

SAS[®] Enterprise Miner[™] 6, 7, 12, and 13 C and Java Score Code Basics

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® Enterprise Miner™ 6, 7, 12, and 13: C and Java Score Code Basics*. Cary, NC: SAS Institute Inc.

SAS® Enterprise Miner™ 6, 7, 12, and 13: C and Java Score Code Basics

Copyright © 2013, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

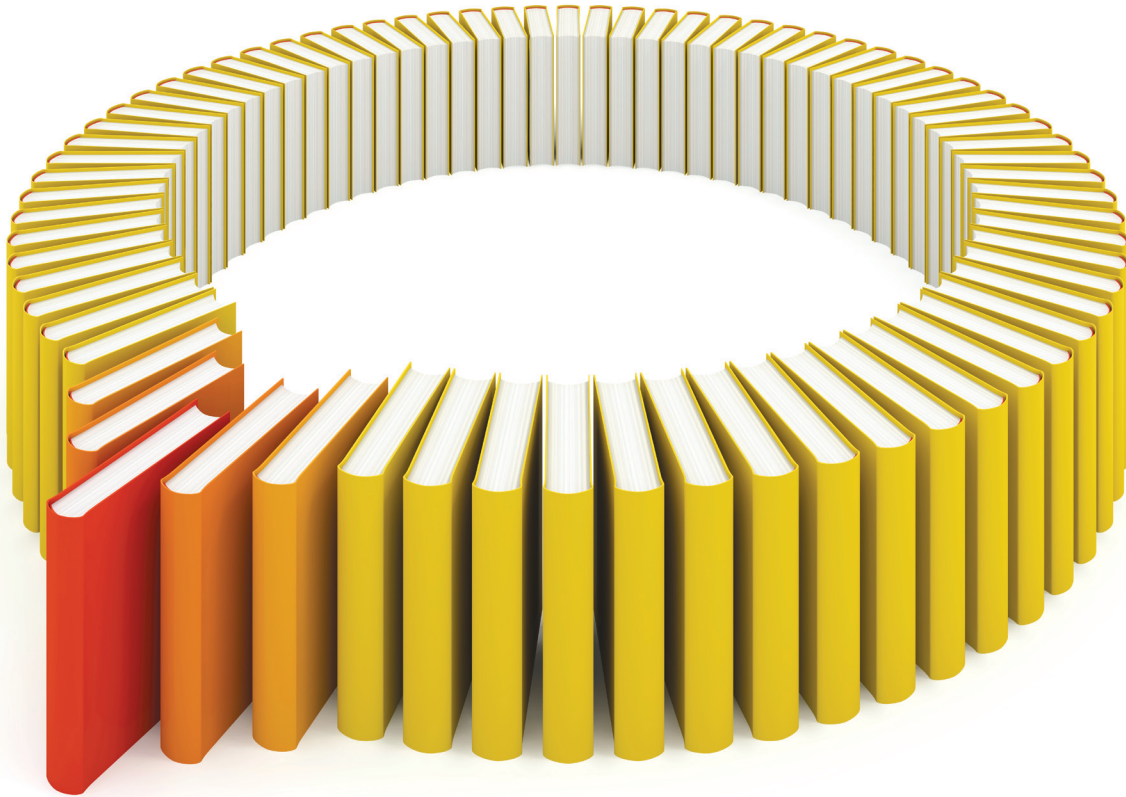
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

December 2013

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit support.sas.com/bookstore or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.



