Enterprise Miner
C and Java Score Code Basics

Last updated July 11, 2006

Table of Contents

SAS/Enterprise Miner C and Java Score Code

SAS/Enterprise Miner provides score code generated by SAS/Enterprise Miner tools into the C programming language and the Java language. The C and Java score code together with the source code and binaries distributed with Enterprise Miner, can be complied in your preferred C, C++ or Java development environment. This allows experienced C, C++ or Java programmers to greatly extend the functionality of a huge number of new and existing programs by embedding the power of SAS/Enterprise Miner predictive analytics.

After you run an Enterprise Miner modeling flow, there are a number of ways to export the contents of your model along with the generated C and Java Scoring code. See Exporting the Results and the Score Node, in the Enterprise Miner help.

Supported Tools

Currently Enterprise Miner can provide C and Java for most of the scoring code it produces. Enterprise Miner provides C and Java scoring code for only the production tools distributed with SAS/Enterprise Miner that produce SAS data step scoring code. Specifically, C and Java scoring code is provided, only for the following SAS/Enterprise Miner tools.

Sample Tools:
Sample Node
Data Partition Node
Explore Tools:
Variable Selection Node
Cluster Node
Multiplot Node
Association Node
Modify Tools
Transform Variables Node
Filter Node
Impute Node
Principal Components Node

Model Tools:
Regression Node
Dmine Regression Node
Decision Tree Node
Rule Induction Node
Neural Network Node
AutoNeural Node
DMNeural Node
TwoStage Node
Ensemble Node
Assess Tools:
Score Node
Model Comparison Node
Credit Scoring Tools:
Credit Exchange
Interactive Grouping
Scorecard

Most EM flows that produce Data step, C and Java code. However any EM flow that produces SAS step code (Proc or Data statements) will not produce C or Java code. Only the code generated by the EM tools is provided in C and Java form. Any other tools or nodes e.g. SAS Code tool or user-defined tools, that are not listed below, will have any code they produce omitted from the generated C and Java code. In such cases, the generated C code will contain a comment to indicate the location of the omitted input. The C comment may appear like the following:

```
/*-------------------------------------------------------------**/
/* insert c code here*/
/* datastep scorecode for emcode*/
/* is not supported for conversion*/
/*-------------------------------------------------------------**/
```

In most cases, it should be possible for the user to insert C code to take the place of the omitted code

The Java code generated will also omit any score code from tools not in the list. In such cases a call to a specific method is inserted in the code in place of the omitted code and source code for an empty stub method is produced to allow the user to insert hand coded Java code for any omitted code.

The Generated Code

The C and Java code generated by Enterprise Miner, will reflect the same algorithm and operations used in the SAS Data Step scoring code produced by the flow. The generated C and Java code represents only the functionality explicit in the SAS data step scoring code. The C code produced conforms to the "ISO/IEC 9899 International Standard for Programming Languages – C" and the Java code conforms to the Java Language Specification, published in 1996 by Addison-Wesley.

While the generated scoring code can represent the core analytics of a scoring system it should not be confused with a complete scoring system. In either C or Java languages, a user-written program must provide a suitable environment invoking the generated code for scoring.

The generated C and Java scoring code is generated as several output files. They may include all of the following:

- Cscore.xml is the XML description of the model that produced the code and the generated C code. It is valid XML. No DTD supplied.
- DB2_Score.c is C code for DB2 scalar User Defined Functions for each of the output variables present in the scoring code.
- DS.class is the actual data step code generated directly to Java binary code, there is no Java source code supplied.
- DS_UEXIT.java will be generated only if code from unsupported tools was omitted from the generated Java code. This Java source code is a template the customer can use to provide their own code for the omitted tool or node.
- Jscore.xml is an XML description of the model that produced the code and the generated Java code. It is valid XML. No DTD supplied.
- JscoreUserFormats.java is the Java source code that supports any user written formats that may be used in the model. It is Java source code and must be compiled before it can be executed.
- Score.c is the model score code generated as a C function. It is C source code and must be compiled before it can be executed.
- Score.java is the Java source code that implements the interface to DS.class. It is Java source code and must be compiled before it can be executed.

DB2 User Defined Functions

In addition to generating the scoring algorithms developed in Enterprise Miner Models the C Scoring component can generate the C code for IBM DB2 User Defined functions for the useful predicted values defined by a model. IBM User Defined Functions or UDFs is a mechanism with which you can write your own extensions to SQL. While the functions built-in to DB2 are useful, they do not offer the customized power of SAS analytics. The Enterprise Miner generated UDFs allow you to greatly increase the efficiency, versatility and power of your RDBMS. The key advantages of using UDFs are performance, modularity and the object oriented UDF process.
The Enterprise Miner generated UDF code is matched to each specific model's training data and generated C scoring function. There are several ways one might go about scoring in DB2 and User Defined Functions and the code generated by Enterprise Miner represents but one approach. The generated source code for the UDFs is simple but expandable. Contained in the comments of the UDF source code are examples or templates of the SQL commands to register and invoke the UDFs generated. The Enterprise Miner generated score function can return many values that are not logically useful in the context of scoring. The UDFs generated for a specific model are limited to those returning the values of the scoring outputs deemed likely to be of interest. The scoring output variables names are created by concatenating a prefix (for each kind of computed variable) with the name of the corresponding target variable or decision data set. The Enterprise Miner scoring output variables for which UDFs will be produced are those that begin with:

D_	Decision chosen by the model
EL_	Expected loss of the decision chosen by the model
EP_	Expected profit of the decision chosen by the model
I_	Normalized category that the case is classified into

P_	Predicted values and estimates of posterior probabilities

And those scoring output variables with the following names:

NODE	
SEGMENT	
WARN	indicates problems computing predicted values or making decisions
EM_CCF	average CCF (credit cost factor) value
EM_CLASSIFICATION	
EM_DECISION	
EM_EVENTPROBABILITY	
EM_EXPOSURE	average Exposure value
EM_FILTER	identifies filtered observations
EM_LGD	average LGD (Loss Given default) value
EM_PD	average predicted value
EM_PREDICTION	
EM_PROBABILITY	
EM_PROFITLOSS	
EM_SEGMENT	
SCORECARD_BIN	bin assigned to each observation
SCORECARD_POINTS	total score for each individual
SOM_DIMENSION1	identifies rows in the SOM
SOM_DIMENSION2	identifies columns in the SOM
SOM_ SEGMENT	identifies clusters created by a SOM

Most of the code in the generated UDFs is to handle the conversion of data types and missing values before and after the score function is called. The first function in the generated UDF code (load_indata_vec) is invoked by all the UDFs in the file to load the input data vector for the score function.

Implementation Issues and Limitations

It should be emphasized that creating a scoring application is a very complex and highly advanced task that requires expertise in a several areas. The likelihood of successfully implementing a scoring system incorporating Enterprise Miner generated C or Java code will be exactly proportional to your fluency and experience with DB2, SQL, the C language, application development and your attention to detail. Even with experience and expertise, embedding advanced analytics into a data base to create a high performance scoring application is, in many ways, very close to the limits of current technology. Any time you work close to the edges of a technology you can gain unexpected rewards but, you should also expect problems. Testing of both the application and the UDFs will be critical to the success of any such project.

As documented in all current releases of DB2, in each reference to a function, whether it is a UDF, or a built-in function, the arguments can number from 0 to 90. This is of course a critical limitation for data mining where it is common for hundreds of values to be needed for even a simple model. While Enterprise Miner will produce UDF code with more than 90 arguments, such code it will not be usable. IBM is aware of this issue and is investigating solutions to the problem.

Data Types

The Enterprise Miner generated UDFs only accept 2 SQL data types, DOUBLE and VARCHAR. Since most data bases will contain data in other data types, great care must be taken when converting the DB2 data types for the UDF calls. DB2 provides functions to cast most data types to DOUBLE or VARCHAR. These should be used as appropriate. Another approach is to make any necessary data type conversions in the ETL step for both the training data and the data to be scored. The Enterprise Miner generated UDF source code can also be modified to handle the type conversions for the score function.

SAS System Formats Support

SAS formats are how SAS writes data for display or analysis. There are two basic types of formats. The pre-defined format supplied with all SAS Systems and those formats defined in code. The formats defined in code are supported in the Enterprise Miner generated code. The SAS System formats are supported through libraries distributed as part of Enterprise Miner and the SAS System.

Formats in C

The C scoring code optionally supports the use of SAS System Formats through the SAS Standalone Formats product. The SAS Standalone Formats libraries are distributed with SAS in the Client Media on SAS Client-Side Components Volume 1. See the Standalone SAS System Formats installation instructions for more details

The Standalone Formats product contains a header file that will be needed for compilation, the compiled code needed to link the Enterprise Miner generated, score function, and a set of callable routines that the score code may invoke. Most of the SAS System formats have been packaged into these libraries of loadable routines. These routines are loaded on demand at run time. They are intended to operate from Enterprise Miner generated scoring code and do not depend on a SAS System environment in any way. The SAS Standalone Formats needed for compiling, linking and running Enterprise Miner generated C code are named as follows:

jazwfbin	binary formats
jazwfdte	date formats
jazwfmsc	miscellaneous formats
jazwftme	time formats
jazwfuwf	originally UWF-style modules
jazxfbrg	the format loading an invocation routines
jazz.h	the stand alone formats header file

On windows these sets of routines will have the ".dll" file name extension and the link library is a separate file jazxfbrg.lib.

In addition to the files listed above and depending on your target system, you may also find additional files in your installation of the Standalone Formats files. These are files that support the SAS System Informats which are NOT used by Enterprise Miner generated code. These include:

jazwibin	binary informats
jazwidte	date informats
jazwimsc	miscellaneous informats
jazwiuwi	originally UWI-style informats

The function/application is linked to jazxfbrg so that at runtime the code in jazxfbrg can dynamically load the rest of the needed routines to support the SAS System formats. While only jazxfbrg may need to be present when linking the function/application, all of the files must be available at runtime.

For the Standalone formats dynamic loading is accomplished through calls to standard System routines. Dynamic loading is an advanced topic in any C environment. The exact procedures, options and environment variables used in compiling, linking and running dynamically loaded code are different for every compiler, linker and operating system. For example, on Windows, shared libraries are loaded from the environment variable PATH. This environment variable must be set to contain the path for the Standalone Formats shared libraries (jazwf*). The value of this environment variable must be the fully qualified directory name for the directory that holds the Standalone formats. On Solaris systems the Standalone Formats are dynamically loaded from shared libraries via the environment variable LD_LIBRARY_PATH. On HP/UX, on the other hand, uses a slightly different environment variable, SHLIB_PATH. A thorough understanding of your target systems procedures for compiling, linking and running with dynamically loaded code will be required to successfully exploit the Standalone Formats and the code generated by the Enterprise Miner C Scoring component.

The Standalone Formats product supports most but not all of the SAS System formats. The list of formats is as follows:

$BINARY	MMYYN	WEEKDAY
$CHAR	MMYYP	WORDDATE
$EBCDIC	MMYYS	WORDDATX
$HEX	MONNAME	WORDF
$OCTAL	MONTH	WORDS
BEST	MONYY	YEAR
BINARY	NEGPAREN	YYMM
COMMA	NENGO	YYMMC
COMMAX	OCTAL	YYMMD
DATE	PERCENT	YYMMN
DATETIME	PIB	YYMMP
DAY	PK	YYMMS
DDMMYY	QTR	YYMMDD
DOLLAR	QTRR	YYMON
DOLLARX	RB	YYQ
E	ROMAN	YYQC
FRACT	SSN	YYQD
HEX	S370FIB	YYQN
HHMM	S370FPD	YYQP
HOUR	S370FPIB	YYQS
IB	S370FRB	YYQR
JULIAN	TIME	YYQRC
MMDDYY	TOD	YYQRD
MMSS	F (referred to as "w.d"	YYQRN
MMYY	format)	YYQRP
MMYYC	WEEKDATE	YYQRS
MMYYD	WEEKDATX	Z

Formats known to be unsupported:

$VARYING	- Not load able from the format package. (Not supported)
DOWNNAME	- Not load able from the format package. (Not supported)
JULDAY	- Not load able from the format package. (Not supported)
PD	- No binary formats supported by proc cscore. (SAS hexadecimal notation is not handled)

For those environments where the Standalone formats are not available or not desired, it should be possible for an experienced C programmer to modify the source code in cscore.h header file that is distributed with EM in SASROOT/dmine/sasmisc, to remove the dependency on the Standalone Formats or to support any format they want, in their own C code. If the customer were to write their own formats functions, it is possible to integrate those functions into the logic that handles formats in cscore.h. The cscore.h already contains two examples of such C formatting code, partial support for $CHAR and BEST formats. If the customers situation allows them to accept the limitations of those examples, no padding for $CHAR and no scientific notation for BEST, they could use the example formats as is, and add any other formats that are needed. Then, any generated C scoring code derived from training data that contained only those formats and compiled with that header file, would support those formats.

