SAS® Enterprise Miner™ 5.3
C and Java Score Code Basics

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2008. *SAS® Enterprise Miner™ 5.3: C and Java Score Code Basics*. Cary, NC: SAS Institute Inc.

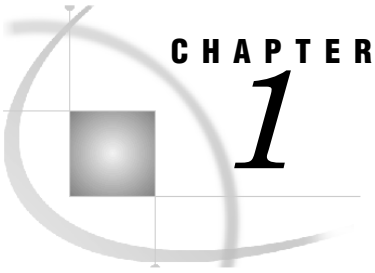
SAS® Enterprise Miner™ 5.3: C and Java Score Code Basics

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, July 2008

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents



CHAPTER
1

C and Java Score Code in SAS Enterprise Miner

Analytical data mining models generate score code that can be applied to new data in order to evaluate candidates for some defined event of interest. The model scoring code can exist in any number of programming languages. SAS Enterprise Miner generates model scoring code not only in SAS code, but for most models, in C and Java programming languages as well.

Generating model score code in programming languages like C and Java provides greater flexibility in organizational deployment. Data mining score code in C and Java can be combined with source code and binary files that are distributed with Enterprise Miner, and then compiled for deployment in external C, C++, or Java environments. Experienced C, C++, or Java programmers can use this feature to extend the functionality of new and existing software by embedding the power of SAS Enterprise Miner analytical model scoring.

SAS Enterprise Miner Nodes That Produce C and Java Score Code

SAS Enterprise Miner can generate C and Java score code for analytical models that are built from nodes that produce SAS DATA step scoring code. The following SAS Enterprise Miner nodes produce SAS DATA step scoring code, and are therefore able to create C and Java score code:

Sample Nodes:
Filter*
Explore Nodes:
Cluster
SOM\Kohonen
Variable Clustering
Variable Selection
Modify Nodes
Impute
Interactive Binning
Principal Components
Replacement
Rules Builder
Transform Variables*
Model Nodes:
AutoNeural
Decision Tree
Dmine Regression
DMNeural
Ensemble**
Gradient Boosting
Neural Network
Partial Least Squares
Regression
Rule Induction
Support Vector Machine
TwoStage
Assess Nodes:
Cutoff
Decisions
Score**
Model Comparison
Credit Scoring Nodes:
Interactive Grouping
Scorecard
Credit Exchange

* Generated C code applies only to data that has no numeric missing values.
**Requires all nodes in the process flow diagram to produce SAS DATA step scoring code.

The Enterprise Miner Score node can produce DATA step, C, and Java score code for most modeling process flow diagrams. However, a process flow diagram will not produce C or Java score code if the diagram includes a node that produces SAS code and also contains PROC statements or DATA statements. Enterprise Miner process flow diagrams that contain a node that is not listed in the above table will not generate C or Java code.

The Enterprise Miner SAS Code node is a special case because it is an open-ended tool for user-entered SAS code. Enterprise Miner does not translate user-entered SAS code into C and Java score code. When Enterprise Miner encounters a model process flow diagram that includes the SAS Code node, it attempts to generate C and Java score code for the remaining portions of the process flow diagram. For the portion of the process flow diagram represented by the SAS Code node, Enterprise Miner inserts a comment in the generated C and Java score code that indicates the omitted input. For example, the comment in generated C code might resemble the following:

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

In some cases, it may be possible for you to insert your own C code to take the place of the omitted SAS Code node content.

Enterprise Miner also does not translate SAS Code node content when it generates Java score code. When Enterprise Miner encounters a SAS Code node while generating Java code, the omitted code from the SAS Code node is replaced in the generated Java code with a call to a specific method.

Enterprise Miner produces source code for an empty stub method with that specific name. This might enable you to substitute your own Java code to take the place of the omitted SAS Code node content.

After you successfully run an Enterprise Miner model process flow diagram that generates C or Java score code, you can export the contents of your model as an SPK file that contains the generated C and Java code.

Generated C and Java Code

The C and Java code that Enterprise Miner generates is a conversion of the algorithms and operations that the SAS DATA step code performed in the process flow diagram. The generated C and Java code represents only the functions that are explicitly expressed in the SAS DATA step scoring code. The C score code that Enterprise Miner generates conforms to the "ISO/IEC 9899 International Standard for Programming Languages – C." The Java code that Enterprise Miner generates conforms to the Java Language Specification, published in 1996 by Addison-Wesley.

While generated C or Java scoring code from an Enterprise Miner analytical model can be used as the core analytics for a scoring system, you should not confuse the generated C or Java scoring code with a complete scoring system. In either C or Java languages, the programs that you write to enclose the scoring code must provide a suitable environment for performing the data analysis.

When Enterprise Miner creates C and Java scoring code, the code is output in several files. The output files can include some or all of the following items:

- Cscore.xml is the XML description of the model that produced the score code and the generated C code. The Cscore.xml file is valid XML. No Document Type Definition (DTD) is supplied.
- DB2_Score.c is C code for DB2 scalar user-defined functions for each of the output variables that are present in the scoring code.
- DS.class is the actual DATA step code that was converted to Java binary code. No Java source code is supplied.
- DS_UEXIT.java is generated only if the generated Java code contains omissions because one or more unsupported nodes exists in the model process flow diagram. The DS_UEXIT.java file is a Java source code template that you can use to provide your own code for the omitted portions of the process flow diagram.
- Jscore.xml is an XML description of the data mining model that produced the original score code as well as the generated Java code. Jscore.xml is valid XML. No DTD is supplied.
- JscoreUserFormats.java is the Java source code that supports any user-written formats that are used in the model. JscoreUserFormats.java is Java source code, which must be compiled before it can be executed.

- Score.c is the model process flow diagram score code, generated as a C function. Score.c is C source code, which must be compiled before it can be executed.

- Score.java is the Java source code that implements the interface to DS.class. Score.java is Java source code, which 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 generates the C code for IBM DB2 user-defined functions. IBM user-defined functions, or UDFs, are tools that you can use to write your own extensions to SQL. The functions that are integrated in DB2 are useful, but do not offer the customizable power of SAS Analytics. The Enterprise Miner generated UDFs enable you to greatly increase the efficiency, versatility, and power of your DB2 database. The key advantages of using UDFs are performance, modularity, and the object-oriented UDF process. The UDF code that Enterprise Miner generates is matched to each specific model's training data and the C scoring functions that are associated with the model.

The UDF code that Enterprise Miner generates is only one of several ways to create score code in DB2. The generated source code for the UDFs is simple but expandable. The comments in the UDF source code contain templates of SQL commands that need to be registered in order to invoke the generated UDFs.

Enterprise Miner can generate score functions that return values that are not useful in a scoring context. The UDFs that Enterprise Miner generates for a specific model are limited to the functions that return scoring output values that are considered to be of interest heuristically. The names of the scoring output variables are created by concatenating a prefix (for each kind of computed variable) with the name of the corresponding target variable (or decision data set).

