKACC;
  @1 ACNUMBER $CHAR12.
  @13 BALANCE $D.
  @18 SYSDATE $MDY.
  @26 SYMTOTAL $D.
%MEND CHECKACC;
```
DATA &DATANAM is the first SAS code included in the CUSTSEQ macro. &DATANAM tells the macro processor to substitute the first positional parameter specified in the macro invocation into the DATA statement as the name of the data set to receive the output.

INFILE CUSREAD DLI is the SAS code needed to specify the PSB CUSREAD and the DLI option that tells SAS CUSREAD is a DL/I PSB instead of a DName.

INPUT %CUSTOMER invokes the CUSTOMER macro that was defined earlier to be used as the input. When executed, this INPUT statement causes a GN (get-next) call to be issued. The specified PCB is sensitive only to the CUSTOMER segment, so the get-next calls retrieve only CUSTOMER segments.

The next SAS code could optionally be a check of the DLI status code returned after each get-next call. For simplicity, no status code check is included in this example.

PROC SORT DATA=&DATANAM causes the output data set from the DATA step execution to be sorted. The name specified as the first positional parameter in the macro invocation is substituted for the &DATANAM.

BY &BYSRT specifies the variable in the data set that the data set is to be sorted by. The second positional parameter specified on the macro invocation will be substituted for the &BYSRT.

The PROC PRINT statement causes the output data set to be printed. VAR &VAR specifies which variables to print. If the VAR statement was omitted, all the variables in the data set would be printed. The &VAR allows all variables named in the macro invocation between the comma after the variable name for &BYSRT and the commas before the positional parameter &IDPRT to be substituted to tell the SAS System the variables to print. The &IDPRT in the PROC PRINT statement allows the fourth positional parameter named in the macro invocation to be substituted to tell the SAS System the ID to use in the print of each observation.

TITLE "&T" allows the T= keyword parameter value given in the macro invocation to be used for the title of the printed report.

Examples of the invocation of the CUSTSEQ macro could be

$CUSTSEQ(CUSTLIST,CUSTOMER,PHOME OPHONE,CUSTNAME,
&DATANAM=YOURDATA, &IDPRT=SIDPRT, &T=CUSTOMER PHONE LIST);

This produces a listing of all customers, sorted by CUSTNAME, including the customer name, home phone, and office phone information.

$CUSTSEQ(CUSTOMER,ADDRES,CUSTZIP,STMTDATE,STMTBAL,
&DATANAM=YOURDATA, &IDPRT=SIDPRT, &T=CUSTOMER ADDRESS LIST);

This produces a listing of all customers sorted by CUSTZIP within CUSTSTAT within CUSTLAND. A value for the positional parameter &IDPRT is omitted; therefore, the ID is the number of the observation. &DATANAM becomes CUSTLAND. ADDRESS, which is a two-level permanent SAS data set name set.

As mentioned earlier, a path call to a DL/I data base returns multiple segments as specified in the SSAs named in the call to the I/O area in hierarchical order. The following example is an example of a sequential read with path calls to retrieve the CUSTOMER and CHCKACCT segments together as defined in a macro named CUSTPATH:

MACRO CUSTPATH;
DATA &DATANAM;
RETAIN SSA1 'CUSTOMER*D' 'SSA2 'CHCKACCT' ;
INFILE ACEREAD DLI DROP-(SSA1,SSA2) STATUS=ST;
INPUT CHCKACCT;
IF _ERROR_ THEN DO;
_ERROR_ = 0;
IF ST ^= "GR" THEN STOP;
PUT _ALL_;
ABORT 88B ABEND;
END;
END;
END;
END;
END;
END;
END;
The following are examples of invoking the CUSTPATH, SORTPATH, and PRNTPATH macros:

CUSTPATH(ACCOUNT) PROC PRINT;

The macro CUSTPATH is invoked and executed to generate the temporary data set ACCOUNT that is then printed with the print including all variables.

CUSTPATH(ACCOUNT) PRNTPATH();

The same processing as above takes place; the omission of the parameters for the PRNTPATH macro causes defaults to be taken.

CUSTPATH(ACCOUNT) ISORTPATH(ACCOUNT, CUSTNAME) PRNTPATH(ACCOUNT, CUSTNAME, T=CUSTOMER PHONE LIST)

In this example the data set ACCOUNT is sorted by the variable CUSTNAME before the print is done.

CUSTPATH(ACCOUNT) CUSTREAD; PRNTPATH(ACCOUNT, CUSTREAD, CUSTNAME, T=CUSTOMER PHONE LIST)

The ANUMBER and address information is printed along with the CUSTNAME that is specified as the ID, and a title is also included.

CUSTPATH(ACCOUNT) CUSTREAD; CUSTREAD; PRNTPATH(ACCOUNT, CUSTREAD, CUSTNAME, T=CUSTOMER PHONE LIST)

No title is given and CUSTLAND is printed as the ID.