In cases where Standalone Formats support is not desired, the dependency on the Standalone formats support can be removed. This can be accomplished by modifying a copy of the cscore.h header file that is distributed with EM in SASROOT/dmine/sasmisc. In the cscore.h file the preprocessor symbol FMTLIB is set to 0 to disable the Standalone formats support.

Formats in Java

The Java scoring code supports the use of SAS System Formats through the Java code distributed with Enterprise Miner in SASROOT\dmine\sasmisc. They are:

sas.text.jar	- routines for both SAS system and user written formats
sas.core.jar	- routines for both SAS system and user written formats
sas.entities.jar	- routines for both SAS system and user written formats

These class libraries and others must be made available to the Java VM in order to run the Enterprise Miner generated Java score code.

Simplistic Scoring Example

Both the Enterprise Miner tools used and the training data directly effect the scoring code produced by Enterprise Miner. Any variation in the tools, their options, or the data, can produce changes in the generated code. So, the score code may be unique for every flow. The following example is purely for illustrative purposes and not intended to illustrate a real application. For this example the code will be generated running EM Client on Windows and the code extracted the moved to Solaris to be compiled and run.

Example Folders

The following steps take place on the machine where the EM client is running, unless otherwise noted.

1. Create a folder in C:\Temp called "ScoreCode".
2. Create a folder in C:\Temp\ScoreCode called “cscore”.
3. Create a folder in C:\Temp\ScoreCode called “jscore”.

EM Client

4. Start the EM Client and open a project and diagram.
5. Create a new modeling flow starting with an Input Data Source Node that references the sample SAS table DMEXA1 from the SAMPSIO library. Make sure to use the basic advisor to set the initial measurement levels and roles for the data source. Then in the columns meta data, set the role for the PURCHASE variable to be Target.
6. From the Enterprise Miner Explore tab add a Variable Selection Node on the right of the Input Data Source Node.
7. From the Enterprise Miner Model tab add a Regression Node on the right of the Variable Selection Node
8. From the Enterprise Miner Assess tab add a Score Node on the right of the Regression Node
9. Connect the Nodes from left to right
10. Select the Score Node, right click it and select Run
11. When the run completes, select the EM Options Menu and select “Preferences…”
12. For this example in the Model Package Options make sure the boxes are checked for “Generate C score Code” and “Generate Java Score Code”.
13. In the “Java Score Code Package” text box enter “dom.corp.proj” (without the quotes).
14. Click OK.
15. Right Click that terminal Score Node and select “Create Model Package…”
16. For this example use “example” as the Model Package Name
17. After the model package has been created you should see the Run Status dialog, select OK.
18. Navigate the Enterprise Miner project panel to the Model Package folder and expand the folder.
19. Select the “example” package folder.
20. In the Enterprise Miner File menu select Save as…
21. Name the file “example” and save it in the folder “C:\Temp\ScoreCode” created in step 1.
22. Open the model package file (example.spk) with WinZip or the zip file utility of your choice.
23. Then open the PATHSCORE.spk
24. Then open the CSCORE.SPK and extract all the files it contains into C:/Temp/ScoreCode/cscore.
25. Then open the JSCORE.SPK and extract all the files it contains into C:/Temp/ScoreCode/jscore.

Since the generated C and Java code is handled differently, the next steps in this example will be determined by your choice of Scoring with C Code or Scoring with Java Code.

Scoring with C Code

The generated C code and the C code distributed with Enterprise Miner can be compiled in most modern C or C++ development environments. The result of compilation will vary depending upon the compiler and the options used. In particular some compilers may produce warning messages for type conversions that represent generic risks. It is up to the users to properly investigate and handle as appropriate any warnings. An example of the generated C scoring code is in Appendix C

26. Locate the Client Media, SAS Client-Side Components Volume 1 CDROM that was shipped with the SAS System. This CDROM contains the Standalone Formats files for all supported systems.
27. To avoid starting the SAS Software Navigator when you load the CDROM, hold down the shift key when you insert the CDROM. If auto insert notification is enabled on your machine and the SAS Software Navigator starts, it will prompt you for a language. For this example choose “English” and click OK then select Exit from the file menu.
28. In a file system (Explorer or My Computer) select the CDROM drive with the SAS Client-Side Components Volume 1 CDROM loaded, right click the drive and select Open.
29. Navigate to “\client1cd\safmts”.
30. Select “safmtss64.tar”. Right click the file and select Copy.
31. Paste the file in C:\Temp\ScoreCode, created in Step 1.
32. Locate within the folders created when installing the Workspace Server for Enterprise Miner, the "dmine\sasmisc" folder. If you accepted the defaults when installing, that would be "C:\program files\sas\sas 9.1\dmine\sasmisc". Copy two files, "cscore.h" and "csparm.h" from there to the folder created for this example C:\Temp\ScoreCode.

33. On the target Unix System, in your HOME directory create a directory named "example".

34. FTP or otherwise copy to the “example” directory, all of the following files:
    C:\Temp\ScoreCode\safmtss64.tar
    C:\Temp\ScoreCode\csparm.h
    C:\Temp\ScoreCode\cscore.h
    C:\Temp\ScoreCode\cscore\ Score.c
    Most FTP clients will take care of the carriage-returns in the windows text files. If not, most Solaris systems provide a dos2unix command. It is usually found in the "/bin" directory.

On Unix

From here on all steps will be on the Unix system unless specifically noted.

35. To unpack the Standalone formats TAR file change directory to the "example" directory and enter
    ```
    tar xf safmtss64.tar
    ```
    This should result in the creation of the "safmts" directory containing all the Standalone formats files.

36. By default the cscore.h header file contains definitions and values that are windows specific. Edit the "cscore.h" file to change these definitions and values for Solaris.

37. Change line 70 of cscore.h to look like
    ```
    unsigned char NaN[8] = {0xff,0xff,0xfe,0,0,0,0,0};
    ```
    Each operating system has its own value for missing. Cscore.h must be modified for each system. See Missing Values for more details.

38. Change line 78 of cscore.h to look like
    ```
    #define SFDKeyWords
    ```
    Some systems require special directives to correctly store the function name in an objects export table.

The cscore.h header file provides a macro "SFDKeyWords" for those systems. By default it is set to the Windows directive. For systems other than Windows or if you are creating an object other than a DLL you will need to modify this macro. If you situation does not require any directives, change the #define for the SFDKeyWords macro to define SFDKeyWords as blank.

39. View the ".c" file and inspect it to determine how it should be called. Meta-data describing both the model and the function is available in the generated Cscore.xml file. By default the generated function accepts two pointers as arguments. The first argument points to an array of input data values. The second points to and array of output data values. The memory, required for each data value, must be allocated by the calling program. Both arrays must be composed of the PARM data structure defined in the csparm.h header file. Each array element must contain either a double or a char *. The length of the memory referenced by each char* can be extracted from the Cscore.xml or by inspection of the original training data set. If the appropriate memory is not allocated prior to calling score() the results are undefined. The position of each data value in its array can also be extracted from the XML or inferred from the #defines for the variable names found in the generated C code. These variable names are usually taken directly from the training data or derived from names in the training data. The generated C file for this example looks like the code in Appendix C

40. Create a main program to invoke the score function. Such a program might look like the code in Appendix B. For this example, name the file csbasic.c

41. If GNU C version 3.2.3 (sparc-sun-solaris2.8) is available, it is usually installed in /usr/local/bin/ and it can be used to compile and link with a single command like:
    gcc -m64 -ansi -I$HOME/example/safmts csbasic.c Score.c -lm $HOME/example/safmts/jazxfbrg

    Where:

-m64	selects the 64-bit environment
-ansi	turns off features of gnu C which are incompatible with ANSI C
-I	specifies the path to search for part of the formats support, the jazz.h header file.
csbasic.c	is the main C program to be compiled.
Score.c	is the C scoring code generated by Enterprise Miner.
-lm	specifies the math link library
jazxfbrg	is an object library from which the Standalone formats objects are linked.

    This should produce a single executable file called "a.out".

42. The Standalone formats support routines are dynamically loaded from some of the files in the safmts folder. On Solaris the path searched for dynamically loaded code can be modified by setting the environment variable LD_LIBRARY_PATH. This environment variable (which is analyzed at process start-up) can be set to a colon-separated list of directories. In this case enter:
    ```
    LD_LIBRARY_PATH=.:$HOME/example/safmts
    ```

43. Enter: `export LD_LIBRARY_PATH`

44. To execute the main program enter:
    ```
    a.out
    ```

    The output should look something like the following:
    ```
    $ a.out

    >> First observation...
    csEM_PREDICTION = 0.701972
    csG_STATECOD = 8.000000
    csP_PURCHASE = 0.701972
    cs_WARN_ =

    >> Second observation...
    csEM_PREDICTION = 0.794347
    csG_STATECOD = 8.000000
    csP_PURCHASE = 0.794347
    cs_WARN_ =
    $
    ```

Scoring with Java Code

The code output by Enterprise Miner includes Java source and binary code. A Java development environment will be a pre-requisite using the generated Java code. The Java code distributed with and produced by Enterprise Miner was developed with JAVA™ 2 SDK, Standard Edition, Version 1.2 Software. There are no know reasons to expect any recent version of Java to be a problem, but no other version has been tested. The JDK (Software Development Kit) is available at http://java.sun.com/.

Java Package Name

The Enterprise Miner generated code contains an assigned package name. The package name effectively becomes the first part of the absolute class name. When compiling Java source code with a package name the Java compiler (javac) will search for the source and class files related by the package name in a path relative to the current working directory/folder. The Java compiler uses the package name to form a hierarchical path for each related file. If for example, the package name is the default of "eminer.user.Score", the Java compiler will search for the package's files in "eminer\user\Score. In order to compile the generated Java source code, all of the generated Java files (Jscore.xml not required) must be placed in a directory/folder tree that looks like the package name "eminer.user.Score". The default package name can be changed in the Enterprise Miner Client before the flow is run by selecting Options then Preferences and filling in a package name of at least 2 levels.

Required Java Class Libraries

If you wish to compile or run Enterprise Miner generated Java code you will have to make the Class libraries distributed with Enterprise Miner available to Java. One methods is to set the CLASSPATH environment variable to contain the required Class libraries. These libraries must be listed in the CLASSPATH environment variable, by their absolute path names. They are distributed in the SAS image under SASROOT/dmine/sasmisc and they are:

Class Library	Description
dtj.jar	- compile time and runtime routines also used by Enterprise Miner for code generation
sas.analytics.eminer.jsutil.jar	- JscoreException and Jscore interface classes binaries and source
sas.text.jar	- routines for both SAS system and user written formats
sas.core.jar	- routines for both SAS system and user written formats
sas.entities.jar	- routines for both SAS system and user written formats

1. Locate within the folders created when installing the Enterprise Miner Workspace server, the "dmine\sasmisc" folder. If you accepted the defaults when installing, that would be "C:\Program Files\SAS\SAS 9.1\dmine\sasmisc ". Copy the JAR files found there "dtj.jar", "sasjsutl.jar", "sastext.jar", "sas.core.jar", and "sas.entities.jar" from there to the folder created for this example C:\Temp\ScoreCode.
2. On the Unix System in your HOME directory create a directory named "example".
3. FTP or otherwise copy all the JAR files from the windows folder C:\Temp\ScoreCode and all the Java and related .class and .xml files from the subfolder "jscore" to the "example" directory in your HOME directory on the target Unix system. When finished the list of files in the example directory should look like:

```
$ ls -1
DS.class
Jscore.xml
JscoreUserFormats.java
Score.java
dtj.jar
sas.core.jar
sas.entities.jar
sas.text.jar
```

sasjsutl.jar

Note: The ".java" files will need to have the carriage-returns used in Windows text removed. Many FTP clients will handle the removal of carriage-returns in text files. If not, most Solaris systems provide a dos2unix command. It is usually found in the "/bin" directory.

On Unix

From here on all steps will be on the Unix system unless specifically noted.

1. Determine the package name of the generated code. One way to determine the package name is to edit the Jscore.xml and find the Java class name. In this case it the class name is "dom.corp.proj.Score". Remove the last qualifier "Score" and the remainder is the package name.
2. In the example directory create a directory tree starting with the "dom" directory. Make subdirectories for each subsequent qualifier in the package name in the preceding qualifier's directory. The resulting path should look like:
   $HOME/example/dom/corp/proj
   Move all of the generated ".java" and ".class" files from the example to the leaf folder "proj".
3. In the leaf directory "proj", create a Java main program to create an instance of the generated scoring class, and provide input data, invoke the score method and handle the outputs for scoring. A simplistic example of such a program might look like the code in Appendix E. For this example, name the file Jsbasic.java.
4. Set the "CLASSPATH" environment variable to contain absolute paths for the JAR files distributed for Enterprise Miner Java scoring, sas.analytics.eminer.jsutil.jar, dtj.jar, sas.text.jar, sas.core.jar and sas.entities.jar. One way to set environment variables is at a prompt, on a single line, enter something like:
   ```
   export
   CLASSPATH=.:$HOME/example/dtj.jar:$HOME/example/sas.analytics.eminer.jsutil.jar:$HOME/
   example/sas.core.jar:$HOME/example/sas.entities.jar:$HOME/example/sas.text.jar
   ```
5. Once the CLASSPATH is set and the *.java and DS.class files put into the directory structure or folder tree indicated by the package name, change directories to the parent directory or folder of the package (where dom is) and invoke the Java compiler on the source files. In this case that command would be like:
   javac dom/corp/proj/*.java

   The result will be the creation of a set of Java class files that implement the Jscore interface.