Contents

Chapter 1 C and Java Score Code in SAS Enterprise Miner 1

SAS Enterprise Miner Tools That Produce C and Java Score Code 2

SAS Formats Support 4

Generated C and Java Code 5

Generated C Code 5

DB2 User-Defined Functions 5

C Code Usage 7

C Formats Support 8

 C Formats Support Distribution 9

 C Formats Usage 9

Generated Java Code 10

 Java Package Name 11

Java Code Usage 11

Java Scoring JAR Files 11

 Java Scoring JAR File Distribution 12

 Java Scoring JAR File Usage 12

 SAS System Formats 12

Chapter 2 Scoring Example 13

Create Folders for the Example 13

Gather Files 13

Create SAS Enterprise Miner Process Flow Diagram 15

Scoring with C Code 15

 Save and Edit C Code Component Files 16

 Organize C Code Component Files 17

 Compile, Link, and Run C Score Code in UNIX 17

Scoring with Java Code 19

 Save and Organize Java Code Component Files 20

 Create Java Main Program 20

 Compile and Run Java Score Code in UNIX 22

Chapter 3 C and Java Score Code in SAS Enterprise Miner 23

SAS Enterprise Miner Tools That Produce C and Java Score Code 24

SAS Formats Support 26

Generated C and Java Code 27

Generated C Code 27

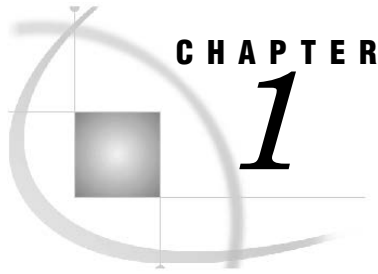
DB2 User-Defined Functions 27

C Code Usage 29

C Formats Support 30

 C Formats Support Distribution 31

C Formats Usage	31
Generated Java Code	32
Java Package Name	33
Java Code Usage	33
Java Scoring JAR Files	33
Java Scoring JAR File Distribution	34
Java Scoring JAR File Usage	34
SAS System Formats	34
Appendix 1 Programming Information	35
General Code Limitations	35
Supported Functions	36
Supported SAS Operators	37
Arithmetic Operators	37
Comparison Operators	37
Logical Operators	37
Other Operators	37
Conditional Statement Syntax	38
Variable Name Length	38
Character Data Length	38
Extended Character Sets	38
Appendix 2 Example Java Main Program	39
Appendix 3 Example C Main Program	41
Appendix 4 SAS System Formats Supported Java Scoring	45
Appendix 5 SAS System Formats Supported for C Scoring	51
Appendix 6 C Compiler Command Examples	53
C Compiler Command Examples	53
W32 – Windows 32-bit (x86)	53
LAX- Linux for x64 (x86-64)	55
LNX- Linux 32-bit (x86)	56
H64- HP-UX on PA-RISC	57
H6I- HP-UX on Itanium	59
R64 – AIX on Power	61
S64 - Solaris on SPARC	62
SAX – Solaris 10 x64 (x64-86)	63



C and Java Score Code in SAS Enterprise Miner

<i>SAS Enterprise Miner Tools That Produce C and Java Score Code</i>	<i>2</i>
<i>SAS Formats Support.....</i>	<i>4</i>
<i>Generated C and Java Code</i>	<i>5</i>
<i>Generated C Code.....</i>	<i>5</i>
<i>DB2 User-Defined Functions</i>	<i>5</i>
<i>DB2 Data Types</i>	<i>7</i>
<i>C Code Usage</i>	<i>7</i>
<i>C Formats Support.....</i>	<i>8</i>
<i>C Formats Support Distribution</i>	<i>9</i>
<i>C Formats Usage</i>	<i>9</i>
<i>Generated Java Code</i>	<i>10</i>
<i>Java Package Name</i>	<i>11</i>
<i>Java Code Usage</i>	<i>11</i>
<i>Java Scoring Jars</i>	<i>11</i>
<i>Java Scoring Jars Distribution.....</i>	<i>12</i>
<i>Java Scoring Jars Usage.....</i>	<i>12</i>
<i>SAS System Formats.....</i>	<i>12</i>

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. These files are distributed with SAS 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.

It should be emphasized that creating a scoring application is a very complex and highly advanced task that requires expertise in several areas. The likelihood of successfully implementing a scoring system that incorporates C or Java code that is generated in SAS Enterprise Miner is exactly proportional to your fluency and experience with the environment in which you choose to implement your application. Testing of both the application and the generated code are critical to the success of any such project.

SAS Enterprise Miner Tools That Produce C and Java Score Code

SAS Enterprise Miner can generate C and Java score code for most analytical models that are built from nodes that produce SAS DATA step scoring code. The following list of SAS Enterprise Miner nodes by area indicates which nodes can produce C and Java score code. Any nodes that are not listed cannot produce C or Java score code.