Enterprise Miner produces UDFs for scoring output variables that begin with the following prefixes:

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

Enterprise Miner also produces UDFs for scoring output variables with the following names:

NODE	tree node identifier
SEGMENT	segment or cluster identifier
WARN	indicates problems with computing predicted values or making decisions
EM_CCF	average credit cost factor value
EM_CLASSIFICATION	fixed name for the I_ variable
EM_DECISION	fixed name for the D_*targetname* variable
EM_EVENTPROBABILITY	fixed name for the posterior probability of a target event
EM_EXPOSURE	average exposure value
EM_FILTER	identifies filtered observations
EM_LGD	average loss given default value
EM_PD	average predicted value
EM_PREDICTION	fixed name for the predicted value of an interval target
EM_PROBABILITY	fixed name for the maximum posterior probability associated with the predicted classification
EM_PROFITLOSS	fixed name for the value of expected profit or loss
EM_SEGMENT	fixed name for the name of the segment variable
SCORECARD_BIN	bin assigned to each observation
SCORECARD_POINTS	total score for each individual
SOM_DIMENSION1	identifies rows in a Self Organizing Map (SOM)
SOM_DIMENSION2	identifies columns in a SOM
SOM_ SEGMENT	identifies clusters created by a SOM

Most of the code in the UDFs that Enterprise Miner generates is designed 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 in order to load the input data vector for the score function.

Implementation Issues and Limitations

Creating a scoring application is a complex and advanced task that requires expertise in multiple areas. Creating and implementing a scoring system that uses C or Java score code generated by Enterprise Miner requires fluency and experience with DB2, SQL, C or Java language, application development skills, and attention to detail. Even with experience and expertise, the process of integrating advanced analytic code into a database in order to create a high performance scoring application approaches the limits of

current technology. When you work close to the edges of a new technology, you can gain unexpected rewards, but you should also expect problems. Testing your scoring application as well as the supporting UDFs is a critical step for your project.

Current DB2 code documentation states that each reference to a DB2 function (UDF or built-in) is allowed to have arguments that number from 0 to 90. The limitation on the number of arguments for each reference is a critical limitation for data mining jobs where even simple models can require hundreds of values. Enterprise Miner is capable of producing UDF code that contains more than 91 arguments, but DB2 cannot utilize any of the additional arguments.

Data Types

The UDFs that Enterprise Miner generates accept only two SQL data types: DOUBLE and VARCHAR. Most databases use more than two SQL data types, so you should use care when you convert your DB2 data types for UDF calls in your code. DB2 provides functions that you can use to convert most data types to DOUBLE or VARCHAR. Another way to handle additional SQL data types in training data and score data is to perform the required data type conversions during the extract, transfer, and load (ETL) step of your data preparation. You can also modify the UDF source code that Enterprise Miner generates in order to convert data types for scoring.

SAS Formats Support

SAS formats are SAS System functions that organize and configure raw data for display and analysis. There are two basic types of SAS formats: pre-defined formats and user-defined formats. Pre-defined formats are a part of SAS software. User-defined formats that are implemented in Enterprise Miner C and Java score code are supported primarily in the generated code. The SAS System formats for Java are supported through libraries that are distributed with Enterprise Miner. The SAS System formats for C are supported through the SAS Stand-alone Formats libraries.

SAS System Formats in C

The C scoring code that Enterprise Miner generates optionally supports SAS System formats through the SAS Stand-alone Formats product. The SAS Stand-alone Formats libraries are distributed with SAS in the client media on SAS Client-Side Components, Volume 1.

The Stand-alone Formats product contains (1) a header file that is needed for compilation, (2) the compiled code needed to link the Enterprise Miner generated score function, and (3) a set of callable run-time routines that the score code can invoke. Most of the SAS System formats have been packaged into libraries of loadable routines. The routines are loaded on demand at run time. The routines function with the scoring code that Enterprise Miner generates. The loadable routines do not require the SAS System environment. The SAS Stand-alone Formats that are needed for compiling,

linking, and running Enterprise Miner generated C code are as follows:

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

These routines have .dll file extensions in Windows. Windows also supplies the link library file, **jazxfbrg.lib**.

The C scoring function is linked to the jazxfbrg library. At run time, the code in jazxfbrg.lib dynamically loads the rest of the routines that are needed to support SAS System formats. You might need to only have the jazxfbrg library present when you link to the scoring function or application, but you will need to have all of the routines physically available at run time.

The Stand-alone Formats routines are accessed through calls to standard operating system functions, using a process called dynamic loading. Dynamic loading is an advanced topic in any C environment. The exact procedures, options, and environment variables that are used to compile, link, and run dynamically loaded code are different for every compiler, linker, and operating system. For example, in Windows, libraries are loaded from the environment variable PATH. The PATH variable must contain the directory path for the Stand-alone Formats files (jazwf*.dll).

On Solaris systems, the Stand-alone Formats dynamically load from shared libraries via the environment variable LD_LIBRARY_PATH. HP/UX systems use a different environment variable, SHLIB_PATH. You should thoroughly understand the procedures that your system requires in order to compile, link, and run dynamically loaded code if you want to successfully exploit C scoring code that uses Stand-alone Formats and was generated by Enterprise Miner.

The SAS Stand-alone Formats product supports most (but not all) of the SAS System formats. The list of supported formats is detailed in Appendix 5. The following SAS formats are not supported:

$VARYING	not loadable from the format package (not supported)
DOWNNAME	not loadable from the format package (not supported)
JULDAY	not loadable from the format package (not supported)
PD	no binary formats supported (SAS hexadecimal notation is not handled)

For environments where the Stand-alone Formats are not available, an experienced C programmer can modify the code in the provided **cscore.h** header file to remove the dependency on the Stand-alone Formats. (The **cscore.h** header file is distributed with Enterprise Miner and is located in

the directory **SASROOT/dmine/sasmisc**, where SASROOT is a substitute for the path specification to the root directory of your SAS installation.) Experienced C programmers can also modify the source code in **cscore.h** to support any format they want, using their own C code. If you want to write your own format functions, you can code them into the format logic of the **cscore.h** file.

The **cscore.h** file that is distributed with Enterprise Miner contains two examples of C formatting code: (1) $CHAR and (2) BEST formats. Both are partial implementations of the SAS System $CHAR and BEST formats. If you can accept the limitations of these two examples (no padding for $CHAR and no scientific notation for BEST), you can use the example formats as they are. If your C score code requires other formats, you can add supporting code for those formats in your **cscore.h** file. Your support code should follow the structure given in the existing $CHAR and BEST format examples in **cscore.h**. This enables you to support the formats that your specific score code requires without the SAS Stand-alone Formats.