6. After the Jsbasic Main program and the Enterprise Miner generated source code is compiled, copy the Jsbasic.class file from /dom/corp/proj to the example directory, make it your current working directory.
7. On the command line enter java Jsbasic. The output should look something like the following:

   ```
   >> First observation...
   EM_CLASSIFICATION = YES
   EM_EVENTPROBABILITY = 0.723472623240661
   EM_PROBABILITY = 0.723472623240661
   G_STATECOD = 1.0
   I_PURCHASE= YES
   P_PURCHASENO = 0.276527376759339
   P_PURCHASEYES = 0.723472623240661
   U_PURCHASE = 1.0
   _WARN_ =

   >> Second observation...
   EM_CLASSIFICATION = YES
   ```

EM_EVENTPROBABILITY = 0.7923360394252702
EM_PROBABILITY = 0.7923360394252702
G_STATECOD = 1.0
I_PURCHASE= YES
P_PURCHASENO = 0.2076639605747298
P_PURCHASEYES = 0.7923360394252702
U_PURCHASE = 1.0
WARN =

Notes and Tips

Missing Values

In the SAS System, numeric missing values are represented by IEEE Not-a-Number values. An IEEE Not-a-Number or NaN is an IEEE floating-point bit pattern that represents something other than a valid numeric value. These numbers are not computationally derivable, meaning that the bit pattern will never be generated by an arithmetic operation. The exact bit pattern for the NaN is different on different operating systems. This bit pattern must be set in the cscore.h header file. Within cscore.h you will find recommendations for the values to be used on the systems supported by SAS. These NaNs can be problematic if not handled correctly. Consequently when developing a scoring system it may be necessary to consider special handling of numeric data that may contain missing values. Especially in C it may be necessary to check the bits of each numeric value before performing any operation on them. Examples of C functions that perform such a check are available in cscore.h as the “nmiss” function and in this example's C main program (csbasic.c) as the “amiss” function.

Disclaimer

This document provides both code and methodologies for illustrative purposes and due caution should be used in making any assumptions as to their utility or correctness. The use or mention here in of any vendor or product should not be misconstrued as a recommendation or endorsement. This is a very simple example and not intended to illustrate a real application. It should in fact be possible to find many more suitable ways to use the scoring code produced by SAS Enterprise Miner as well as any other software mentioned in this document.

Appendix A

SAS Data Step Code

```
*-------------------------------------------------------------*;
* Macro variable identifying the scored data set;
*%let EM_SCORE_OUTPUT=;
*-------------------------------------------------------------*;
*-------------------------------------------------------------*;
* TOOL: Input Data Source;
* TYPE: SAMPLE;
* NODE: Ids2;
*-------------------------------------------------------------*;
*-------------------------------------------------------------*;
* TOOL: Variable selection Class;
* TYPE: EXPLORE;
* NODE: Varsel4;
*-------------------------------------------------------------*;
*********************************************;
**** Begin scoring code for variable selection;
*********************************************;
length _WARN_ $ 4;
label _WARN_ = "Warnings";
length _NORM1 $%DMNORLEN;
length _FORMAT $200;
_NORM1 = ' ';
%DMNORMCP(STATECOD, _NORM1 );
select(_NORM1 );
when("AE") G_STATECOD = 8;
when("AK") G_STATECOD = 0;
when("AL") G_STATECOD = 4;
when("AP") G_STATECOD = 1;
when("AR") G_STATECOD = 5;
when("AZ") G_STATECOD = 5;
when("CA") G_STATECOD = 6;
when("CO") G_STATECOD = 5;
when("CT") G_STATECOD = 6;
when("DC") G_STATECOD = 9;
when("DE") G_STATECOD = 5;
when("FL") G_STATECOD = 5;
when("GA") G_STATECOD = 7;
when("HI") G_STATECOD = 7;
when("IA") G_STATECOD = 4;
when("ID") G_STATECOD = 2;
when("IL") G_STATECOD = 5;
when("IN") G_STATECOD = 5;
when("KS") G_STATECOD = 7;
when("KY") G_STATECOD = 2;
when("LA") G_STATECOD = 7;
when("MA") G_STATECOD = 8;
when("MD") G_STATECOD = 7;
when("ME") G_STATECOD = 7;
when("MI") G_STATECOD = 7;
when("MN") G_STATECOD = 6;
```

```
when("MO") G_STATECOD = 7;
when("MS") G_STATECOD = 2;
when("MT") G_STATECOD = 7;
when("NC") G_STATECOD = 5;
when("ND") G_STATECOD = 5;
when("NE") G_STATECOD = 4;
when("NH") G_STATECOD = 1;
when("NJ") G_STATECOD = 6;
when("NM") G_STATECOD = 2;
when("NV") G_STATECOD = 5;
when("NY") G_STATECOD = 4;
when("OH") G_STATECOD = 3;
when("OK") G_STATECOD = 7;
when("OR") G_STATECOD = 7;
when("PA") G_STATECOD = 4;
when("PR") G_STATECOD = 9;
when("RI") G_STATECOD = 4;
when("SC") G_STATECOD = 3;
when("SD") G_STATECOD = 8;
when("TN") G_STATECOD = 4;
when("TX") G_STATECOD = 6;
when("UT") G_STATECOD = 6;
when("VA") G_STATECOD = 3;
when("VI") G_STATECOD = 9;
when("VT") G_STATECOD = 3;
when("WA") G_STATECOD = 3;
when("WI") G_STATECOD = 3;
when("WV") G_STATECOD = 6;
when("WY") G_STATECOD = 5;
otherwise _WARN_ = 'U';
end;
label G_STATECOD = "Group: STATECOD";
DROP _NORM1 _FORMAT;
*-------------------------------------------------------------*;
* TOOL: Regression;
* TYPE: MODEL;
* NODE: Reg;
*-------------------------------------------------------------*;
***********************************;
*** begin scoring code for regression;
***********************************;

length _WARN_ $4;
label _WARN_ = 'Warnings' ;


drop _DM_BAD;
_DM_BAD=0;

*** Check AMOUNT for missing values ;
if missing( AMOUNT ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check APPAREL for missing values ;
if missing( APPAREL ) then do;
```

```
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check BLANKETS for missing values ;
if missing( BLANKETS ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check DOMESTIC for missing values ;
if missing( DOMESTIC ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check FREQUENT for missing values ;
if missing( FREQUENT ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check HOMEACC for missing values ;
if missing( HOMEACC ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check LAMPS for missing values ;
if missing( LAMPS ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check LUXURY for missing values ;
if missing( LUXURY ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check OUTDOOR for missing values ;
if missing( OUTDOOR ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check RECENCY for missing values ;
if missing( RECENCY ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;

*** Check WCOAT for missing values ;
if missing( WCOAT ) then do;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;
```

```
*** Generate dummy variables for G_STATECOD ;
drop _0_0 _0_1 _0_2 _0_3 _0_4 _0_5 _0_6 _0_7 _0_8 ;
*** encoding is sparse, initialize to zero;
_0_0 = 0;
_0_1 = 0;
_0_2 = 0;
_0_3 = 0;
_0_4 = 0;
_0_5 = 0;
_0_6 = 0;
_0_7 = 0;
_0_8 = 0;
if missing( G_STATECOD ) then do;
   _0_0 = .;
   _0_1 = .;
   _0_2 = .;
   _0_3 = .;
   _0_4 = .;
   _0_5 = .;
   _0_6 = .;
   _0_7 = .;
   _0_8 = .;
   substr(_warn_,1,1) = 'M';
   _DM_BAD = 1;
end;
else do;
   length _dm12 $ 12; drop _dm12 ;
   _dm12 = put( G_STATECOD , BEST12. );
   %DMNORMIP( _dm12 )
   _dm_find = 0; drop _dm_find;
   if _dm12 <= '4'  then do;
      if _dm12 <= '2'  then do;
         if _dm12 <= '1'  then do;
            if _dm12 = '0'  then do;
               _0_0 = 1;
               _dm_find = 1;
            end;
            else do;
               if _dm12 = '1'  then do;
                  _0_1 = 1;
                  _dm_find = 1;
               end;
            end;
         end;
         else do;
            if _dm12 = '2'  then do;
               _0_2 = 1;
               _dm_find = 1;
            end;
         end;
      end;
      else do;
         if _dm12 = '3'  then do;
            _0_3 = 1;
            _dm_find = 1;
         end;
```

```
      else do;
         if _dm12 = '4'  then do;
            _0_4 = 1;
            _dm_find = 1;
         end;
      end;
   end;
end;
else do;
   if _dm12 <= '7'  then do;
      if _dm12 <= '6'  then do;
         if _dm12 = '5'  then do;
            _0_5 = 1;
            _dm_find = 1;
         end;
         else do;
            if _dm12 = '6'  then do;
               _0_6 = 1;
               _dm_find = 1;
            end;
         end;
      end;
      else do;
         if _dm12 = '7'  then do;
            _0_7 = 1;
            _dm_find = 1;
         end;
      end;
   end;
   else do;
      if _dm12 = '8'  then do;
         _0_8 = 1;
         _dm_find = 1;
      end;
      else do;
         if _dm12 = '9'  then do;
            _0_0 = -1;
            _0_1 = -1;
            _0_2 = -1;
            _0_3 = -1;
            _0_4 = -1;
            _0_5 = -1;
            _0_6 = -1;
            _0_7 = -1;
            _0_8 = -1;
            _dm_find = 1;
         end;
      end;
   end;
end;
if not _dm_find then do;
   _0_0 = .;
   _0_1 = .;
   _0_2 = .;
   _0_3 = .;
   _0_4 = .;
   _0_5 = .;
```

```
      _0_6 = .;
      _0_7 = .;
      _0_8 = .;
      substr(_warn_,2,1) = 'U';
      _DM_BAD = 1;
   end;
end;

*** If missing inputs, use averages;
if _DM_BAD > 0 then do;
   _LP0 =     0.50813835198372;
   goto REGDR1;
end;

*** Compute Linear Predictor;
drop _TEMP;
drop _LP0;
_LP0 = 0;

***  Effect: AMOUNT ;
_TEMP = AMOUNT ;
_LP0 = _LP0 + (   -0.00004993870147 * _TEMP);

***  Effect: APPAREL ;
_TEMP = APPAREL ;
_LP0 = _LP0 + (    0.01297149165591 * _TEMP);

***  Effect: BLANKETS ;
_TEMP = BLANKETS ;
_LP0 = _LP0 + (    0.01307320528946 * _TEMP);

***  Effect: DOMESTIC ;
_TEMP = DOMESTIC ;
_LP0 = _LP0 + (    0.00908957848067 * _TEMP);

***  Effect: FREQUENT ;
_TEMP = FREQUENT ;
_LP0 = _LP0 + (    0.02265229436888 * _TEMP);

***  Effect: G_STATECOD ;
_TEMP = 1;
_LP0 = _LP0 + (   -0.38058915743906) * _TEMP * _0_0;
_LP0 = _LP0 + (   -0.19842465513138) * _TEMP * _0_1;
_LP0 = _LP0 + (   -0.09040404754398) * _TEMP * _0_2;
_LP0 = _LP0 + (   -0.06565678604598) * _TEMP * _0_3;
_LP0 = _LP0 + (   -0.02912411263384) * _TEMP * _0_4;
_LP0 = _LP0 + (    0.00128070484317) * _TEMP * _0_5;
_LP0 = _LP0 + (    0.00897606795372) * _TEMP * _0_6;
_LP0 = _LP0 + (    0.09650736361157) * _TEMP * _0_7;
_LP0 = _LP0 + (    0.20808691085082) * _TEMP * _0_8;

***  Effect: HOMEACC ;
_TEMP = HOMEACC ;
_LP0 = _LP0 + (   -0.00550672193636 * _TEMP);

***  Effect: LAMPS ;
_TEMP = LAMPS ;
```

```
_LP0 = _LP0 + (    0.01856257780278 * _TEMP);

***  Effect: LUXURY ;
_TEMP = LUXURY ;
_LP0 = _LP0 + (    0.04536833034207 * _TEMP);

***  Effect: OUTDOOR ;
_TEMP = OUTDOOR ;
_LP0 = _LP0 + (    0.00806192635828 * _TEMP);

***  Effect: RECENCY ;
_TEMP = RECENCY ;
_LP0 = _LP0 + (   -0.00020781136645 * _TEMP);

***  Effect: WCOAT ;
_TEMP = WCOAT ;
_LP0 = _LP0 + (    0.01930625999309 * _TEMP);
*--- Intercept ---*;
_LP0 = _LP0 + (    0.43801900950123);

REGDR1:

*** Predicted Value;
label P_PURCHASE = 'Predicted: PURCHASE' ;
P_PURCHASE = _LP0;


***********************************;
***** end scoring code for regression;
***********************************;
*-------------------------------------------------------------*;
* TOOL: Score Node;
* TYPE: ASSESS;
* NODE: Score;
*-------------------------------------------------------------*;
*-------------------------------------------------------------*;
* Score: Creating Fixed Names;
*-------------------------------------------------------------*;
LABEL EM_PREDICTION= "Prediction for PURCHASE";
EM_PREDICTION = P_PURCHASE;
```