Sample Tools	
C and Java Generated	No C or Java score code
Filter Node	Append
	Data Partition ^{nc}
	File Import ^{nc}
	Input Data ^{nc}
	Merge
	Sample ^{nc}
	Time Series ^{nc}

Explore Tools	
C and Java Generated	No C or Java score code
Cluster	Association
SOM/Kohonen	DMDB ^{nc}
Variable Clustering	Graph Explore ^{nc}
Variable Selection	Market Basket
	Multiplot ^{nc}
	Path Analysis
	Stat Explore ^{nc}
	Text Miner

Modify Tools	
C and Java Generated	No C or Java score code
Impute	Drop ^{nc}
Interactive Binning	
Principal Components	
Replacement	
Rules Builder **	
Transform Variables **	

Model Tools	
C and Java Generated	No C or Java score code
AutoNeural	MBR
Decision Tree	
Dmine Regression	
DMNeural	
Ensemble***	
Gradient Boosting	
LARS	
Model Import	
Neural Network	
Partial Least Squares	
Regression	
Rule Induction	
Two Stage	

Utility Tools	
C and Java Generated	No C or Java score code
End Groups	Control Point ^{nc}
Start Groups	Merge
	Metadata ^{nc}
	Reporter ^{nc}
	SAS Code

Credit Scoring Tools	
C and Java Generated	No C or Java score code
Credit Exchange	Reject Inference ^{nc}
Interactive Grouping	
Scorecard	

^{nc} Tool produces no score code.

** It is possible to create code that cannot be correctly generated as C or Java. When you are creating transformations or expressions in the Transformation tool or the Rules Builder, careful inspection and testing is required to make sure your C and Java score code is generated correctly.

*** Depends on members of process flow diagram.

**** Any nodes that are not listed in the above tables cannot produce C or Java score code.

The SAS Enterprise Miner Score node can produce DATA step, C, and Java score code for most modeling process flow diagrams. However, a process flow diagram does 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. SAS Enterprise Miner process flow diagrams that contain a node not listed in the above table do not generate C or Java code.

The SAS Enterprise Miner SAS Code node is a special case because it is an open-ended tool for user-entered SAS code. SAS Enterprise Miner does not translate user-entered SAS code into C and Java score code. When SAS 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 that is represented by the SAS Code node, SAS 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 might be possible for you to insert your own C code to take the place of the omitted SAS Code node content.

SAS Enterprise Miner also does not generate Java score code for SAS Code node content. When SAS 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. SAS 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.

SAS Formats Support

SAS formats are functions used in the SAS System to configure the size, form, or pattern of raw data for display and analysis. There are two basic types of SAS formats: the pre-defined formats that are supplied with all SAS Systems, and the formats that are defined by the customer. The formats that are defined by the customer are also referred to as user-defined formats. The user-defined formats that are used in the generated C and Java score code are supported by a combination of generated code and distributed functions. The SAS System formats for Java are supported through libraries that are distributed with SAS Enterprise Miner. The SAS System formats for C are supported through libraries that are distributed as the SAS Stand-alone Formats.

Generated C and Java Code

The C and Java code that SAS Enterprise Miner generates is a conversion of the algorithms and operations that the SAS DATA step code performed in the process flow diagram. You can use the tools available in SAS Enterprise Miner to generate valid SAS score code that cannot be correctly generated as C or Java score code. Any generated code should be tested thoroughly before deployment.

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 SAS Enterprise Miner generates conforms to the “ISO/IEC 9899 International Standard for Programming Languages – C.” The Java code that SAS Enterprise Miner generates conforms to the Java Language Specification, published in 1996 by Addison-Wesley.

You can use generate C or Java scoring code from a SAS Enterprise Miner analytical model as the core for a scoring system, but you should not confuse the generated C or Java scoring code with a complete scoring system. In both C or Java languages, the programs that you write to enclose the scoring code must provide a suitable environment for performing the data analysis.

After you successfully run a SAS Enterprise Miner model process flow diagram that generates C or Java score code, you can export your model as an SPK file that contains the generated C and Java code, or you can use the File menu in the results browser to save individual files.

Generated C Code

The C scoring code is generated as several output files. They include 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 Document Type Definition (DTD) is supplied.
- **DB2_Score.c** is C code for DB2 scalar User Defined Functions for each of the output variables defined in the scoring code.
- **Score.c** is the model score code that **is** generated as a C function. It is C source code and must be compiled before it can be executed.