If your deployment does not require Stand-alone Formats, you can disable them by modifying the **cscore.h** header file. To disable Stand-alone Formats support, set the preprocessor symbol FMTLIB to 0.

Formats in Java

When you use Enterprise Miner to generate Java scoring code, the SAS System formats are supported through a set of Java archives, or jar files. These jar files are distributed with Enterprise Miner and are located in the directory **SASROOT/dmine/sasmisc**, where SASROOT is a substitute place holder for the path specification to the root directory of your SAS installation.

The Java code that is distributed with Enterprise Miner supports SAS System Formats. The Java code that is distributed with Enterprise Miner is located in the directory SASROOT/dmine/sasmisc, where SASROOT is a substitute for the path specification to your SAS installation root directory.

The following Java code components are distributed with Enterprise Miner:

dtj.jar	nonstandard methods required by Enterprise Miner score code
sas.analytics.eminer.jsutil.jar	JscoreException and Jscore interface classes
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 the required Java class library files must be made available to the Java Virtual Machine (VM) in order to run the Enterprise Miner generated Java score code.

Support for Enterprise Miner Java formats includes most, but not all of the SAS System formats. Supported formats are listed in Appendix 4.

CHAPTER 2

Scoring Example

The scoring code that Enterprise Miner produces is affected by the choice of the data mining nodes that you use in your Enterprise Miner process flow diagram, by the sequence of the nodes in the process flow diagram, and by the data that you use to train your model. Likewise, changing the configuration of node settings in a process flow diagram, or modifying the variable roles, structure, or size of the training data set can change the generated scoring code. The score code that Enterprise Miner generates can be unique for every process flow diagram.

The following example is for illustrative purposes and is not intended to be deployed as a real application. The example includes sections for producing both C and Java score code. The example score code is generated using an Enterprise Miner client on a Windows system. After the score code is created, it is extracted. Then the extracted score code is moved to a Solaris system, where it can be compiled and run.

Create Folders for the Example

This example uses a number of folders or directories that you will need to create on your Enterprise Miner client. The example assumes that you will create the folders **`c:\temp\scorecode`**, **`c:\temp\scorecode\cscore`**, and **`c:\temp\scorecode\jscore`**.

Create Enterprise Miner Process Flow Diagram

1. Launch Enterprise Miner, create a new project, and in your new Enterprise Miner project, create a new diagram.
2. Use the Enterprise Miner Toolbar shortcut button for Create Data Source  to open the Data Source Wizard. Use the Data Source Wizard to specify the sample SAS table **`SAMPSIO.DMAGECR`**, and then use the wizard's Advanced Advisor setting to configure the **`SAMPSIO.DMAGECR`** variable **good_bad** as the target variable. Keep the wizard's default settings for the rest of the variables, and then save the All: German Credit data source with the data set role of **Train**.
3. Drag your newly created German Credit Data data source from the **Data Sources** folder of the Projects panel to the diagram workspace.
4. Drag an Interactive Grouping node from the **Credit Scoring** tab of the node toolbar to the diagram workspace, and connect it to the German Credit data source node. Leave the Interactive Grouping node in its default configuration.
   Note: The Interactive Grouping node is located in the **Credit Scoring** tab of the node toolbar in Enterprise Miner 5.3. If you are using Enterprise Miner 5.2, the Interactive Grouping node is located in the **Modify** tab of the node toolbar.
5. Drag a Regression node from the **Model** tab of the node toolbar to the diagram workspace, and connect it to the Interactive Grouping Node. Use the Selection Model property to configure the Regression node to perform Stepwise selection.
6. Drag a Score node from the **Assess** tab of the node toolbar to the diagram workspace, and connect it to the Regression node. Leave the Score node in its default configuration.
7. Right-click the Score node, select **Run**, and then select **Yes** in the confirmation dialog box to run your newly constructed process flow diagram.

The C scoring code and the Java scoring code that Enterprise Miner generates are handled differently. Depending on which type of score code you intend to compile and deploy, your next steps are provided in either the section on Scoring with C Code or in the section on Scoring with Java Code.

Scoring with C Code

The C scoring code that you generate with Enterprise Miner process flow diagrams and the C code header files that are distributed with Enterprise Miner can be compiled in most modern C or C++ development environments. The compilation results will vary, depending upon the compiler and its option settings. For example, some compilers produce warning messages about data type conversions because the compiler interprets data type conversion as a generic risk. Each compiler environment is different, and the range of option settings that are available through different compilers can generate different

results. It is up to you, the score code programmer, to properly configure and investigate your chosen compiler settings and warnings.

To invoke your scoring function and determine how the scoring function should be called, inspect the .c and the .xml files that Enterprise Miner generated. Your metadata file, **cscore.xml**, describes the model, the generated scoring function, and the generated scoring function's arguments.

By default, the generated scoring function accepts two pointers as arguments. Both arguments are arrays of the PARM data structure that is defined in the **csparm.h** header file that is distributed with Enterprise Miner. The **csparm.h** file is located in your SAS Enterprise Miner installation, in the directory **`SASROOT/dmine/sasmisc`**, where SASROOT is place holder text that substitutes for the path specification to the root directory of your SAS installation.

The first score function argument is a pointer to an array of input data values. The second argument is a pointer to an array of output data values. The calling program must allocate memory for both the input and output arrays. The exact size of the array of PARM structures and the order of each array's values can be determined by inspecting the code in your generated **score.c** or **cscore.xml** file.

Each PARM structure in the array contains a member named "data" that is a union object. The union is declared as an object of type DOUBLE and type CHAR *. You can assign appropriate double precision values directly to the **`data.fnum`** element of the union. For character values, you must allocate memory and assign it to the **`data.str`** element of the union before you can copy or assign any values. Each character variable in the input and output arrays must have memory allocated and assigned to its pointer in the union.

Inside the Cscore.xml File

You can use the contents of your generated **cscore.xml** file to determine the following information:

- the original variable names
- the position of each variable in its input or output array
- for character values, the "length" attribute, which specifies the amount of memory (in bytes) required for computation and display

The **cscore.xml** file that Enterprise Miner generates contains a root element that is called <Score>. The root element <Score> contains five sub-elements: <Producer>, <TargetList>, <Input>, <Output>, and <C>. The <C> element and it sub-elements contain the information you will need to complete your custom scoring code, such as original variable names, variable array position, and length of memory required for character variable values.

The <C> element contains the <Function> element. The <Function> element contains the name of the function in the <Name> element and the <ParameterList> element. The <ParameterList> element contains two children, each a <Parameter> sub-element. The first <Parameter> element describes the inputs to the generated function, and the second <Parameter> element describes the outputs that are generated by the function.