Appendix B

Sample Main Program

```
/*---------------------------------------------------------------------
 * CSBASIC - Enterprise Miner C scoring example program
 *
 * V2
 *--------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include "csparm.h"
#include <string.h>

void score ( PARM *, PARM *);

/*---------------------------------------------------------------------
* Missing Value definition copied from cscore.h
*--------------------------------------------------------------------*/

unsigned char SNaN[8] = {0xff,0xff,0xfe,0,0,0,0,0};

#define MISSINGNUM  (*((double *)SNaN))

#define MISSINGCHAR (" ")

#define amiss(a) (!(memcmp(&a, &(MISSINGNUM), sizeof(double))))

#define InSize 12
#define OutSize 4

/*---------------------------------------------------------------------
 * definitions copied from EM generated C source file
 *--------------------------------------------------------------------*/
#define csAMOUNT         indata[0].data.fnum
#define csAPPAREL        indata[1].data.fnum
#define csBLANKETS       indata[2].data.fnum
#define csDOMESTIC       indata[3].data.fnum
#define csFREQUENT       indata[4].data.fnum
#define csHOMEACC        indata[5].data.fnum
#define csLAMPS          indata[6].data.fnum
#define csLUXURY         indata[7].data.fnum
#define csOUTDOOR        indata[8].data.fnum
#define csRECENCY        indata[9].data.fnum
#define csSTATECOD       indata[10].data.str
#define csWCOAT          indata[11].data.fnum
#define csEM_PREDICTION  outdata[0].data.fnum
#define csG_STATECOD     outdata[1].data.fnum
#define csP_PURCHASE     outdata[2].data.fnum
#define cs_WARN_         outdata[3].data.str

int main(argc, argv)
```

```
     int argc;
     char *argv[];
{
   PARM* indata;
   PARM* outdata;

/* allocate and clear vectors */
   indata = (PARM *)malloc(sizeof(PARM)*InSize);
   outdata = (PARM *)malloc(sizeof(PARM)*OutSize);
   memset(outdata,0,sizeof(PARM)*OutSize);
   memset(indata,0,sizeof(PARM)*InSize);

/* allocate and clear character inputs */
   csSTATECOD   = (char *)malloc(34);      /* indata[10].data.str   */
   memset(csSTATECOD,0,sizeof(char)*34);


/* allocate and clear outputs */
   csEM_PREDICTION  = 0;                   /* outdata[0].data.fnum  */
   csG_STATECOD     = 0;                   /* outdata[1].data.fnum  */
   csP_PURCHASE     = 0;                   /* outdata[2].data.fnum  */
   cs_WARN_         = (char *)malloc(5);   /* outdata[3].data.str   */
   memset(cs_WARN_,0,sizeof(char)*5);

/* set inputs */
   csAMOUNT   = 740.0;                     /* indata[0].data.fnum   */
   csAPPAREL  = 1.0;                       /* indata[1].data.fnum   */
   csBLANKETS  =1.0;                       /* indata[2].data.fnum   */
   csDOMESTIC  = 4.0;                      /* indata[3].data.fnum   */
   csFREQUENT  = 1.23;                     /* indata[4].data.fnum   */
   csHOMEACC  = 1.0;                       /* indata[5].data.fnum   */
   csLAMPS   = 0;                          /* indata[6].data.fnum   */
   csLUXURY   = 0;                         /* indata[7].data.fnum   */
   csOUTDOOR   = 1.0;                      /* indata[8].data.fnum   */
   csRECENCY   = 0;                        /* indata[9].data.fnum   */
   strncpy(csSTATECOD,"MA",34);            /* indata[10].data.str   */
   csWCOAT   = 0;                          /* indata[11].data.fnum  */

/* invoke the EM generated C scoring function */
   score(indata,outdata);

/* handle output from the EM generated C scoring function */
   printf("\n>> First observation...\n");
   printf("csEM_PREDICTION = %f\n", csEM_PREDICTION);
   printf("csG_STATECOD = %f\n", csG_STATECOD);
   printf("csP_PURCHASE = %f\n", csP_PURCHASE);
   printf("cs_WARN_ = %s\n", cs_WARN_);

/* clear output vector */
   csEM_PREDICTION = 0;                    /* outdata[0].data.str   */
   csG_STATECOD    = 0;                    /* outdata[1].data.fnum  */
   csP_PURCHASE    = 0;                    /* outdata[2].data.fnum  */
   memset(cs_WARN_,0,sizeof(char)*5);      /* outdata[3].data.str   */

/* set inputs for second row */
   csAMOUNT   = 333.0;                     /* indata[0].data.fnum   */
   csAPPAREL  = 2.0;                       /* indata[1].data.fnum   */
```

```
   csBLANKETS  = 0.0;                      /* indata[2].data.fnum   */
   csDOMESTIC  = 1.0;                      /* indata[3].data.fnum   */
   csFREQUENT  = 3.62;                     /* indata[4].data.fnum   */
   csHOMEACC  = 9.0;                       /* indata[5].data.fnum   */
   csLAMPS   = 0;                          /* indata[6].data.fnum   */
   csLUXURY   = 0;                         /* indata[7].data.fnum   */
   csOUTDOOR   = 5.0;                      /* indata[8].data.fnum   */
   csRECENCY   = 4.0;                      /* indata[9].data.fnum   */
   strncpy(csSTATECOD,"MA",34);            /* indata[10].data.str   */
   csWCOAT   = 3;                          /* indata[11].data.fnum  */

/* invoke the EM generated C scoring function again*/
   score(indata,outdata);

/* handle output from the EM generated C scoring function */
   printf("\n>> Second observation...\n");
   printf("csEM_PREDICTION = %f\n", csEM_PREDICTION);
   printf("csG_STATECOD = %f\n", csG_STATECOD);
   printf("csP_PURCHASE = %f\n", csP_PURCHASE);
   printf("cs_WARN_ = %s\n", cs_WARN_);

/* clean up allocated memory */
   free(csSTATECOD);
   free(cs_WARN_);
   free(indata);
   free(outdata);

   return 0; /* end main */
}
```