DB2 User-Defined Functions

In addition to generating the scoring algorithms that are developed in SAS 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 UDFs that are generated by SAS Enterprise

Miner 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 SAS 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 SAS 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.

SAS Enterprise Miner can generate score functions that return values that are not useful in a scoring context. The UDFs that SAS 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 type of computed variable) with the name of the corresponding target variable (or decision data set).

SAS 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

SAS 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_ <i>targetname</i> 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 that is 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 SAS 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.

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. SAS Enterprise Miner is capable of producing UDF code that contains more than 91 arguments, but DB2 cannot use any of the additional arguments.

DB2 Data Types

The UDFs that SAS 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 SAS Enterprise Miner generates in order to convert data types for scoring.

C Code Usage

To compile, link, and run C code that is generated in SAS Enterprise Miner, you need to first gather the required tools, libraries, and files.

The generated C code conforms to the “ISO/IEC 9899 International Standard for Programming Languages – C”, so any current compiler should be able to compile the code.

Other than the standard C libraries, the generated C code will have dependencies on the SAS Stand-alone Formats libraries. See the SAS Formats section below for details.

The generated C code also depends on three C header files.

- **cscore.h**
- **csparm.h**
- **jazz.h**

The **cscore.h** and **csparm.h** files are distributed with the SAS Enterprise Miner Server Windows systems. They are located in **SASROOT\dmisc\sasmisc**. For UNIX systems, they are located in **SASROOT/misc/dmisc**. Copy them to your development environment. The **cscore.h** file has several operating system specific definitions that will, in most cases, need to be modified for your target operating system. Those modifications are documented in comments in the header file and in the following example.

The **jazz.h** header file is distributed with the SAS Stand-alone Formats product. See details below in the C Formats Support section. Copy it to your development environment.

To run the generated C code, you need to create a main program to invoke the score function. You can view the **score.c** file and inspect it to determine how the score function should be called.

The metadata file **Cscore.xml** also describes the generated function and its arguments. By default, the generated function accepts two pointers as arguments. The first argument points to an array of input data values. The second argument points to an array of output data values. The memory, which is required for each data value, must be allocated by the calling program.

Both arrays must be composed of the PARM data structure that is 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 for each character value is not allocated before 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 that are 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. Such a main program can be as simple as the code in [Appendix 3](#).

C Formats Support

The C scoring code that is generated in SAS Enterprise Miner supports the use of SAS System Formats through the SAS Stand-alone Formats product. The SAS Stand-alone Formats do not depend on a SAS System environment in any way. The SAS Stand-alone Formats product contains a header file. It also contains a set of libraries that are needed for compilation, linking, and running the SAS Enterprise Miner-generated C scoring function. The SAS

Stand-alone Formats are a set of link and run-time libraries that are compiled for each supported operating system. Those include the following:

- Windows 32-bit
- Windows Itanium 64-bit
- Windows x64-bit
- Solaris 64-bit
- AIX 64-bit
- Linux 32-bit
- Linux 64-bit
- HP-Itanium 64-bit
- HP 64-bit

C Formats Support Distribution

The SAS Stand-alone Formats are distributed as downloads from the SAS Customer Support website. Look in the Knowledge Base section for Samples and SAS Notes. Search for Note 35872, titled “SAS Stand-alone Formats for SAS Enterprise Miner C Score Code.” Follow the instructions to download the package for your target operating system.

C Formats Usage

The C scoring function or application that is generated in SAS Enterprise Miner is linked to **jazxfbrg**. **This means** that at run time the code in **jazxfbrg** can dynamically load the rest of the routines that are needed to support the SAS System formats. Although only **jazxfbrg** might need to be present when linking the function or application, all of the files must be available at run time.

For the Stand-alone 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 that are 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 directory path for the Stand-alone Formats shared libraries (**jazwf***). The value of this environment variable must be the fully qualified directory name for the directory that holds the Stand-alone Formats.