Consider the following portion of a **cscore.xml** file:

```
<Parameter>
 <Array length="12">
  <Type>Parm</Type>
  <DataMap>
   <Element index="0">
    <Value>
     <Origin> D_BAD </Origin>
      <Array length="10">
        <Type>char</Type>
      </Array>
    </Value>
   </Element>
   <Element index="1">
    <Value>
     <Origin> EL_BAD </Origin>
       <Type>double</Type>
    </Value>
   </Element>
```

The first child in each <Parameter> element is an <Array> element. The **length=** attribute of this <Array> element defines the number of Parm structures that the function requires.

The <Array> element contains a <Type> element that specifies the C type of the array, which is Parm. The <Parameter> element also contains a <DataMap> element. The <DataMap> element contains an <Element> child for each variable in the array.

Each <Element> element contains a <Value> element. Each <Value> element contains an <Origin> element, and either a <Type> element or an <Array> element with a **length=** attribute. The **length=** attribute specifies the required length, in bytes, of the memory buffer that the variable requires.

The memory buffer size, or length, that is specified for each char type variable in the **cscore.xml** file includes one extra byte for the null terminator that is required for all C string character values. If memory is not properly allocated for character values, or if character values are not null terminated, the results are undefined.

Save and Organize C Code Component Files

1. When your Enterprise Miner process flow diagram run completes, select the **Results** button in the Run Status window.
2. From the main menu in the Results window, select **View ➧ Scoring ➧ C Score Code** to open the C Score Code window.
3. In the C Score Code window, ensure that the list box at the bottom of the window is set to **Scoring Function Metadata**.
4. From the Results window main menu, select **File ➧ Save As** and save the file as **cscore.xml** in the **`c:\temp\scorecode\cscore`** directory that you created at the beginning of this example.

5. In the C Score Code window, return to the list box at the bottom of the window and change the setting from **Scoring Function Metadata** to **Score Code**.
6. From the Results window main menu, select **File ➧ Save As** and save the file as **score.c** in the **`c:\temp\scorecode\cscore`** directory that you created at the beginning of this example.
7. Locate the Client Media, SAS Client-Side Components, Volume 1, CD-ROM that shipped with your SAS System. This CD-ROM contains the Stand-alone Formats files for all supported systems.
8. On the SAS Client-Side Components, Volume 1, CD-ROM, use a file utility (such as Windows Explorer) to navigate to **`[cd drive letter]:\client1cd\safmts`**.
9. From the **`\safmts`** directory on the CD-ROM, copy the file **safmtss64.tar** to your C score example folder, **`c:\temp\scorecode\cscore`**.
10. Copy the files **cscore.h** and **csparm.h** from the **`\sasmisc`** folder of your Enterprise Miner installation into the **`c:\temp\scorecode\cscore`** folder that you created for this example. If your copy of Enterprise Miner was installed using default SAS file locations, the **`\sasmisc`** folder is located at **`c:\Program Files\sas\sas 9.1\dmine\sasmisc`** or **`c:\Program files\sas\sas 9.2\dmine\sasmisc`**, depending on your installed version of SAS.
11. By default, the **cscore.h** header file contains definitions and values that are specific to the Windows operating environment. Edit the copy of "cscore.h" to change these definitions and values for Solaris.
12. For C score code deployment on SPARC operating environments, change line 82 of the example **cscore.h** file so it reads as follows:

    ```
    #define MISSING UNX_BE_MISSING
    ```

    Each operating environment has its own value for missing data. Your **cscore.h** file must be modified for the environment where you plan to deploy your score code. See the section on Missing Values for more details.
13. Change line 88 of your **cscore.h** file so that it reads as follows:

    ```
    #define SFDKeyWords
    ```

    Some systems, such as Windows, require special source code directives in order to correctly store the function name in an executable object's export table. The cscore.h header file provides a macro SFDKeyWords for those systems. By default, the **cscore.h** header file macro is set for Windows systems. For systems other than Windows, or if you are creating an object other than a DLL, you will need to modify the SFDKeyWords macro. If your system does not require any directives, change the **#define** statement for the SFDKeyWords macro to define SFDKeyWords as blank.
14. Create the main C program that you will use to invoke your scoring function. Inspect your generated .c or .xml file to determine the requirements for calling the generated scoring function.

The main C program for this example can be as simple as the sample code that is provided in Appendix 3. Name your main C program file **csbasic.c** and copy it to your work folder in **c:\temp\scorecode\cscore**.

15. If you are deploying your code on a UNIX System, create a directory in your HOME directory and call it **example**.
16. In your new **example** directory, create a subdirectory called **cscore**.
17. Copy or FTP all of the following files to your **example/cscore** directory:

```
c:\temp\scorecode\cscore\safmtss64.tar
c:\temp\scorecode\cscore\csparm.h
c:\temp\scorecode\cscore\cscore.h
c:\temp\scorecode\cscore\Score.c
c:\temp\scorecode\cscore\Cscore.xml
```

Most FTP clients will take care of the carriage-returns in Windows text files. If not, most Solaris systems provide a dos2unix command that you can use to handle carriage returns. The dos2unix command is usually found in the **/bin** directory.

Compile, Link, and Run C Score Code in UNIX

All steps in this section are performed on the UNIX operating system.

1. Navigate to your UNIX **example/cscore** directory and unpack the Stand-alone Formats TAR file by submitting the following command:

```
tar xf safmtss64.tar
```

The tar process creates the **example/cscore/safmts** directory, which contains the Stand-alone Formats files.

2. The Stand-alone Formats routines are dynamically loaded from some of the files in the **safmts** folder. In the Solaris operating environment, you can modify the path that is searched for dynamically loaded code by setting the **LD_LIBRARY_PATH** environment variable. The **LD_LIBRARY_PATH** environment variable is read at process start-up, and is a colon-delimited list of locations to include in the load library search path. To include your newly extracted **safmts** directory in your Solaris load library path, enter this command:

```
LD_LIBRARY_PATH=.:$HOME/example/cscore/safmts
```

3. After you set the library path environmental variable, export the setting so that it is visible to your child processes. Enter this command:

```
export LD_LIBRARY_PATH
```

4. If GNU C Version 3.2.3 (SPARC-SUN-Solaris 2.8) is available, it is usually installed in **/usr/local/bin/.** You can use GNU C to compile and link your C scoring program with a single command. The link and compile command might resemble the following:

```
gcc -m64 -ansi -I$HOME/example/safmts csbasic.c Score.c
-lm $HOME/example/safmts/jazxfbrg
```

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

The link and compile command above should produce a single executable file called **a.**out.

5. To execute the main program, submit the following code:

```
a.out
```

The output from running the executable file **a.out** should resemble the following:

```
$ a.out

>> First observation...
csEM_CLASSIFICATION   = GOOD
csEM_EVENTPROBABILITY = 0.8710097610
csEM_PROBABILITY      = 0.8710097610
cs_WARN_              =

>> 4th observation...
csEM_CLASSIFICATION   = BAD
csEM_EVENTPROBABILITY = 0.4103733144
csEM_PROBABILITY      = 0.5896266856
cs_WARN_              =
$
```

Scoring with Java Code

Enterprise Miner can generate Java source code and binary class files. You must have access to a Java development environment in order to be able to use the Java code that you generate with Enterprise Miner. The Java code that is distributed with and produced by Enterprise Miner was developed with JAVA 2 SDK, Standard Edition, Version 1.2 software. You can obtain a Java Developer's Kit (JDK) from Sun at http://java.sun.com/.