Appendix C

Generated C Code

```
/*-------------------------------------------------------------
Copyright (C) 2000 SAS Institute, Inc. All rights reserved.

Notice:
  The following permissions are granted provided that the
  above copyright and this notice appear in the code and
  any related documentation. Permission to copy, modify
  and distribute the C language source code generated using
  or distributed with SAS Enterprise Miner C Scoring software
  and any executables derived from such source code is
  limited to customers of SAS Institute with a valid license
  for SAS Enterprise Miner C Scoring software. Any distribution
  of such executables or source code shall be on an "AS IS"
  basis without warranty of any kind.  SAS and all other SAS
  Institute. Inc. product and service names are registered
  trademarks or trademarks of SAS Institute Inc. in the USA
  and other countries. Except as contained in this notice,
  the name of the SAS Institute, SAS Enterprise Miner and
  SAS Enterprise Miner C Scoring software shall not be used in
  the advertising or promotion of products or services without
  prior written authorization from SAS Institute Inc.
-------------------------------------------------------------*/

/*--- start generated code ---*/

#include <math.h>
#include <string.h>
#include <memory.h>
#include <ctype.h>

#include "cscore.h"

#include "csparm.h"

#define csAMOUNT       indata[0].data.fnum
#define csAPPAREL       indata[1].data.fnum
#define csBLANKETS       indata[2].data.fnum
#define csDOMESTIC       indata[3].data.fnum
#define csFREQUENT       indata[4].data.fnum
#define csHOMEACC       indata[5].data.fnum
#define csLAMPS       indata[6].data.fnum
#define csLUXURY       indata[7].data.fnum
#define csOUTDOOR       indata[8].data.fnum
#define csRECENCY       indata[9].data.fnum
#define csSTATECOD       indata[10].data.str
#define csWCOAT       indata[11].data.fnum
#define csEM_PREDICTION       outdata[0].data.fnum
#define csG_STATECOD       outdata[1].data.fnum
#define csP_PURCHASE       outdata[2].data.fnum
#define cs_WARN_       outdata[3].data.str
```

```
SFDKeyWords void score ( PARM *indata, PARM *outdata )
{

/*--- Auto Variables ---*/
      double cs_0_0;
      double cs_0_1;
      double cs_0_2;
      double cs_0_3;
      double cs_0_4;
      double cs_0_5;
      double cs_0_6;
      double cs_0_7;
      double cs_0_8;
      char   cs_DM12[13];
      double cs_DM_BAD;
      double cs_DM_FIND;
      double cs_LP0;
      char   cs_NORM1[33];
      double cs_TEMP;

      strcpy(cs_WARN_,"    ");

   /*--- initialize stand alone formats library ---*/
   #if FMTLIB == 1
      xfinit();
   #endif

   /*-------------------------------------------------------------**/
   /* macro variable identifying the scored data set*/
   /*%let em_score_output=*/
   /*-------------------------------------------------------------**/
   /*-------------------------------------------------------------**/
   /* tool: input data source*/
   /* type: sample*/
   /* node: ids2*/
   /*-------------------------------------------------------------**/
   /*-------------------------------------------------------------**/
   /* tool: variable selection class*/
   /* type: explore*/
   /* node: varsel4*/
   /*-------------------------------------------------------------**/
   /*********************************************/
   /*** begin scoring code for variable selection*/
   /*********************************************/
    strncpyn( cs_NORM1, 32," ");
    dmnormcp(csSTATECOD,32, cs_NORM1);
   {
      if ( streq(cs_NORM1,"AE")) csG_STATECOD = 8;
      else if ( streq(cs_NORM1,"AK")) csG_STATECOD = 0;
      else if ( streq(cs_NORM1,"AL")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"AP")) csG_STATECOD = 1;
      else if ( streq(cs_NORM1,"AR")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"AZ")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"CA")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"CO")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"CT")) csG_STATECOD = 6;
```

```
      else if ( streq(cs_NORM1,"DC")) csG_STATECOD = 9;
      else if ( streq(cs_NORM1,"DE")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"FL")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"GA")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"HI")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"IA")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"ID")) csG_STATECOD = 2;
      else if ( streq(cs_NORM1,"IL")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"IN")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"KS")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"KY")) csG_STATECOD = 2;
      else if ( streq(cs_NORM1,"LA")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"MA")) csG_STATECOD = 8;
      else if ( streq(cs_NORM1,"MD")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"ME")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"MI")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"MN")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"MO")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"MS")) csG_STATECOD = 2;
      else if ( streq(cs_NORM1,"MT")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"NC")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"ND")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"NE")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"NH")) csG_STATECOD = 1;
      else if ( streq(cs_NORM1,"NJ")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"NM")) csG_STATECOD = 2;
      else if ( streq(cs_NORM1,"NV")) csG_STATECOD = 5;
      else if ( streq(cs_NORM1,"NY")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"OH")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"OK")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"OR")) csG_STATECOD = 7;
      else if ( streq(cs_NORM1,"PA")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"PR")) csG_STATECOD = 9;
      else if ( streq(cs_NORM1,"RI")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"SC")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"SD")) csG_STATECOD = 8;
      else if ( streq(cs_NORM1,"TN")) csG_STATECOD = 4;
      else if ( streq(cs_NORM1,"TX")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"UT")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"VA")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"VI")) csG_STATECOD = 9;
      else if ( streq(cs_NORM1,"VT")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"WA")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"WI")) csG_STATECOD = 3;
      else if ( streq(cs_NORM1,"WV")) csG_STATECOD = 6;
      else if ( streq(cs_NORM1,"WY")) csG_STATECOD = 5;
      else  strncpyn( cs_WARN_, 4,"U");
   }
/*---------------------------------------------------------------**/
   /* tool: regression*/
   /* type: model*/
   /* node: reg*/
   /*---------------------------------------------------------------**/
   /************************************/
   /** begin scoring code for regression*/
   /************************************/
   cs_DM_BAD=0;
```

```
/** check amount for missing values */
if ( missingn( csAMOUNT))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check apparel for missing values */
if ( missingn( csAPPAREL))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check blankets for missing values */
if ( missingn( csBLANKETS))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check domestic for missing values */
if ( missingn( csDOMESTIC))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check frequent for missing values */
if ( missingn( csFREQUENT))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check homeacc for missing values */
if ( missingn( csHOMEACC))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check lamps for missing values */
if ( missingn( csLAMPS))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check luxury for missing values */
if ( missingn( csLUXURY))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check outdoor for missing values */
if ( missingn( csOUTDOOR))
{
   lsubstr(cs_WARN_,1,1, "M");
   cs_DM_BAD = 1;
}
/** check recency for missing values */
if ( missingn( csRECENCY))
{
```

```
      lsubstr(cs_WARN_,1,1, "M");
      cs_DM_BAD = 1;
   }
   /** check wcoat for missing values */
   if ( missingn( csWCOAT))
   {
      lsubstr(cs_WARN_,1,1, "M");
      cs_DM_BAD = 1;
   }
   /** generate dummy variables for g_statecod */
/** encoding is sparse, initialize to zero*/
   cs_0_0 = 0;
   cs_0_1 = 0;
   cs_0_2 = 0;
   cs_0_3 = 0;
   cs_0_4 = 0;
   cs_0_5 = 0;
   cs_0_6 = 0;
   cs_0_7 = 0;
   cs_0_8 = 0;
   if ( missingn( csG_STATECOD))
   {
      cs_0_0 = MISSING;
      cs_0_1 = MISSING;
      cs_0_2 = MISSING;
      cs_0_3 = MISSING;
      cs_0_4 = MISSING;
      cs_0_5 = MISSING;
      cs_0_6 = MISSING;
      cs_0_7 = MISSING;
      cs_0_8 = MISSING;
      lsubstr(cs_WARN_,1,1, "M");
      cs_DM_BAD = 1;
   }
   else {
      nput( csG_STATECOD ,"BEST",12.,0., cs_DM12);
       dmnormip(cs_DM12,32);
      cs_DM_FIND = 0;
      if (strlteq( cs_DM12 ,"4"))
      {
         if (strlteq( cs_DM12 ,"2"))
         {
            if (strlteq( cs_DM12 ,"1"))
            {
               if (streq( cs_DM12,"0"))
               {
                  cs_0_0 = 1;
                  cs_DM_FIND = 1;
               }
               else {
                  if (streq( cs_DM12,"1"))
                  {
                     cs_0_1 = 1;
                     cs_DM_FIND = 1;
                  }
               }
            }
```

```
      else {
         if (streq( cs_DM12,"2"))
         {
            cs_0_2 = 1;
            cs_DM_FIND = 1;
         }
      }
   }
   else {
      if (streq( cs_DM12,"3"))
      {
         cs_0_3 = 1;
         cs_DM_FIND = 1;
      }
      else {
         if (streq( cs_DM12,"4"))
         {
            cs_0_4 = 1;
            cs_DM_FIND = 1;
         }
      }
   }
}
else {
   if (strlteq( cs_DM12 ,"7"))
   {
      if (strlteq( cs_DM12 ,"6"))
      {
         if (streq( cs_DM12,"5"))
         {
            cs_0_5 = 1;
            cs_DM_FIND = 1;
         }
         else {
            if (streq( cs_DM12,"6"))
            {
               cs_0_6 = 1;
               cs_DM_FIND = 1;
            }
         }
      }
      else {
         if (streq( cs_DM12,"7"))
         {
            cs_0_7 = 1;
            cs_DM_FIND = 1;
         }
      }
   }
   else {
      if (streq( cs_DM12,"8"))
      {
         cs_0_8 = 1;
         cs_DM_FIND = 1;
      }
      else {
         if (streq( cs_DM12,"9"))
```

```
            {
               cs_0_0 = -1;
               cs_0_1 = -1;
               cs_0_2 = -1;
               cs_0_3 = -1;
               cs_0_4 = -1;
               cs_0_5 = -1;
               cs_0_6 = -1;
               cs_0_7 = -1;
               cs_0_8 = -1;
               cs_DM_FIND = 1;
            }
         }
      }
   }
   if (! cs_DM_FIND)
   {
      cs_0_0 = MISSING;
      cs_0_1 = MISSING;
      cs_0_2 = MISSING;
      cs_0_3 = MISSING;
      cs_0_4 = MISSING;
      cs_0_5 = MISSING;
      cs_0_6 = MISSING;
      cs_0_7 = MISSING;
      cs_0_8 = MISSING;
      lsubstr(cs_WARN_,2,1, "U");
      cs_DM_BAD = 1;
   }
}
/** if missing inputs, use averages*/
if ( cs_DM_BAD > 0)
{
   cs_LP0 = 0.50813835198372;
    goto REGDR1;
}
/** compute linear predictor*/
cs_LP0 = 0;
/**  effect: amount */
cs_TEMP = csAMOUNT;
cs_LP0 = cs_LP0 + ( -0.00004993870147 * cs_TEMP);
/**  effect: apparel */
cs_TEMP = csAPPAREL;
cs_LP0 = cs_LP0 + ( 0.01297149165591 * cs_TEMP);
/**  effect: blankets */
cs_TEMP = csBLANKETS;
cs_LP0 = cs_LP0 + ( 0.01307320528946 * cs_TEMP);
/**  effect: domestic */
cs_TEMP = csDOMESTIC;
cs_LP0 = cs_LP0 + ( 0.00908957848067 * cs_TEMP);
/**  effect: frequent */
cs_TEMP = csFREQUENT;
cs_LP0 = cs_LP0 + ( 0.02265229436888 * cs_TEMP);
/**  effect: g_statecod */
cs_TEMP = 1;
cs_LP0 = cs_LP0 + ( -0.38058915743906) * cs_TEMP * cs_0_0;
cs_LP0 = cs_LP0 + ( -0.19842465513138) * cs_TEMP * cs_0_1;
```

```
   cs_LP0 = cs_LP0 + ( -0.09040404754398) * cs_TEMP * cs_0_2;
   cs_LP0 = cs_LP0 + ( -0.06565678604598) * cs_TEMP * cs_0_3;
   cs_LP0 = cs_LP0 + ( -0.02912411263384) * cs_TEMP * cs_0_4;
   cs_LP0 = cs_LP0 + ( 0.00128070484317) * cs_TEMP * cs_0_5;
   cs_LP0 = cs_LP0 + ( 0.00897606795372) * cs_TEMP * cs_0_6;
   cs_LP0 = cs_LP0 + ( 0.09650736361157) * cs_TEMP * cs_0_7;
   cs_LP0 = cs_LP0 + ( 0.20808691085082) * cs_TEMP * cs_0_8;
   /**  effect: homeacc */
   cs_TEMP = csHOMEACC;
   cs_LP0 = cs_LP0 + ( -0.00550672193636 * cs_TEMP);
   /**  effect: lamps */
   cs_TEMP = csLAMPS;
   cs_LP0 = cs_LP0 + ( 0.01856257780278 * cs_TEMP);
   /**  effect: luxury */
   cs_TEMP = csLUXURY;
   cs_LP0 = cs_LP0 + ( 0.04536833034207 * cs_TEMP);
   /**  effect: outdoor */
   cs_TEMP = csOUTDOOR;
   cs_LP0 = cs_LP0 + ( 0.00806192635828 * cs_TEMP);
   /**  effect: recency */
   cs_TEMP = csRECENCY;
   cs_LP0 = cs_LP0 + ( -0.00020781136645 * cs_TEMP);
   /**  effect: wcoat */
   cs_TEMP = csWCOAT;
   cs_LP0 = cs_LP0 + ( 0.01930625999309 * cs_TEMP);
   /*--- intercept ---**/
   cs_LP0 = cs_LP0 + ( 0.43801900950123);
   REGDR1: ;
   /** predicted value*/
   csP_PURCHASE = cs_LP0;
   /************************************/
   /**** end scoring code for regression*/
   /************************************/
   /*-------------------------------------------------------------**/
   /* tool: score node*/
   /* type: assess*/
   /* node: score*/
   /*-------------------------------------------------------------**/
   /*-------------------------------------------------------------**/
   /* score: creating fixed names*/
   /*-------------------------------------------------------------**/
   csEM_PREDICTION = csP_PURCHASE;



  return;

}
usrNbucket usrnBuckets[]={
/* not used */
{0,0,0,NULL,NULL,' ',0.0}
};

usrCbucket usrcBuckets[]={
/* not used */
{NULL, NULL, 0, NULL}
```

```
};
usrFormat usrnFormats[] = {
{"**unused**",0.0,0,0,0,-1,-1,0}
};
usrFormat usrcFormats[] = {
{"**unused**",0.0,0,0,0,-1,-1,0}
};
int usrNumNfmts = 0;
int usrNumCfmts = 0;
usrFormat usrcFormats[] = {
{"**unused**",0.0,0,0,0,-1,-1,0}
};
int usrNumNfmts = 0;
int usrNumCfmts = 0;usrFormat usrnFormats[] = {
{"**unused**",0.0,0,0,0,-1,-1,0}
};
usrFormat usrcFormats[] = {
{"**unused**",0.0,0,0,0,-1,-1,0}
};
int usrNumNfmts = 0;
int usrNumCfmts = 0;
```

Appendix D

Cscore XML

```
<?xml version="1.0" encoding="utf-8"?>
<Score>
   <Producer>
      <Name> SAS Enterprise Miner </Name>
      <Version> 1.0 </Version>
   </Producer>
   <TargetList>
   </TargetList>
   <Input>
      <Variable>
         <Name> AMOUNT </Name>
         <Type> numeric </Type>
         <Description>
            Dollars Spent
         </Description>
      </Variable>
      <Variable>
         <Name> APPAREL </Name>
         <Type> numeric </Type>
         <Description>
            Apparel Purch.
         </Description>
      </Variable>
      <Variable>
         <Name> BLANKETS </Name>
         <Type> numeric </Type>
         <Description>
            Blankets Purch.
         </Description>
      </Variable>
      <Variable>
         <Name> DOMESTIC </Name>
         <Type> numeric </Type>
         <Description>
            Domestic Prod.
         </Description>
      </Variable>
      <Variable>
         <Name> FREQUENT </Name>
         <Type> numeric </Type>
         <Description>
            Order Frequency
         </Description>
      </Variable>
      <Variable>
         <Name> HOMEACC </Name>
         <Type> numeric </Type>
         <Description>
            Home Furniture
         </Description>
      </Variable>
```

```
   <Variable>
      <Name> LAMPS </Name>
      <Type> numeric </Type>
      <Description>
         Lamps Purch.
      </Description>
   </Variable>
   <Variable>
      <Name> LUXURY </Name>
      <Type> numeric </Type>
      <Description>
         Luxury Items
      </Description>
   </Variable>
   <Variable>
      <Name> OUTDOOR </Name>
      <Type> numeric </Type>
      <Description>
         Outdoor Prod.
      </Description>
   </Variable>
   <Variable>
      <Name> RECENCY </Name>
      <Type> numeric </Type>
      <Description>
         Recency
      </Description>
   </Variable>
   <Variable>
      <Name> STATECOD </Name>
      <Type> character </Type>
      <Description>
         State Code
      </Description>
   </Variable>
   <Variable>
      <Name> WCOAT </Name>
      <Type> numeric </Type>
      <Description>
         Ladies Coats
      </Description>
   </Variable>
</Input>
<Output>
   <Variable>
      <Name> EM_CLASSIFICATION </Name>
      <Type> character </Type>
      <Description>
         Prediction for PURCHASE
      </Description>
   </Variable>
   <Variable>
      <Name> EM_EVENTPROBABILITY </Name>
      <Type> numeric </Type>
      <Description>
         Probability for level YES of PURCHASE
      </Description>
```

```
   </Variable>
   <Variable>
      <Name> EM_PROBABILITY </Name>
      <Type> numeric </Type>
      <Description>
         Probability of Classification
      </Description>
   </Variable>
   <Variable>
      <Name> G_STATECOD </Name>
      <Type> numeric </Type>
      <Description>
         Group: STATECOD
      </Description>
   </Variable>
   <Variable>
      <Name> I_PURCHASE </Name>
      <Type> character </Type>
      <Description>
         Into: PURCHASE
      </Description>
   </Variable>
   <Variable>
      <Name> P_PURCHASENO </Name>
      <Type> numeric </Type>
      <Description>
         Predicted: PURCHASE=No
      </Description>
   </Variable>
   <Variable>
      <Name> P_PURCHASEYES </Name>
      <Type> numeric </Type>
      <Description>
         Predicted: PURCHASE=Yes
      </Description>
   </Variable>
   <Variable>
      <Name> U_PURCHASE </Name>
      <Type> numeric </Type>
      <Description>
         Unnormalized Into: PURCHASE
      </Description>
   </Variable>
   <Variable>
      <Name> _WARN_ </Name>
      <Type> character </Type>
      <Description>
         Warnings
      </Description>
   </Variable>
</Output>
<C>
   <Function>
      <Name>
         score
      </Name>
      <ParameterList>
```

```
<Parameter>
   <Array length="12">
      <Type>
         Parm
      </Type>
      <DataMap>
         <Element index="0">
            <Value>
               <Origin> AMOUNT </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="1">
            <Value>
               <Origin> APPAREL </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="2">
            <Value>
               <Origin> BLANKETS </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="3">
            <Value>
               <Origin> DOMESTIC </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="4">
            <Value>
               <Origin> FREQUENT </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="5">
            <Value>
               <Origin> HOMEACC </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="6">
            <Value>
               <Origin> LAMPS </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="7">
            <Value>
               <Origin> LUXURY </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="8">
            <Value>
               <Origin> OUTDOOR </Origin>
```

```
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="9">
            <Value>
               <Origin> RECENCY </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="10">
            <Value>
               <Origin> STATECOD </Origin>
               <Array length="34">
                 <Type> char </Type>
              </Array>
            </Value>
         </Element>
         <Element index="11">
            <Value>
               <Origin> WCOAT </Origin>
               <Type> double </Type>
            </Value>
         </Element>
      </DataMap>
   </Array>
</Parameter>

<Parameter>
   <Array length="9">
      <Type>
         Parm
      </Type>
      <DataMap>
         <Element index="0">
            <Value>
               <Origin> EM_CLASSIFICATION </Origin>
               <Array length="33">
                 <Type> char </Type>
              </Array>
            </Value>
         </Element>
         <Element index="1">
            <Value>
               <Origin> EM_EVENTPROBABILITY </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="2">
            <Value>
               <Origin> EM_PROBABILITY </Origin>
               <Type> double </Type>
            </Value>
         </Element>
         <Element index="3">
            <Value>
               <Origin> G_STATECOD </Origin>
               <Type> double </Type>
```

```
                        </Value>
                     </Element>
                     <Element index="4">
                        <Value>
                           <Origin> I_PURCHASE </Origin>
                           <Array length="4">
                             <Type> char </Type>
                           </Array>
                        </Value>
                     </Element>
                     <Element index="5">
                        <Value>
                           <Origin> P_PURCHASENO </Origin>
                           <Type> double </Type>
                        </Value>
                     </Element>
                     <Element index="6">
                        <Value>
                           <Origin> P_PURCHASEYES </Origin>
                           <Type> double </Type>
                        </Value>
                     </Element>
                     <Element index="7">
                        <Value>
                           <Origin> U_PURCHASE </Origin>
                           <Type> double </Type>
                        </Value>
                     </Element>
                     <Element index="8">
                        <Value>
                           <Origin> _WARN_ </Origin>
                           <Array length="5">
                             <Type> char </Type>
                           </Array>
                        </Value>
                     </Element>
                  </DataMap>
               </Array>
            </Parameter>
         </ParameterList>
      </Function>
   </C>
</Score>
```