On Solaris systems, the Stand-alone Formats are dynamically loaded from shared libraries via the environment variable LD_LIBRARY_PATH. HP and UNIX systems use a slightly different environment variable, SHLIB_PATH. A thorough understanding of your target system’s procedures for compiling, linking, and running with dynamically loaded code is required to successfully exploit the Stand-alone Formats and the code that is generated by the SAS Enterprise Miner C Scoring component.

For environments where the Stand-alone Formats support is not available or not desired, it should be possible for an experienced C programmer to modify

the source code in the **cscore.h** header file that is distributed with SAS Enterprise Miner in **SASROOT/dmine/sasmisc**. The object of the modifications is to remove the dependency on the Stand-alone Formats and to support any format that they want, with their own C code.

If you write your own format functions, you can integrate those functions into the logic that handles formats in **cscore.h**. The **cscore.h** file that is distributed with SAS Enterprise Miner already contains two examples of such C formatting code—partial support for \$CHAR and BEST formats. If your situation enables you to accept the limitations of those examples (no padding for \$CHAR, and no scientific notation for BEST), you can use the example formats without any modification. You can also add any additional formats that you might need. In that case, C scoring code that is generated by SAS Enterprise Miner will contain only those formats, and will be compiled with the **cscore.h** header file that will support those formats.

In cases where Stand-alone Formats support is not desired, the dependency on the Stand-alone formats support can be removed. This can be accomplished by modifying a copy of the **cscore.h** header file that is distributed with SAS Enterprise Miner in **SASROOT/dmine/sasmisc**. In the **cscore.h** file, the preprocessor symbol **FMTLIB** is set to 0, which disables support for the SAS Stand-alone formats.

Generated Java Code

Java scoring code is generated in several output files. The primary model logic is generated as a Java class file. The other files are generated as Java source files, and the model's variable metadata is encoded as XML. The Java code that is generated by SAS Enterprise Miner is compatible with the version of Java that SAS Enterprise Miner uses. The generated Java files might include some or all of the following:

- **DS.class** is the actual DATA step code that is generated directly to Java binary code. There is no Java source code supplied.
- **DS_UEXIT.java** is generated only if code from unsupported tools was omitted from the generated Java code. This Java source code is a template that customers 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 is supplied.
- **JscoreUserFormats.java** is the Java source code that supports any user-written formats that might be used in the model. It is Java 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.

After you run a SAS 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 SAS Enterprise Miner Reference Help.

Java Package Name

The code that is generated by SAS Enterprise Miner 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) searches for the related source and class files by the package name in a path relative to the current working directory. The Java compiler uses the package name to form a hierarchical path for each related file. For example, if the package name has the default of "eminer.user.Score," the Java compiler searches for the package's files in the path **eminer\user\Score**. In order to compile the generated Java source code, all of the generated Java files (Jscore.xml is not required) must be placed in a directory or folder tree that looks like the package name "eminer.user.Score." You can change the default package name in the SAS Enterprise Miner Client before the flow is run. On the the main menu, select **Options** ➔ **Preferences**. Then fill in a package name of at least two levels.

Java Code Usage

Java scoring code that is generated by SAS Enterprise Miner depends on the classes and methods that are distributed as the SAS Enterprise Miner Java Scoring JAR files. In order to compile or run Java score code that is generated by SAS Enterprise Miner, you need to copy the supporting Java archives and configure your system to make the JAR files available to Java.

Java Scoring JAR Files

The SAS Enterprise Miner Java Scoring JAR files support the classes and methods that are used in the generated Java code, including the use of SAS formats. The SAS Enterprise Miner Java Scoring JAR files are as follows:

- **dtj.jar**
- **icu4j.jar**
- **sas.analytics.eminer.jsutil.jar**
- **sas.core.jar**
- **sas.core.nls.jar**
- **sas.icons.jar**
- **sas.icons.nls.jar**
- **sas.nls.collator.jar**
- **tkjava.nls.jar**

These JAR files include support for most, but not all of the SAS System formats. The list of supported Java formats is detailed in [Appendix 4](#).