Save and Organize Java Code Component Files

1. Run the example Enterprise Miner process flow diagram. When the run completes, click the **Results** button in the Run Status window.
2. Select **View ➧ Scoring ➧ Java Score Code** from the main menu of the Results window. The Java Score Code window opens.
3. In the Java Score Code window, ensure that the list box at the bottom of the window is set to **Scoring Function Metadata**.
4. From the Results window main menu, select **File ➧ Save As** and save the file as **JScore.xml** in the **`c:\temp\scorecode\jscore`** directory that you created at the beginning of this example.
5. In the Java Score Code window, return to the list box at the bottom and change the setting from **Scoring Function Metadata** to **Score Code**.
6. From the Results window main menu, select **File ➧ Save As** and save the file as **Score.java** in the **`c:\temp\scorecode\jscore`** directory that you created at the beginning of this example.
7. In the Java Score Code window, return to the list box at the bottom and change the setting from **Score Code** to **User-defined Formats**.
8. From the Results window main menu, select **File ➧ Save As** and save the file as **JscoreUserFormats.java** in the **`c:\temp\scorecode\jscore`** directory that you created at the beginning of this example.
9. In the Java Score Code window, return to the list box at the bottom and change the setting from **User-defined Formats** to **Java Class**. If no **Java Class** item exists in the list box, see SAS Technical Support Usage Note 14569.
10. Save the Java Class file as **DS.class** in the **`c:\temp\scorecode\jscore`** directory that you created at the beginning of this example.

Java Package Name

The Java code that you generate using Enterprise Miner has an assigned package name. The package name becomes the first part of the absolute class name. When you compile Java source code using a package name, the Java compiler (javac) searches for source and class files that are related to that package name. The compiler searches a hierarchical path that is relative to the current working directory. The Java compiler uses the package name to form the hierarchical storage location for each related file.

For example, the default Enterprise Miner Java package name is **eminer.user**. The Java compiler searches for the **eminer.user** package files in **`eminer\user\Score`**. Before you can compile the Java source code that Enterprise Miner generates, all of the generated Java files (except Jscore.xml) must reside in a hierarchical directory structure that corresponds to the levels of your package name structure.

This example assumes that the package name is eminer.user.Score. To change the default Enterprise Miner package name, select **Options ➧ Preferences** from the Enterprise Miner main menu to open the Preferences window. In the Preferences window, in the space to the right of the Java Score Code Package property, type your new package name (the name must have at least two levels) and click **OK**. In order for the change to take effect, you must run your process flow diagram after you change your Java Score Code Package.

Required Java Class Libraries

To compile or run Java code that you generated with Enterprise Miner, you must make the Java class libraries that are distributed with Enterprise Miner available to your Java environment. One method is to include the full path of each required class library in a CLASSPATH environment statement. The required Java class libraries that are distributed with SAS can be found at **`SASROOT/dmine/sasmisc`**.

Note: SASROOT is place holder text that substitutes for the path specification of the root directory of your SAS installation.

The following table describes the required Java class library files that are distributed with Enterprise Miner:

Class Library File	Class Library File Description
dtj.jar	compile time and run-time routines also used by Enterprise Miner for code generation
sas.analytics.eminer.jsutil.jar	JscoreException and Jscore interface classes binaries and source code
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

Create Directories to Compile and Deploy Score Code

1. Locate the folder on your Enterprise Miner client that contains the Enterprise Miner miscellaneous files. On UNIX systems, the normal folder location is **`SASROOT/sasmisc/dmine`**. On Windows systems, the normal folder location is **`c:\Program Files\sas\sas 9.1\dmine\sasmisc`**. Copy the files **dtj.jar**, **sasjsutl.jar**, **sastext.jar**, **sas.core.jar**, and **sas.entities.jar** from the miscellaneous files folder to the folder that you created at the beginning of this example, **`c:\temp\scorecode\jscore`**.
2. Determine the package name for your Enterprise Miner generated Java code. One way is to view the Jscore.xml file and find the Java class name. The Java class name should contain the package name. This example uses the class name eminer.user.Score.Score. Remove the last qualifier Score, and the remainder is the package name: eminer.user.Score.

3. You must provide a Java main program. The Java main program needs to instantiate the generated scoring class, provide input data, invoke the score method, and handle the scoring outputs. Appendix 2 contains an example Java main program. For this example, save the code in Appendix 2 as Jsbasic.java, and move the Jsbasic.java file to the folder that you created at the beginning of this example, **c:\temp\scorecode\jscore**.
4. On the UNIX system where the score code will be deployed, create the following directory structure in your HOME directory:

   $HOME/example/jscore/eminer/user/Score

5. FTP or copy all the *.jar files from the Windows folder **c:\temp\scorecode\jscore** to the UNIX folder that you created, **$HOME/example/jscore**.

FTP or otherwise copy all the *.java and related *.class files from the **c:\temp\scorecode\jscore** folder to the UNIX folder at **$HOME/example/jscore/eminer/user/Score**.

When you are finished copying files to your UNIX system, the list of files in the **$HOME/example/jscore** directory should look like this:

```
$ ls -1
dtj.jar
eminer
sas.analytics.eminer.jsutil.jar
sas.core.jar
sas.entities.jar
sas.text.jar
```

The list of files in your **$HOME/example/jscore/eminer/user/Score** directory should look like this:

```
$ ls -1
DS.class
Jsbasic.java
JscoreUserFormats.java
Score.java
```

Note: The *.java files will need to be stripped of the carriage returns that are used in Windows text files. Many FTP clients automatically handle stripping carriage-returns from text files. If your FTP client does not clean up carriage returns in text files, most Solaris systems provide a dos2unix command that cleans up DOS and Windows text files. The dos2unix command executable is usually found in the UNIX **/bin** directory.

Compile and Run Java Score Code in UNIX

All steps in this section are performed on the UNIX operating system.