Appendix E

Example Java Main Program

```
import eminer.user.Score2.*;
import com.sas.analytics.eminer.jscore.util.*;
import java.util.Map;
import java.util.HashMap;

public class Jsbasic {
  public static void main(String[] args) {
    Map outdata;

    Map indata = new HashMap(12);
    Jscore jsb = new Score();

    // load data into input Map
    indata.put("AMOUNT",((Object)new Double(740)));
    indata.put("APPAREL",((Object)new Double(1)));
    indata.put("BLANKETS",((Object)new Double(1)));
    indata.put("DOMESTIC",((Object)new Double(4)));
    indata.put("FREQUENT",((Object)new Double(1.23)));
    indata.put("HOMEACC",((Object)new Double(1)));
    indata.put("LAMPS",((Object)new Double(0)));
    indata.put("LUXURY",((Object)new Double(0)));
    indata.put("OUTDOOR",((Object)new Double(1)));
    indata.put("RECENCY",((Object)new Double(0)));
    indata.put("STATECOD",((Object)"MA"));
    indata.put("WCOAT",((Object)new Double(0)));

    try {
      //invoke the scoring method
      outdata = jsb.score(indata);

      // process scoring output
      System.out.println(">> First observation...");

      System.out.println("EM_CLASSIFICATION = " + (String)outdata.get("EM_CLASSIFICATION"));
      System.out.println("EM_EVENTPROBABILITY = " +
(Double)outdata.get("EM_EVENTPROBABILITY"));
      System.out.println("EM_PROBABILITY = " + (Double)outdata.get("EM_PROBABILITY"));
      System.out.println("G_STATECOD = " + (Double)outdata.get("G_STATECOD"));
      System.out.println("I_PURCHASE= " + (String)outdata.get("I_PURCHASE"));
      System.out.println("P_PURCHASENO = " + (Double)outdata.get("P_PURCHASENO"));
      System.out.println("P_PURCHASEYES = " + (Double)outdata.get("P_PURCHASEYES"));
      System.out.println("U_PURCHASE = " + (Double)outdata.get("U_PURCHASE"));
      System.out.println("_WARN_ = " + (String)outdata.get("_WARN_"));

    }  catch (Exception ex) {
                System.out.println("Exception caught....Scoring failed");
                return;
    }
```

```
    // load obs2 data into input Map
    indata.put("AMOUNT",((Object)new Double(333)));
    indata.put("APPAREL",((Object)new Double(2)));
    indata.put("BLANKETS",((Object)new Double(0)));
    indata.put("DOMESTIC",((Object)new Double(1)));
    indata.put("FREQUENT",((Object)new Double(3.62)));
    indata.put("HOMEACC",((Object)new Double(9)));
    indata.put("LAMPS",((Object)new Double(0)));
    indata.put("LUXURY",((Object)new Double(0)));
    indata.put("OUTDOOR",((Object)new Double(5)));
    indata.put("RECENCY",((Object)new Double(4)));
    indata.put("STATECOD",((Object)"MA"));
    indata.put("WCOAT",((Object)new Double(3)));

    try {
      //invoke the scoring method
      outdata = jsb.score(indata);

      // process scoring output

      System.out.println("\n>> Second observation...");

      System.out.println("EM_CLASSIFICATION = " + (String)outdata.get("EM_CLASSIFICATION"));
      System.out.println("EM_EVENTPROBABILITY = " +
(Double)outdata.get("EM_EVENTPROBABILITY"));
      System.out.println("EM_PROBABILITY = " + (Double)outdata.get("EM_PROBABILITY"));
      System.out.println("G_STATECOD = " + (Double)outdata.get("G_STATECOD"));
      System.out.println("I_PURCHASE= " + (String)outdata.get("I_PURCHASE"));
      System.out.println("P_PURCHASENO = " + (Double)outdata.get("P_PURCHASENO"));
      System.out.println("P_PURCHASEYES = " + (Double)outdata.get("P_PURCHASEYES"));
      System.out.println("U_PURCHASE = " + (Double)outdata.get("U_PURCHASE"));
      System.out.println("_WARN_ = " + (String)outdata.get("_WARN_"));

    }  catch (Exception ex) {
                System.out.println("Exception caught....Scoring failed");
                return;
    }

  } // end main
} //end class Try
```

Appendix F

Generated Score Class

```
package eminer.user.Score2;
import java.util.Map;
import java.util.HashMap;
import com.sas.ds.*;
import com.sas.analytics.eminer.jscore.util.*;


/**
* The Score class encapsulates data step scoring code translated
* by the SAS Enterprise Miner Java Scoring Component.
*
* @since 1.0
* @version J*Score 1.0
* @author SAS Enterprise Miner Java Scoring Component
* @see com.sas.analytics.eminer.jscore.util.Jscore
*/

public class Score
      implements Jscore {
  private boolean dataModified;
  public boolean reuseOutputMap;
  private DS dscode;

/**
* A map of the (key) name, (value) reference pair for every
* variable modified by the score method.
* This is provided primarily for optional advanced development.
*/
  protected Map outputVariables;

  public Score() {
    DSFormats formatLib = new JscoreUserFormats();
    dataModified = false;
    reuseOutputMap = false;
    dscode = new DS(formatLib);
  }


/**
* Executes the scoring code and returns an output Map.
* By default a new instance of the output map is allocated
* the method is invoked. To modify this behavior set the public
* field reuseOutputMap to true. This will cause only one
* output map to be allocated, subsequent calls will reuse the
* same map. Note this will cause the content of the map to be
* over written at each time this method is invoked.
* @param a reference to a Map object that contains the
* (key) name String, (value) pair, for the
* input variables to be used in the "scoring" code.
* @return a Map of the (key) name, (value) reference pair for all
```

```
* variables modified in the "scoring" code.
* @throws com.sas.analytics.eminer.jscore.util.JscoreException Invalid input data type for String
\"variableName\".
* @throws com.sas.analytics.eminer.jscore.util.JscoreException Invalid input data type for Double
\"variableName\".
*/
  public Map score ( Map indata) throws JscoreException {
    Object tmpVar;

    dscode.initialize();
    if ( reuseOutputMap == false || outputVariables == null)
        outputVariables = new HashMap( 28, .75F );

    tmpVar = indata.get("AMOUNT");
    if( tmpVar != null) {
      try {
         dscode.AMOUNT = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"AMOUNT\".");
      }
    }
    else
      dscode.AMOUNT = Double.NaN;

    tmpVar = indata.get("APPAREL");
    if( tmpVar != null) {
      try {
         dscode.APPAREL = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"APPAREL\".");
      }
    }
    else
      dscode.APPAREL = Double.NaN;

    tmpVar = indata.get("BLANKETS");
    if( tmpVar != null) {
      try {
         dscode.BLANKETS = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"BLANKETS\".");
      }
    }
    else
      dscode.BLANKETS = Double.NaN;

    tmpVar = indata.get("DOMESTIC");
    if( tmpVar != null) {
      try {
         dscode.DOMESTIC = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"DOMESTIC\".");
      }
    }
    else
      dscode.DOMESTIC = Double.NaN;
```

```
tmpVar = indata.get("FREQUENT");
if( tmpVar != null) {
  try {
     dscode.FREQUENT = ((Double)tmpVar).doubleValue();
  } catch (Exception ex) {
     throw new JscoreException("Invalid input data type for Double \"FREQUENT\".");
  }
}
else
  dscode.FREQUENT = Double.NaN;

tmpVar = indata.get("HOMEACC");
if( tmpVar != null) {
  try {
     dscode.HOMEACC = ((Double)tmpVar).doubleValue();
  } catch (Exception ex) {
     throw new JscoreException("Invalid input data type for Double \"HOMEACC\".");
  }
}
else
  dscode.HOMEACC = Double.NaN;

tmpVar = indata.get("LAMPS");
if( tmpVar != null) {
  try {
     dscode.LAMPS = ((Double)tmpVar).doubleValue();
  } catch (Exception ex) {
     throw new JscoreException("Invalid input data type for Double \"LAMPS\".");
  }
}
else
  dscode.LAMPS = Double.NaN;

tmpVar = indata.get("LUXURY");
if( tmpVar != null) {
  try {
     dscode.LUXURY = ((Double)tmpVar).doubleValue();
  } catch (Exception ex) {
     throw new JscoreException("Invalid input data type for Double \"LUXURY\".");
  }
}
else
  dscode.LUXURY = Double.NaN;

tmpVar = indata.get("OUTDOOR");
if( tmpVar != null) {
  try {
     dscode.OUTDOOR = ((Double)tmpVar).doubleValue();
  } catch (Exception ex) {
     throw new JscoreException("Invalid input data type for Double \"OUTDOOR\".");
  }
}
else
  dscode.OUTDOOR = Double.NaN;
```

```
    tmpVar = indata.get("RECENCY");
    if( tmpVar != null) {
      try {
         dscode.RECENCY = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"RECENCY\".");
      }
    }
    else
      dscode.RECENCY = Double.NaN;

    tmpVar = indata.get("STATECOD");
    if( tmpVar != null) {
      try {
        dscode.STATECOD = (String)tmpVar;
      } catch (Exception ex) {
        throw new JscoreException("Invalid input data type for String \"STATECOD\".");
      }
    }
    else
      dscode.STATECOD = " ";

    tmpVar = indata.get("WCOAT");
    if( tmpVar != null) {
      try {
         dscode.WCOAT = ((Double)tmpVar).doubleValue();
      } catch (Exception ex) {
         throw new JscoreException("Invalid input data type for Double \"WCOAT\".");
      }
    }
    else
      dscode.WCOAT = Double.NaN;

    tmpVar = indata.get("_FORMAT");
    if( tmpVar != null) {
      try {
        dscode._FORMAT = (String)tmpVar;
      } catch (Exception ex) {
        throw new JscoreException("Invalid input data type for String \"_FORMAT\".");
      }
    }
    else
      dscode._FORMAT = " ";


    dscode.run();

    outputVariables.put("EM_CLASSIFICATION",((Object)dscode.EM_CLASSIFICATION));
    outputVariables.put("EM_EVENTPROBABILITY",((Object)new
Double(dscode.EM_EVENTPROBABILITY)));
    outputVariables.put("EM_PROBABILITY",((Object)new Double(dscode.EM_PROBABILITY)));
    outputVariables.put("G_STATECOD",((Object)new Double(dscode.G_STATECOD)));
    outputVariables.put("I_PURCHASE",((Object)dscode.I_PURCHASE));
    outputVariables.put("P_PURCHASENO",((Object)new Double(dscode.P_PURCHASENO)));
    outputVariables.put("P_PURCHASEYES",((Object)new Double(dscode.P_PURCHASEYES)));
    outputVariables.put("U_PURCHASE",((Object)new Double(dscode.U_PURCHASE)));
```

```
    outputVariables.put("_WARN_",((Object)dscode._WARN_));
    return outputVariables;
  }

} // end class Score
```