Java Scoring JAR File Distribution

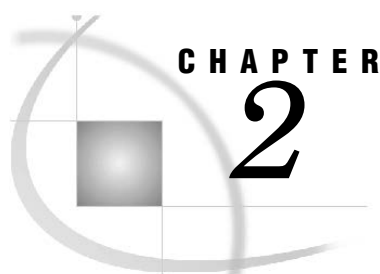
The SAS Enterprise Miner Java scoring JAR files are distributed as part of the SAS Enterprise Miner Server image. On Windows systems, they are found in the path `SASROOT\dmisc\sasmisc`. For UNIX systems, check the path `SASROOT/misc/dmisc`. It is recommended that you save copies of your Java scoring JAR files in your scoring environment.

Java Scoring JAR File Usage

Wherever you want to compile and run the Java code that is generated by SAS Enterprise Miner, you need to make the SAS Enterprise Miner Java scoring JAR files available to Java. Adding the directory path that contains your Java scoring JAR files to your CLASSPATH environment variable enables both the compile and execution steps.

SAS System Formats

Supported SAS System formats are listed in Appendix 4.



Scoring Example

<i>Create Folders for the Example.....</i>	<i>13</i>
<i>Gather Files.....</i>	<i>13</i>
<i>Create Enterprise Miner Process Flow Diagram.....</i>	<i>15</i>
<i>Scoring with C Code.....</i>	<i>15</i>
<i>Save and Edit C Code Component Files</i>	<i>16</i>
<i>Organize C Code Component Files.....</i>	<i>17</i>
<i>Compile, Link, and Run C Score Code in UNIX.....</i>	<i>17</i>
<i>Scoring with Java Code.....</i>	<i>19</i>
<i>Save and Organize Java Code Component Files</i>	<i>20</i>
<i>Create Java Main Program.....</i>	<i>20</i>
<i>Compile and Run Java Score Code in UNIX.....</i>	<i>22</i>

The scoring code that SAS Enterprise Miner produces is affected by the choice of the data mining nodes that you use in your SAS 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 SAS 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 a SAS 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 SAS Enterprise Miner client. The example assumes that you will create the folders `c:\temp\scorecode`, `c:\temp\scorecode\cscore`, and `c:\temp\scorecode\jscore`.


Gather Files

For the C scoring example, locate the SAS Stand-alone Formats for the system on which you will be scoring. In this example, we will run the code that is generated by SAS Enterprise Miner on a Solaris system that requires the **safmtss64.tar** file. The “C Formats Support Distribution” section of this

document contains additional details about locating the SAS Stand-alone Formats.

1. Copy the TAR file to the temporary cscore folder that you created for this example: **C:\Temp\ScoreCode\cscore**.
2. Locate the **\sasmisc** folder that is created when the Workspace Server for SAS Enterprise Miner is installed. The default path for the sasmisc folder on a Windows Workspace Server for the SAS Enterprise Miner installation is **C:\Program Files\SAS\SASFoundation\9.2\dmisc\sasmisc**.
3. Copy two files, **cscore.h** and **csparm.h**, from the **\sasmisc** folder to the **C:\Temp\ScoreCode\cscore** Folder.
4. For the Java scoring example, locate the folder in your Workspace Server for the SAS Enterprise Miner installation that contains the SAS Enterprise Miner Java Scoring JAR files. The default folder location in UNIX is **!SASROOT/misc/dmisc**. The default folder location on Windows systems is **C:\Program Files\SAS\SASFoundation\9.2\dmisc\sasmisc**.
5. Copy the SAS Enterprise Miner Java Scoring JAR files from the installation source folder to the local folder that you created at the beginning of this example: **C:\Temp\ScoreCode\jscore**.

Create SAS Enterprise Miner Process Flow Diagram

1. Launch SAS Enterprise Miner and create a new project. In your new SAS Enterprise Miner project, create a new diagram.
2. Click the SAS 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**. 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. 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. 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 on the **Credit Scoring** tab of the node toolbar in SAS Enterprise Miner 5.3. If you are using SAS Enterprise Miner 5.2, the **Interactive Grouping** node is located on the **Modify** tab of the node toolbar.
5. Drag a **Regression** node from the **Model** tab of the node toolbar to the diagram workspace. 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. Connect it to the **Regression** node. Leave the **Score** node in its default configuration.
7. Right-click the **Score** node, click **Run**, and then click **Yes** in the confirmation dialog box to run your newly constructed process flow diagram.