1. Set your CLASSPATH environment variable so that it contains the absolute path to the *.jar files that you are using for your Enterprise Miner Java scoring: **sas.analytics.eminer.jsutil.jar**, **dtj.jar**, **sas.text.jar**, **sas.core.jar**, and **sas.entities.jar**. You can set the CLASSPATH environment variable at a command line prompt using a colon-delimited path list such as the following:

```
export CLASSPATH=.
:$HOME/example/jscore/dtj.jar
:$HOME/example/jscore/sas.analytics.eminer.jsutil.jar
:$HOME/example/jscore/sas.core.jar
:$HOME/example/jscore/sas.entities.jar
:$HOME/example/jscore/sas.text.jar
```

2. Your current working directory should be the parent directory of the package tree. The parent directory of the package tree in the example is **$HOME/example/jscore**. Invoke the Java compiler on the source files using a command that resembles the following:

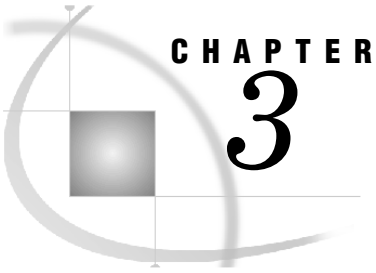
```
javac eminer/user/Score/*.java
```

3. The result should be a set of newly-created Java class files that implement the Jscore interface.
4. After you compile the the Jsbasic main program and the Enterprise Miner generated source code, copy the **Jsbasic.class** file from **$HOME/example/jscore/eminer/user/Score** to the working directory you want to use to deploy the Java scoring code.
5. To execute your Java scoring code program, on the command line enter this code:

```
java Jsbasic
```

6. The output from your Java scoring code program should resemble the following:

```
>> First observation...
EM_CLASSIFICATION = GOOD
EM_EVENTPROBABILITY = 0.8710097609996974
EM_PROBABILITY = 0.8710097609996974
_WARN_ =

>> 4th observation...
EM_CLASSIFICATION = BAD
EM_EVENTPROBABILITY = 0.41037331440653074
EM_PROBABILITY = 0.5896266855934693
_WARN_ =
```



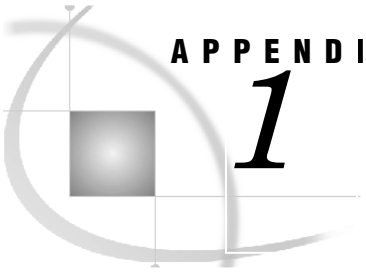
CHAPTER 3

Notes and Tips

Missing Values

In the SAS System, missing numeric 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, which means 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.

For C scoring, the NaN bit pattern must be set in the **cscore.h** header file. Text in the body of **cscore.h** contains recommendations for NaN values that you should use with systems that are supported by SAS. NaNs can be problematic if they are not handled correctly. When you develop a scoring system, you should consider how to handle numeric data that contains missing values. In the C language, it may be necessary to use a C function to perform a bit check on each numeric value before you perform an operation on the numeric value. Examples of C functions that perform bit checks are available in **cscore.h** as the nmiss function and in the C main program **csbasic.c** used in this example as the amiss function.



APPENDIX

1 Programming Information

General Code Limitations

The SAS DATA step language is a flexible and powerful development environment. The Enterprise Miner component that generates C and Java scoring code supports only a small portion of the syntax, expressions, and functions that the SAS System supports. Every effort has been made to ensure that the DATA step code that Enterprise Miner produces is compatible with the restrictions imposed by the C and Java code generation process.

It is possible to create code in Enterprise Miner that cannot be correctly translated into C or Java code. This is particularly a problem with data transformations performed within Enterprise Miner. When you use the Enterprise Miner Expression Builder to create transformations, and you want to migrate your scoring code to C or Java, you must take great care to ensure that your data transformations are expressed using code structures that resemble C or Java structures as much as possible, in order to facilitate the correct generation of score code. It is best to attempt to structure DATA step code for any transformation to be as much like C as possible. In other words, any SAS operand or function that is not native to the C or Java languages should be avoided in your data transformation expressions unless the operand or function is explicitly supported by the C and Java code generation process.

Supported Functions

The following SAS System functions are supported either directly by the target language libraries or by code that is 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);
```

Note: n1, n2, …, nx indicates numeric variables, and c1, c2, …, cx indicates character variables.

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 might have a missing value must be tested for missing values before any other operation is performed. When comparing a character type variable with a quoted character constant, the quoted character constant must be the second operand.

Variable Name Length

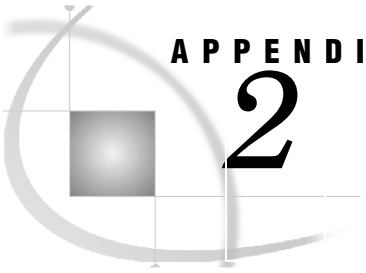
Enterprise Miner truncates column names at 32 bytes. Care must be exercised during the ETL process for training data and scoring data in order to be sure that the first 32 characters of all column names are unique.

Character Data Length

The maximum allowable length for character data in Enterprise Miner is 32 bytes. Care must be exercised during the ETL process for training data and scoring data to be sure that the first 32 characters of all character data types are unique.

Extended Character Sets

The generation of C and Java score code for data and variable names that use extended character sets is not supported. The generation of C and Java score code requires single-byte length characters. Multi-byte character names and data are not supported. Generated Java code that contains single-byte, extended character set names and data is untested and unsupported. Because the C code that Enterprise Miner generates 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.



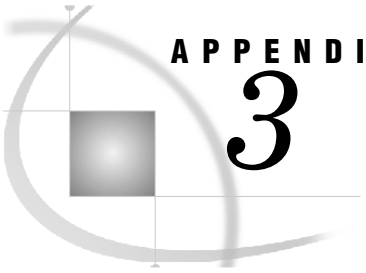
APPENDIX 2

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

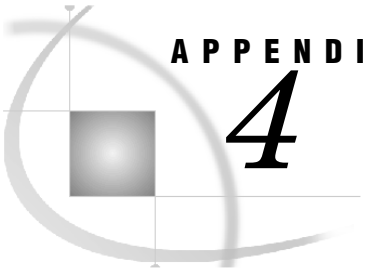
3 Example C Main Program

```
/*-------------------------------------------------------------------
 * CSBASIC - Enterprise Miner C scoring example program simulates
 *           scoring data from the first and fourth rows of the EM
 *           sample data set DMAGECR.
 *
 * V3
 *-------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "csparm.h"

/* EM Score function prototype */
void score ( PARM *, PARM *);

/*-------------------------------------------------------------------
 * Numeric Missing Value definition copied from cscore.h
 *-------------------------------------------------------------------*/
 /* for Windows (little endian) */
#define WIN_LE_MISSING  (*((double*)"\0\0\0\0\0\xD1\xff\xff"))

/* Set the system specific value of missing  */
#define MISSING  WIN_LE_MISSING

/*-------------------------------------------------------------------
 * Sizes derived from Cscore.xml
 *-------------------------------------------------------------------*/
#define InSize 4
#define OutSize 12

/*-------------------------------------------------------------------
 * Definitions copied from EM generated C source file
 *-------------------------------------------------------------------*/
#define csCHECKING            indata[0].data.fnum
#define csDURATION            indata[1].data.fnum
#define csHISTORY             indata[2].data.fnum
#define csPURPOSE             indata[3].data.str