Generated User Written Formats Class

```
package eminer.user.Score2;
import com.sas.text.*;
import com.sas.util.*;
import com.sas.ds.*;

public class JscoreUserFormats extends DSFormats
{
 static
 {
      ValueFormatDescription entry;
      PictureFormatDescription pentry;

 }
}
```

Appendix G

Jscore XML

```
<?xml version="1.0" encoding="utf-8"?>
<Score>
  <Producer>
    <Name> SAS Enterprise Miner </Name>
    <Version> 1.0 </Version>
  </Producer>
  <TargetList>
  </TargetList>
  <Input>
    <Variable>
      <Name> AMOUNT </Name>
      <Type> numeric </Type>
      <Description>
        Dollars Spent
      </Description>
    </Variable>
    <Variable>
      <Name> APPAREL </Name>
      <Type> numeric </Type>
      <Description>
        Apparel Purch.
      </Description>
    </Variable>
    <Variable>
      <Name> BLANKETS </Name>
      <Type> numeric </Type>
      <Description>
        Blankets Purch.
      </Description>
    </Variable>
    <Variable>
      <Name> DOMESTIC </Name>
      <Type> numeric </Type>
      <Description>
        Domestic Prod.
      </Description>
    </Variable>
    <Variable>
      <Name> FREQUENT </Name>
      <Type> numeric </Type>
      <Description>
        Order Frequency
      </Description>
    </Variable>
    <Variable>
      <Name> HOMEACC </Name>
      <Type> numeric </Type>
      <Description>
        Home Furniture
      </Description>
```

```
  </Variable>
  <Variable>
    <Name> LAMPS </Name>
    <Type> numeric </Type>
    <Description>
      Lamps Purch.
    </Description>
  </Variable>
  <Variable>
    <Name> LUXURY </Name>
    <Type> numeric </Type>
    <Description>
      Luxury Items
    </Description>
  </Variable>
  <Variable>
    <Name> OUTDOOR </Name>
    <Type> numeric </Type>
    <Description>
      Outdoor Prod.
    </Description>
  </Variable>
  <Variable>
    <Name> RECENCY </Name>
    <Type> numeric </Type>
    <Description>
      Recency
    </Description>
  </Variable>
  <Variable>
    <Name> STATECOD </Name>
    <Type> character </Type>
    <Description>
      State Code
    </Description>
  </Variable>
  <Variable>
    <Name> WCOAT </Name>
    <Type> numeric </Type>
    <Description>
      Ladies Coats
    </Description>
  </Variable>
</Input>
<Output>
  <Variable>
    <Name> EM_CLASSIFICATION </Name>
    <Type> character </Type>
    <Description>
      Prediction for PURCHASE
    </Description>
  </Variable>
  <Variable>
    <Name> EM_EVENTPROBABILITY </Name>
    <Type> numeric </Type>
    <Description>
```

```
      Probability for level YES of PURCHASE
    </Description>
  </Variable>
  <Variable>
    <Name> EM_PROBABILITY </Name>
    <Type> numeric </Type>
    <Description>
      Probability of Classification
    </Description>
  </Variable>
  <Variable>
    <Name> G_STATECOD </Name>
    <Type> numeric </Type>
    <Description>
      Group: STATECOD
    </Description>
  </Variable>
  <Variable>
    <Name> I_PURCHASE </Name>
    <Type> character </Type>
    <Description>
      Into: PURCHASE
    </Description>
  </Variable>
  <Variable>
    <Name> P_PURCHASENO </Name>
    <Type> numeric </Type>
    <Description>
      Predicted: PURCHASE=No
    </Description>
  </Variable>
  <Variable>
    <Name> P_PURCHASEYES </Name>
    <Type> numeric </Type>
    <Description>
      Predicted: PURCHASE=Yes
    </Description>
  </Variable>
  <Variable>
    <Name> U_PURCHASE </Name>
    <Type> numeric </Type>
    <Description>
      Unnormalized Into: PURCHASE
    </Description>
  </Variable>
  <Variable>
    <Name> _WARN_ </Name>
    <Type> character </Type>
    <Description>
      WarningsWarnings
    </Description>
  </Variable>
</Output>

<Java>
  <Class>
```

```
      <Name>
        eminer.user.Score2.Score
      </Name>
      <Constructor>
        <Name>
          Score
        </Name>
      </Constructor>
      <Implements>
        <Interface>
          <Name>
            com.sas.analytics.eminer.jscore.util.Jscore
          </Name>

          <Method>
            <Name> score </Name>
            <Parameter> java.util.Map </Parameter>
            <Returns> java.util.Map </Returns>
            <Throws>
              com.sas.analytics.eminer.jscore.util.JscoreException
            </Throws>
          </Method>
        </Interface>
      </Implements>
    </Class>
  </Java>
</Score>
```

Appendix H

General Code Limitations

The SAS data step language represents a flexible and powerful development environment. The Enterprise Miner component that generates scoring code in C and Java supports only a small fraction of the syntax, expressions and functions supported by the SAS System. Every effort has been made to insure that the data step code generated by Enterprise Miner fits into the restrictions of the C and Java generation process. It is however quite possible to create, within Enterprise Miner, code that can not be correctly generated as C or Java. This is particularly a problem with transformations created within Enterprise Miner. When creating transformations within the Enterprise Miner Expression Builder, if the ultimate goal is to use the C or Java scoring code, great care must be taken to make sure the transformations are as C or Java like as possible to facilitate the generation of correct code. In general try to structure the data step code for any transformation to be as C like as possible. Any SAS operand or function not native to the C or Java languages should be avoided unless explicitly supported by the C and Java generation process.

Supported Functions

The following SAS System functions are supported either directly by the target language libraries or by code distributed with Enterprise Miner:

```
ARCOS(n);
ARSIN(n);
ATAN(n);
CEIL(n);
COS(n);
COSH(n);
c1= DMNORMCP(c1,n1,c2);
c1 = DMNORMIP(c1,n1);
n2 = DMRAN(n1); Similar to the SAS system function RANUNI
n2 = EXP(n1);
n2 = FLOOR(n1);
INDEX(c1, c2);
INT(n1);
c1= LEFT(c1);
n1 = LENGTH(c1);
n2 = LOG(n1);
n2 = LOG10(n1);
nx = MAX(n1, n2, n3, …);
nx = MIN(n1, n2, n3, …);
n1 = MISSING(<n1/c1>)
nx = N(n1, n2, n3, …);
nx = NMISS(n1, n2, n3, …);
n2 = PROBNORM(n1);
PUT((<n1/c1>,fmtw.d);
n2 = SIN(n);
n2 = SINH(n);
n2 = SQRT(n);
c2 = STRIP(c1);
c2 = SUBSTR(c1, p, n1);   /* n is not optional  */
SUBSTR(c1,p,n1) = strx;
```

```
n2 = TAN(n1);
n2 = TANH(n1);
c1 = TRIM(c1);
c1 = UPCASE(c1);
```

Where ‘n’ variables are numeric type and ‘c’ variables are character type

Supported SAS operators

Arithmetic Operators

SAS System Symbol	C *Score Equivalent	Definition
+	+	addition
-	-	subtraction
*	*	multiplication
**	pow();	exponentiation
/	/	division

Comparison Operators

SAS System Symbol	SAS System Mnemonic	C Equivalent	Definition
=	EQ	==	equal to
^=	NE	!=	not equal to
>	GT	>	greater than
<	LT	<	less than
>=	GE	>=	greater than or equal to
<=	LE	<=	less than or equal to
	IN	IN();	equal to one of a list

Logical Operators

SAS System Symbol	SAS System Mnemonic	C Equivalent
&	AND	&&
|	OR	||
^	NOT	!

Other operators

In SAS, the concatenation operator (||) concatenates character values. Enterprise Miner only supports concatenation of constants (quoted strings) in C or Java score code.

Conditional Statement Syntax

In any conditional statement to be represented in C or Java code, any variable that could possibly have a missing value must be subjected to an exclusionary test for missing before any other operation.

Any comparison of a character type variable and a quoted character constant, the quoted character constant must be the second operand.

Variable Name Length

Enterprise Miner truncates column names at 32 bytes. So, care must be exercised in the ETL process for both the training data and scoring data to insure that all column names are at least unique in the first 32 characters.

Character Data Length

In Enterprise Miner the maximum length of character data is 32 bytes. So, care must be exercised in the ETL process for both the training data and scoring data to insure that all character data is unique in the first 32 characters.

Extended Character Sets

The generation of C and Java score code for extended character set data and variable names is not supported. The generation of C and Java score code depends upon single byte length of characters so, multi byte character names and data can not be supported. The generated Java code that contains single byte, extended character set names and data is untested and unsupported. Since the generated C code depends upon the char type, character values of both variable names and data values are limited to integral values with a minimum value of -127 and a maximum value of 127.

Appendix I

SAS System Formats Supported for Java

$	CRODFDWN	DESDFDDS	ESPDFDDD	FRADFDT
$ASCII	CRODFMN	DESDFDE	ESPDFDDP	FRADFDWN
$BINARY	CRODFMY	DESDFDN	ESPDFDDS	FRADFMN
$CHAR	CRODFWDX	DESDFDT	ESPDFDE	FRADFMY
$F	CRODFWKX	DESDFDWN	ESPDFDN	FRADFWDX
$HEX	CSYDFDD	DESDFMN	ESPDFDT	FRADFWKX
$OCTAL	CSYDFDDB	DESDFMY	ESPDFDWN	FRSDFDD
AFRDFDD	CSYDFDDC	DESDFWDX	ESPDFMN	FRSDFDDB
AFRDFDDB	CSYDFDDD	DESDFWKX	ESPDFMY	FRSDFDDC
AFRDFDDC	CSYDFDDP	DEUDFDD	ESPDFWDX	FRSDFDDD
AFRDFDDD	CSYDFDDS	DEUDFDDB	ESPDFWKX	FRSDFDDP
AFRDFDDP	CSYDFDE	DEUDFDDC	EURDFDD	FRSDFDDS
AFRDFDDS	CSYDFDN	DEUDFDDD	EURDFDDB	FRSDFDE
AFRDFDE	CSYDFDT	DEUDFDDP	EURDFDDC	FRSDFDN
AFRDFDN	CSYDFDWN	DEUDFDDS	EURDFDDD	FRSDFDT
AFRDFDT	CSYDFMN	DEUDFDE	EURDFDDP	FRSDFDWN
AFRDFDWN	CSYDFMY	DEUDFDN	EURDFDDS	FRSDFMN
AFRDFMN	CSYDFWDX	DEUDFDT	EURDFDE	FRSDFMY
AFRDFMY	CSYDFWKX	DEUDFDWN	EURDFDN	FRSDFWDX
AFRDFWDX	DANDFDD	DEUDFMN	EURDFDT	FRSDFWKX
AFRDFWKX	DANDFDDB	DEUDFMY	EURDFDWN	HEX
BEST	DANDFDDC	DEUDFWDX	EURDFMN	HHMM
BINARY	DANDFDDD	DEUDFWKX	EURDFMY	HOUR
CATDFDD	DANDFDDP	DOLLAR	EURDFWDX	HUNDFDD
CATDFDDB	DANDFDDS	DOLLARX	EURDFWKX	HUNDFDDB
CATDFDDC	DANDFDE	DOWNAME	EURO	HUNDFDDC
CATDFDDD	DANDFDN	DTDATE	F	HUNDFDDD
CATDFDDP	DANDFDT	DTMONYY	FINDFDD	HUNDFDDP
CATDFDDS	DANDFDWN	DTWKDATX	FINDFDDB	HUNDFDDS
CATDFDE	DANDFMN	DTYEAR	FINDFDDC	HUNDFDE
CATDFDN	DANDFMY	DTYYQC	FINDFDDD	HUNDFDN
CATDFDT	DANDFWDX	E	FINDFDDP	HUNDFDT
CATDFDWN	DANDFWKX	ENGDFDD	FINDFDDS	HUNDFDWN
CATDFMN	DATE	ENGDFDDB	FINDFDE	HUNDFMN
CATDFMY	DATEAMPM	ENGDFDDC	FINDFDN	HUNDFMY
CATDFWDX	DATETIME	ENGDFDDD	FINDFDT	HUNDFWDX
CATDFWKX	DAY	ENGDFDDP	FINDFDWN	HUNDFWKX
COMMA	DDMMYY	ENGDFDDS	FINDFMN	ITADFDD
COMMAX	DDMMYYB	ENGDFDE	FINDFMY	ITADFDDB
COMMAX	DDMMYYC	ENGDFDN	FINDFWDX	ITADFDDC
CRODFDD	DDMMYYD	ENGDFDT	FINDFWKX	ITADFDDD
CRODFDDB	DDMMYYN	ENGDFDWN	FRADFDD	ITADFDDP
CRODFDDC	DDMMYYP	ENGDFMN	FRADFDDB	ITADFDDS
CRODFDDD	DDMMYYS	ENGDFMY	FRADFDDC	ITADFDE
CRODFDDP	DESDFDD	ENGDFWDX	FRADFDDD	ITADFDN
CRODFDDS	DESDFDDB	ENGDFWKX	FRADFDDP	ITADFDT
CRODFDE	DESDFDDC	ESPDFDD	FRADFDDS	ITADFDWN
CRODFDN	DESDFDDD	ESPDFDDB	FRADFDE	ITADFMN
CRODFDT	DESDFDDP	ESPDFDDC	FRADFDN	ITADFMY