The macro facility with the interface also allows the user of the SAS code defined in the macros to furnish the PSB and segment information for further flexibility. Examples of possible macros using this are shown below.

CUSTPATH(ACCOUNT) ACCOUNT PROC PRINT;

Example SAS programs invoking the macros IMSSEG1, IMSSEG2, or IMSSEG3 follow:

CUSTPATH(ACCOUNT) CUSTREAD; IMSSEG1(ACCOUNT, SSASET, PSBNAME, MACROSEG);

DATA ACCOUNT;
RETAIN ASET QUOTE('SETVAL');
INFILE PSBNAME DLI ASET=ASET;
INPUT ASET MACROSEG;
STOP;

CUSTPATH(ACCOUNT) ACCOUNT PROC PRINT;

ACCTREAD becomes the name of the SAS data set output by the DATA step. CUSTREAD is the segment name for the unqualified SSA to be used. ACCOUNT is the SAS data set name in the INFILE statement. CUSTREAD is the macro to be invoked in the INPUT statement.

CUSTPATH(ACCOUNT) ACCOUNT PROC PRINT;

ACCTREAD becomes the name of the SAS data set output by the DATA step. CUSTREAD is the segment name for the unqualified SSA to be used. ACCOUNT is the SAS data set name in the INFILE statement. CUSTOMER is the macro to be invoked in the INPUT statement.

CUSTPATH(ACCOUNT) ACCOUNT PROC PRINT;

ACCTREAD becomes the name of the SAS data set output by the DATA step. CUSTREAD is the segment name for the unqualified SSA to be used. ACCOUNT is the SAS data set name in the INFILE statement. CUSTREAD is the macro to be invoked in the INPUT statement.

In these examples no call function has been specified; therefore, the GN (get-next) call is issued and the data base is read sequentially. More flexibility could be given by adding a parameter in the macro definition for the call function and adding the appropriate code for the call function to the DATA step. The invocation could then provide for example, a GU (get unique) call function and one or more qualified SSAs, thus, allowing random access of the data base. The macro could still allow the GN sequential access to be used.

In these examples no call function has been specified; therefore, the GN (get-next) call is issued and the data base is read sequentially. More flexibility could be given by adding a parameter in the macro definition for the call function and adding the appropriate code for the call function to the DATA step. The invocation could then provide for example, a GU (get unique) call function and one or more qualified SSAs, thus, allowing random access of the data base. The macro could still allow the GN sequential access to be used.

A random read macro definition follows. The second positional parameter is used for the name of a data set to be used for input. This input data set contains the observations that have an SSNUM (SETKEY) to be used for qualifying SSA1. The qualified SSA could also be included in the macro as a parameter to allow further flexibility.
MACRO CUSTRAND(DATANM, RANDDATA);
DATA &DATANM;
SET &RANDDATA;
INFILE CUSREAD DLI SSA=SSA1 CALL=FUNC;
FUNC = 'GU';
SSA1 = 'CUSTOMER(SNUMBER = II SETKEY II)';
INPUT SCUSTOMER;
END CUSTRAND;

A SAS program to invoke the macro follows:
CUSTRAND(CUSREAD, CUSTOMER, SSN)

A similar example with random update capabilities follows:
MACRO CUSTUPDT(DATANM, RANDDATA);
DATA &DATANM;
SET &RANDDATA;
LENGTH SSA 1 $31;
INFILE ACCUPDT DLI SSA=SSA1 CALL=FUNC DBNAME=DB;
DB = 'ACCOUNT';
FUNC = 'GU';
INPUT;
IF _ERROR_ THEN ABORT 777 ABEND;
FUNC = 'REPL';
SSA1 = '';
FILE ACCUPDT DLI;
PUT 81 _INFILE_ 852 NEWADDR1 $CHAR20,
852 NEWADDR2 $CHAR20, 812 NEWCITY $CHAR28,
812 NEWSTATE $CHAR2, Q162 NEWZIP $CHAR10.
IF _ERROR_ THEN ABORT 888 ABEND;
END CUSTUPDT;

The following SAS program could be used to invoke the macro above:
CUSTUPDT(UPDTCUS, NEW, ADDRESS)

CONCLUSION
As you can see, you can use the SAS macro facility in many ways to allow a nonexperienced user to access DL/I data bases through the SAS/IMS-DL/I interface. The SAS macro facility can also help the experienced user to use SAS code in a more powerful manner and can prove to be quite a time saver. This can be accomplished with macros that are completely defined to meet the users needs, or with macros defined with parameters to provide greater flexibility. As mentioned earlier, the macro facility provides numerous macro statements and other macro capabilities to allow more flexibility than is covered in this paper. Examples and explanations of these can be found in the SAS User's Guide: Basics, Version 5 Edition in the SAS Macro Language chapter. The SAS/IMS-DL/I User's Guide, 1984 Edition is recommended for detailed explanations and examples of the interface.

SAS and SAS/IMS-DL/I are registered trademarks of SAS Institute Inc., Cary, NC, USA.