#define csEM_CLASSIFICATION   outdata[0].data.str
#define csEM_EVENTPROBABILITY outdata[1].data.fnum
#define csEM_PROBABILITY      outdata[2].data.fnum
#define csGRP_CHECKING        outdata[3].data.fnum
#define csGRP_DURATION        outdata[4].data.fnum
#define csGRP_HISTORY         outdata[5].data.fnum
#define csGRP_PURPOSE         outdata[6].data.fnum
#define csI_GOOD_BAD          outdata[7].data.str
#define csP_GOOD_BADBAD       outdata[8].data.fnum
```

```
#define csP_GOOD_BADGOOD      outdata[9].data.fnum
#define csU_GOOD_BAD          outdata[10].data.str
#define cs_WARN_              outdata[11].data.str

int main(argc, argv)
     int argc;
     char *argv[];
{
    PARM * indata;             /* score function input argument    */
    PARM * outdata;            /* score function output argument   */

   /*-------------------------------------------------------------------
    * Allocate and clear memory for score function inputs and outputs
    *-------------------------------------------------------------------*/
   indata =  (PARM *)malloc(sizeof(PARM)*InSize);
   outdata = (PARM *)malloc(sizeof(PARM)*OutSize);
   memset(outdata,0,sizeof(PARM)*OutSize);
   memset(indata,0, sizeof(PARM)*InSize);

   /*-------------------------------------------------------------------
    * Memory for all character type parameters must be allocated
    * Lengths derived from Cscore.xml
    *-------------------------------------------------------------------*/
   /* indata[3].data.str   */
   csPURPOSE   = (char *)malloc(33);
   memset(csPURPOSE,0,sizeof(char)*33);

   /* outdata[0].data.str  */
   csEM_CLASSIFICATION    = (char *)malloc(33);
   memset(csEM_CLASSIFICATION,0,sizeof(char)*33);

   /* outdata[7].data.str*/
   csI_GOOD_BAD = (char *)malloc(9);
   memset(csI_GOOD_BAD,0,sizeof(char)*9);

   /*outdata[10].data.str */
   csU_GOOD_BAD= (char *)malloc(9);
   memset(csU_GOOD_BAD,0,sizeof(char)*9);

   /* outdata[11].data.str */
   cs_WARN_= (char *)malloc(5);
   memset(cs_WARN_,0,sizeof(char)*5);

   /*-------------------------------------------------------------------
    * Always initialize outputs to the type appropriate missing value
    *-------------------------------------------------------------------*/
   strncpy(csEM_CLASSIFICATION," ",2);  /* outdata[0].data.str  */
   csEM_EVENTPROBABILITY = MISSING;     /* outdata[1].data.fnum */
   csEM_PROBABILITY      = MISSING;     /* outdata[2].data.fnum */
   csGRP_CHECKING        = MISSING;     /* outdata[3].data.fnum */
   csGRP_DURATION        = MISSING;     /* outdata[4].data.fnum */
   csGRP_HISTORY         = MISSING;     /* outdata[5].data.fnum */
   csGRP_PURPOSE         = MISSING;     /* outdata[6].data.fnum */
   strncpy(csI_GOOD_BAD," ",2);         /* outdata[7].data.str  */
   csP_GOOD_BADBAD       = MISSING;     /* outdata[8].data.fnum */
   csP_GOOD_BADGOOD      = MISSING;     /* outdata[9].data.fnum */
   strncpy(csU_GOOD_BAD," ",2);         /* outdata[10].data.str */
   strncpy(cs_WARN_," ",2);             /* outdata[11].data.str */

   /*-------------------------------------------------------------------
    * Instead of reading in the data, this sets example values
```

```
 * from the first row in sample data set DMAGECR
 *-------------------------------------------------------------------*/
csCHECKING = 1.0;           /* indata[0].data.fnum */
csDURATION = 6.0;           /* indata[1].data.fnum */
csHISTORY  = 4.0;           /* indata[2].data.fnum */
strncpy(csPURPOSE,"3",33);  /* indata[3].data.str  */

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

/*-------------------------------------------------------------------
 * print some outputs to stdout
 *-------------------------------------------------------------------*/
printf("\n>> First observation...\n");
printf("csEM_CLASSIFICATION   = %s\n", csEM_CLASSIFICATION);
printf("csEM_EVENTPROBABILITY = %12.10f\n", csEM_EVENTPROBABILITY);
printf("csEM_PROBABILITY      = %12.10f\n", csEM_PROBABILITY );
printf("cs_WARN_              = %s\n", cs_WARN_);

/*-------------------------------------------------------------------
 * Always initialize all outputs to the type appropriate missing
 * value.
 *-------------------------------------------------------------------*/
strncpy(csEM_CLASSIFICATION," ",2);  /* outdata[0].data.str  */
csEM_EVENTPROBABILITY = MISSING;     /* outdata[1].data.fnum */
csEM_PROBABILITY      = MISSING;     /* outdata[2].data.fnum */
csGRP_CHECKING        = MISSING;     /* outdata[3].data.fnum */
csGRP_DURATION        = MISSING;     /* outdata[4].data.fnum */
csGRP_HISTORY         = MISSING;     /* outdata[5].data.fnum */
csGRP_PURPOSE         = MISSING;     /* outdata[6].data.fnum */
strncpy(csI_GOOD_BAD," ",2);         /* outdata[7].data.str  */
csP_GOOD_BADBAD       = MISSING;     /* outdata[8].data.fnum */
csP_GOOD_BADGOOD      = MISSING;     /* outdata[9].data.fnum */
strncpy(csU_GOOD_BAD," ",2);         /* outdata[10].data.str */
strncpy(cs_WARN_," ",2);             /* outdata[11].data.str */

/*-------------------------------------------------------------------
 * Instead of reading in new data, this sets example values
 * from the 4th row in sample data set DMAGECR
 *-------------------------------------------------------------------*/
csCHECKING = 1.0;           /* indata[0].data.fnum */
csDURATION = 42.0;          /* indata[1].data.fnum */
csHISTORY  = 2.0;           /* indata[2].data.fnum */
strncpy(csPURPOSE,"2",33);  /* indata[3].data.str  */

/*-------------------------------------------------------------------
 * Call the EM generated C scoring function a second time
 *-------------------------------------------------------------------*/
score(indata,outdata);

/*-------------------------------------------------------------------
 * print some outputs to stdout
 *-------------------------------------------------------------------*/
printf("\n>> 4th observation...\n");
printf("csEM_CLASSIFICATION   = %s\n", csEM_CLASSIFICATION);
printf("csEM_EVENTPROBABILITY = %12.10f\n", csEM_EVENTPROBABILITY);
printf("csEM_PROBABILITY      = %12.10f\n", csEM_PROBABILITY );
printf("cs_WARN_              = %s\n", cs_WARN_);
```