ITADFWDX
ITADFWKX
JULDATE
JULDAY
JULIAN
LOGPROB
MACDFDD
MACDFDDB
MACDFDDC
MACDFDDD
MACDFDDP
MACDFDDS
MACDFDE
MACDFDN
MACDFDT
MACDFDWN
MACDFMN
MACDFMY
MACDFWDX
MACDFWKX
MMDDYY
MMDDYYB
MMDDYYC
MMDDYYD
MMDDYYN
MMDDYYP
MMDDYYS
MMSS
MMYY
MMYYB
MMYYC
MMYYD
MMYYN
MMYYP
MMYYS
MONNAME
MONTH
MONYY
NEGPAREN
NLDATE
NLDATEMN
NLDATEW
NLDATEWN
NLDATM
NLDATMAP
NLDATMTM
NLDATMW
NLDDFDD
NLDDFDDB
NLDDFDDC
NLDDFDDD
NLDDFDDP
NLDDFDDS
NLDDFDE
NLDDFDN
NLDDFDT
NLDDFDWN
NLDDFMN
NLDDFMY
NLDDFWDX
NLDDFWKX
NLMNIAUD
NLMNICAD
NLMNICHF
NLMNICNY
NLMNIDKK
NLMNIEUR
NLMNIGBP
NLMNIHKD
NLMNIILS
NLMNIJPY
NLMNIKRW
NLMNIMYR
NLMNINOK
NLMNINZD
NLMNIPLN
NLMNIRUR
NLMNISEK
NLMNISGD
NLMNITWD
NLMNIUSD
NLMNIZAR
NLMNLAUD
NLMNLCAD
NLMNLCHF
NLMNLCNY
NLMNLDKK
NLMNLEUR
NLMNLGBP
NLMNLHKD
NLMNLILS
NLMNLJPY
NLMNLKRW
NLMNLMYR
NLMNLNOK
NLMNLNZD
NLMNLPLN
NLMNLRUR
NLMNLSEK
NLMNLSGD
NLMNLTWD
NLMNLUSD
NLMNLZAR
NLMNY
NLMNYI
NLNUM
NLNUMI
NLPCT
NLPCTI
NLTIMAP
NLTIME
NORDFDD
NORDFDDB
NORDFDDC
NORDFDDD
NORDFDDP
NORDFDDS
NORDFDE
NORDFDN
NORDFDT
NORDFDWN
NORDFMN
NORDFMY
NORDFWDX
NORDFWKX
NUMX
OCTAL
PERCENT
PERCENTN
POLDFDD
POLDFDDB
POLDFDDC
POLDFDDD
POLDFDDP
POLDFDDS
POLDFDE
POLDFDN
POLDFDT
POLDFDWN
POLDFMN
POLDFMY
POLDFWDX
POLDFWKX
PTGDFDD
PTGDFDDB
PTGDFDDC
PTGDFDDD
PTGDFDDP
PTGDFDDS
PTGDFDE
PTGDFDN
PTGDFDT
PTGDFDWN
PTGDFMN
PTGDFMY
PTGDFWDX
PTGDFWKX
PVALUE
QTR
QTRR
RSTDOCNY
RSTDOCYY
RSTDONYN
RSTDOPNY
RSTDOPYN
RSTDOPYY
RUSDFDD
RUSDFDDB
RUSDFDDC
RUSDFDDD
RUSDFDDP
RUSDFDDS
RUSDFDE
RUSDFDN
RUSDFDT
RUSDFDWN
RUSDFMN
RUSDFMY
RUSDFWDX
RUSDFWKX
SLODFDD
SLODFDDB
SLODFDDC
SLODFDDD
SLODFDDP
SLODFDDS
SLODFDE
SLODFDN
SLODFDT
SLODFDWN
SLODFMN
SLODFMY
SLODFWDX
SLODFWKX
SVEDFDD
SVEDFDDB
SVEDFDDC
SVEDFDDD
SVEDFDDP
SVEDFDDS
SVEDFDE
SVEDFDN
SVEDFDT
SVEDFDWN
SVEDFMN
SVEDFMY
SVEDFWDX
SVEDFWKX
TIME
TIMEAMPM
TOD
WEEKDATE
WEEKDATX
WEEKDAY
WEEKU
WEEKV
WEEKW
WORDDATE
WORDDATX
YEAR
YEN
YEN
YYMM
YYMMB
YYMMC
YYMMD
YYMMDD
YYMMDDB
YYMMDDC
YYMMDDD
YYMMDDN
YYMMDDP
YYMMDDS
YYMMN
YYMMP
YYMMS
YYMON
YYQ
YYQB
YYQC
YYQD
YYQN
YYQP
YYQR
YYQRB
YYQRC
YYQRD
YYQRN
YYQRP
YYQRS
YYQS

Appendix J

SAS System Formats Supported by SAS Standalone Formats

$ASCII	CSYDFMN	DTYYQC	FRADFWDX	MINGUO
$BINARY	CSYDFMY	E	FRADFWKX	MMDDYY
$BYVAL	CSYDFWDX	ENGDFDD	FRSDFDD	MMDDYYB
$CHAR	CSYDFWKX	ENGDFDE	FRSDFDE	MMDDYYC
$CSTR	D	ENGDFDN	FRSDFDN	MMDDYYD
$EBCDIC	DANDFDD	ENGDFDT	FRSDFDT	MMDDYYN
$HEX	DANDFDE	ENGDFDWN	FRSDFDWN	MMDDYYP
$OCTAL	DANDFDN	ENGDFMN	FRSDFMN	MMDDYYS
$QUOTE	DANDFDT	ENGDFMY	FRSDFMY	MMSS
$REVERJ	DANDFDWN	ENGDFWDX	FRSDFWDX	MMYY
$REVERS	DANDFMN	ENGDFWKX	FRSDFWKX	MMYYC
$UPCASE	DANDFMY	ESPDFDD	HEX	MMYYD
$XPORTCH	DANDFWDX	ESPDFDE	HHMM	MMYYN
AFRDFDE	DANDFWKX	ESPDFDN	HOUR	MMYYP
AFRDFDN	DATE	ESPDFDT	HUNDFDD	MMYYS
AFRDFDT	DATEAMPM	ESPDFDWN	HUNDFDE	MONNAME
AFRDFDWN	DATETIME	ESPDFMN	HUNDFDN	MONTH
AFRDFMN	DAY	ESPDFMY	HUNDFDT	MONYY
AFRDFMY	DDMMYY	ESPDFWDX	HUNDFDWN	MRB
AFRDFWDX	DDMMYYB	ESPDFWKX	HUNDFMN	NEGPAREN
AFRDFWKX	DDMMYYC	EURDFDD	HUNDFMY	NENGO
BEST	DDMMYYD	EURDFDE	HUNDFWDX	NLDATE
BESTX	DDMMYYN	EURDFDN	HUNDFWKX	NLDATEMN
BINARY	DDMMYYP	EURDFDT	IB	NLDATEW
CATDFDD	DDMMYYS	EURDFDWN	IBR	NLDATEWN
CATDFDE	DESDFDD	EURDFMN	IEEE	NLDATM
CATDFDN	DESDFDE	EURDFMY	IEEER	NLDATMAP
CATDFDT	DESDFDN	EURDFWDX	ITADFDD	NLDATMTM
CATDFDWN	DESDFDT	EURDFWKX	ITADFDE	NLDATMW
CATDFMN	DESDFDWN	EURO	ITADFDN	NLDDFDD
CATDFMY	DESDFMN	EUROX	ITADFDT	NLDDFDE
CATDFWDX	DESDFMY	F	ITADFDWN	NLDDFDN
CATDFWKX	DESDFWDX	FINDFDD	ITADFMN	NLDDFDT
COMMA	DESDFWKX	FINDFDE	ITADFMY	NLDDFDWN
COMMAX	DEUDFDD	FINDFDN	ITADFWDX	NLDDFMN
COMMAX	DEUDFDE	FINDFDT	ITADFWKX	NLDDFMY
CRODFDD	DEUDFDN	FINDFDWN	JULDATE	NLDDFWDX
CRODFDE	DEUDFDT	FINDFMN	JULDAY	NLDDFWKX
CRODFDN	DEUDFDWN	FINDFMY	JULIAN	NLMNIAUD
CRODFDT	DEUDFMN	FINDFWDX	LOGPROB	NLMNICAD
CRODFDWN	DEUDFMY	FINDFWKX	MACDFDD	NLMNICHF
CRODFMN	DEUDFWDX	FLOAT	MACDFDE	NLMNICNY
CRODFMY	DEUDFWKX	FRACT	MACDFDN	NLMNIDKK
CRODFWDX	DOLLAR	FRADFDD	MACDFDT	NLMNIEUR
CRODFWKX	DOLLARX	FRADFDE	MACDFDWN	NLMNIGBP
CSYDFDD	DOWNAME	FRADFDN	MACDFMN	NLMNIHKD
CSYDFDE	DTDATE	FRADFDT	MACDFMY	NLMNIILS
CSYDFDN	DTMONYY	FRADFDWN	MACDFWDX	NLMNIJPY
CSYDFDT	DTWKDATX	FRADFMN	MACDFWKX	NLMNIKRW
CSYDFDWN	DTYEAR	FRADFMY	MDYAMPM	NLMNIMYR

NLMNINOK
NLMNINZD
NLMNIPLN
NLMNIRUR
NLMNISEK
NLMNISGD
NLMNITWD
NLMNIUSD
NLMNIZAR
NLMNLAUD
NLMNLCAD
NLMNLCHF
NLMNLCNY
NLMNLDKK
NLMNLEUR
NLMNLGBP
NLMNLHKD
NLMNLILS
NLMNLJPY
NLMNLKRW
NLMNLMYR
NLMNLNOK
NLMNLNZD
NLMNLPLN
NLMNLRUR
NLMNLSEK
NLMNLSGD
NLMNLTWD
NLMNLUSD
NLMNLZAR
NLMNY
NLMNYI
NLNUM
NLNUMI
NLPCT
NLPCTI
NLTIMAP
NLTIME
NORDFDD
NORDFDE
NORDFDN
NORDFDT
NORDFDWN
NORDFMN
NORDFMY
NORDFWDX
NORDFWKX
NUMX
OCTAL
ODDSR
PCPIB
PD
PDJULG
PDJULI
PERCENT
PERCENTN
PIB
PIBR
PK
POLDFDD
POLDFDE
POLDFDN
POLDFDT
POLDFDWN
POLDFMN
POLDFMY
POLDFWDX
POLDFWKX
PTGDFDD
PTGDFDE
PTGDFDN
PTGDFDT
PTGDFDWN
PTGDFMN
PTGDFMY
PTGDFWDX
PTGDFWKX
PVALUE
QTR
QTRR
RB
ROMAN
RUSDFDD
RUSDFDE
RUSDFDN
RUSDFDT
RUSDFDWN
RUSDFMN
RUSDFMY
RUSDFWDX
RUSDFWKX
S370FF
S370FHEX
S370FIB
S370FIBU
S370FPD
S370FPDU
S370FPIB
S370FRB
S370FZD
S370FZDL
S370FZDS
S370FZDT
S370FZDU
SETLOCALE
SIZEK
SIZEKB
SIZEKMG
SLODFDD
SLODFDE
SLODFDN
SLODFDT
SLODFDWN
SLODFMN
SLODFMY
SLODFWDX
SLODFWKX
SSN
SVEDFDD
SVEDFDE
SVEDFDN
SVEDFDT
SVEDFDWN
SVEDFMN
SVEDFMY
SVEDFWDX
SVEDFWKX
TIME
TIMEAMPM
TOD
VAXRB
WEEKDATE
WEEKDATX
WEEKDAY
WORDDATE
WORDDATX
WORDF
WORDS
XPORTFLT
XPORTINT
XYYMMDD
YEAR
YEN
YEN
YYMM
YYMMC
YYMMD
YYMMDD
YYMMDDB
YYMMDDC
YYMMDDD
YYMMDDN
YYMMDDP
YYMMDDS
YYMMN
YYMMP
YYMMS
YYMON
YYQ
YYQC
YYQD
YYQN
YYQP
YYQR
YYQRC
YYQRD
YYQRN
YYQRP
YYQRS
YYQS
YYQZ
Z
ZD