```
   /*---------------------------------------------------------------
    * clean up allocated memory
    *---------------------------------------------------------------*/
   free(csPURPOSE);            /* indata[3].data.str   */
   free(csEM_CLASSIFICATION);  /* outdata[0].data.str  */
   free(csI_GOOD_BAD);         /* outdata[7].data.str  */
   free(csU_GOOD_BAD);         /* outdata[10].data.str */
   free(cs_WARN_);             /* outdata[11].data.str */
   free(indata);
   free(outdata);

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



APPENDIX 4

SAS System Formats Supported Java Scoring

$
$ASCII
$BINARY
$CHAR
$F
$HEX
$OCTAL
AFRDFDD
AFRDFDDB
AFRDFDDC
AFRDFDDD
AFRDFDDP
AFRDFDDS
AFRDFDE
AFRDFDN
AFRDFDT
AFRDFDWN
AFRDFMN
AFRDFMY
AFRDFWDX
AFRDFWKX
BEST
BINARY
CATDFDD
CATDFDDB
CATDFDDC
CATDFDDD
CATDFDDP
CATDFDDS
CATDFDE
CATDFDN

CATDFDT
CATDFDWN
CATDFMN
CATDFMY
CATDFWDX
CATDFWKX
COMMA
COMMAX
COMMAX
CRODFDD
CRODFDDB
CRODFDDC
CRODFDDD
CRODFDDP
CRODFDDS
CRODFDE
CRODFDN
CRODFDT
CRODFDWN
CRODFMN
CRODFMY
CRODFWDX
CRODFWKX
CSYDFDD
CSYDFDDB
CSYDFDDC
CSYDFDDD
CSYDFDDP
CSYDFDDS
CSYDFDE
CSYDFDN
CSYDFDT
CSYDFDWN
CSYDFMN
CSYDFMY
CSYDFWDX
CSYDFWKX
DANDFDD
DANDFDDB
DANDFDDC
DANDFDDD
DANDFDDP
DANDFDDS

DANDFDE
DANDFDN
DANDFDT
DANDFDWN
DANDFMN
DANDFMY
DANDFWDX
DANDFWKX
DATE
DATEAMPM
DATETIME
DAY
DDMMYY
DDMMYYB
DDMMYYC
DDMMYYD
DDMMYYN
DDMMYYP
DDMMYYS
DESDFDD
DESDFDDB
DESDFDDC
DESDFDDD
DESDFDDP
DESDFDDS
DESDFDE
DESDFDN
DESDFDT
DESDFDWN
DESDFMN
DESDFMY
DESDFWDX
DESDFWKX
DEUDFDD
DEUDFDDB
DEUDFDDC
DEUDFDDD
DEUDFDDP
DEUDFDDS
DEUDFDE
DEUDFDN
DEUDFDT
DEUDFDWN

DEUDFMN
DEUDFMY
DEUDFWDX
DEUDFWKX
DOLLAR
DOLLARX
DOWNAME
DTDATE
DTMONYY
DTWKDATX
DTYEAR
DTYYQC
E
ENGDFDD
ENGDFDDB
ENGDFDDC
ENGDFDDD
ENGDFDDP
ENGDFDDS
ENGDFDE
ENGDFDN
ENGDFDT
ENGDFDWN
ENGDFMN
ENGDFMY
ENGDFWDX
ENGDFWKX
ESPDFDD
ESPDFDDB
ESPDFDDC
ESPDFDDD
ESPDFDDP
ESPDFDDS
ESPDFDE
ESPDFDN
ESPDFDT
ESPDFDWN
ESPDFMN
ESPDFMY
ESPDFWDX
ESPDFWKX
EURDFDD
EURDFDDB

EURDFDDC
EURDFDDD
EURDFDDP
EURDFDDS
EURDFDE
EURDFDN
EURDFDT
EURDFDWN
EURDFMN
EURDFMY
EURDFWDX
EURDFWKX
EURO
F
FINDFDD
FINDFDDB
FINDFDDC
FINDFDDD
FINDFDDP
FINDFDDS
FINDFDE
FINDFDN
FINDFDT
FINDFDWN
FINDFMN
FINDFMY
FINDFWDX
FINDFWKX
FRADFDD
FRADFDDB
FRADFDDC
FRADFDDD
FRADFDDP
FRADFDDS
FRADFDE
FRADFDN
FRADFDT
FRADFDWN
FRADFMN
FRADFMY
FRADFWDX
FRADFWKX
FRSDFDD

FRSDFDDB
FRSDFDDC
FRSDFDDD
FRSDFDDP
FRSDFDDS
FRSDFDE
FRSDFDN
FRSDFDT
FRSDFDWN
FRSDFMN
FRSDFMY
FRSDFWDX
FRSDFWKX
HEX
HHMM
HOUR
HUNDFDD
HUNDFDDB
HUNDFDDC
HUNDFDDD
HUNDFDDP
HUNDFDDS
HUNDFDE
HUNDFDN
HUNDFDT
HUNDFDWN
HUNDFMN
HUNDFMY
HUNDFWDX
HUNDFWKX
ITADFDD
ITADFDDB
ITADFDDC
ITADFDDD
ITADFDDP
ITADFDDS
ITADFDE
ITADFDN
ITADFDT
ITADFDWN
ITADFMN
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

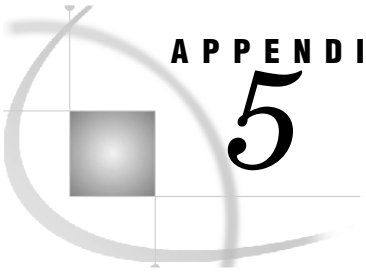
NLDATMAP
NLDATMTM
NLDATMW
NLDDFDD
NLDDFDDB
NLDDFDDC
NLDDFDDD
NLDDFDDP
NLDDFDDS
NLDDFDE
NLDDFDN
NLDDFDT
NLDDFDWN
NLDDFMN
NLDDFMY
NLDDFWDX
NLDDFWKX
NLMNIAUD
NLMNICAD
NLMNICHF
NLMNICNY
NLMNIDKK
NLMNIEUR
NLMNIGBP
NLMNIHKD
NLMNIILS
NLMNIJPY
NLMNIKRW
NLMNIMYR
NLMNINOK
NLMNINZD
NLMNIPLN
NLMNIRUR
NLMNISEK
NLMNISGD
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 5

SAS System Formats Supported for C Scoring

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

MMDDYYN
MMDDYYP
MMDDYYS
MMSS
MMYY
MMYYC
MMYYD
MMYYN
MMYYP
MMYYS
MONNAME
MONTH
MONYY
MRB
NEGPAREN
NENGO
NLDATE
NLDATEMN
NLDATEW
NLDATEWN
NLDATM
NLDATMAP
NLDATMTM
NLDATMW
NLDDFDD
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
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

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.