The C scoring code and the Java scoring code that SAS 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 SAS Enterprise Miner process flow diagrams can be compiled in most modern C or C++ development environments. The compilation results will vary, depending on 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. You, as the score code programmer,

need to decide how to properly configure and investigate your chosen compiler settings and warnings.

Save and Edit C Code Component Files

1. When your SAS Enterprise Miner process flow diagram run completes, click **Results** in the Run Status window.
2. On 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. On the Results window main menu, select **File ➤ Save As**. 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. On the Results window main menu, select **File ➤ Save As**. Save the file as **score.c** in the **c:\temp\scorecode\cscore** directory that you created at the beginning of this example.
7. For the Solaris SPARC architecture, change the value used for missing values. Search the **cscore.h** file for the text string, “#define MISSING”.

You should find a line that looks like this:

```
#define MISSING WIN_LE_MISSING
```

Edit this line so that it reads as follows:

```
#define MISSING UNX_BE_MISSING
```

Each operating system has its own value for missing. Cscore.h must be modified for each system.

Change the defined value of SFDKeywords to be blank. Some systems require special directives to correctly store the function name in an objects export table. The **cscore.h** header file provides a macro called SFDKeywords for those systems.

By default, the SFDKeywords macro is configured for the Windows directive. 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 situation does not require any directives, change your #define statement for the SFDKeywords macro to define SFDKeywords as blank.

Search your **cscore.h** file for the string, “SFDKeywords”. You should find a line that resembles the following:

```
#define SFDKeywords extern __declspec( dllexport )
```

Edit the `#define SFDKeyWords` statement so that it reads as follows:

```
#define SFDKeyWords
```

Then save your changes and close the `cscore.h` file.

Organize C Code Component Files

1. Create a main program to invoke the score function. For this example, such a main program can be copied from [Appendix 3](#). Name the main program file `csbasic.c` and copy it to `C:\temp\ScoreCode\cscore`.
2. In your HOME directory on the target UNIX System, create a directory named `/example`.
3. In your new `/example` directory, create a subdirectory called `\cscore`.
4. 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 in the UNIX operating system.

1. Navigate to your UNIX `example/cscore` directory. Unpack the SAS 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 SAS Stand-alone Formats files.

2. Copy the C header file `jazz.h` from the `/safmts` directory to the folder that you created for this example, `~HOME/example/cscore`.
3. The SAS Stand-alone Formats routines are dynamically loaded from some of the files in the `/safmts` folder. In the Solaris operating environment, you can use the `LD_LIBRARY_PATH` environment variable to modify the path that is searched for dynamically loaded code. The `LD_LIBRARY_PATH` environment variable is read at process start-up. It 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
```

4. 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
```

5. If GNU C Version 3.2.3 (Oracle Solaris for SPARC 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 SAS Enterprise Miner
-lm	specifies the math link library
jazxfbrg	an object library from which the SAS Stand-alone Formats objects are linked

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

6. 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

SAS 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 SAS Enterprise Miner. The Java code distributed with and produced by SAS Enterprise Miner was developed with the Java SE Development Kit 6. You can obtain a Java Developer's Kit (JDK) from Sun at <http://www.oracle.com/technetwork/java/index.html>.

Save and Organize Java Code Component Files

1. Run the example SAS Enterprise Miner process flow diagram. When the run completes, click **Results** in the Run Status window.
2. Select **View ➤ Scoring ➤ Java Score Code** on 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. On the Results window main menu, select **File ➤ Save As**. 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. On the Results window main menu, select **File ➤ Save As**. 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. On the Results window main menu, select **File ➤ Save As**. Save the file as **JscoreUserFormats.java** in the `c:\temp\scorecode\jscore` directory that you created at the beginning of this example.
9. Save the Java Class file as **DS.class** in the `c:\temp\scorecode\jscore` directory that you created at the beginning of this example.

Create Java Main Program

1. Determine the package name of the generated Java code. One way to determine the package name is to view the **Jscore.xml** file and look up the Java class name. In this example, the class name is **eminer.user.Score.Score**. Remove the last qualifier "Score" from the class name, and the remainder **eminer.user.Score** is the package name.
2. 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**. Move the **Jsbasic.java** file to the folder that you created at the beginning of this example, `c:\temp\scorecode\jscore`.
3. 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
```

4. 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.`

5. FTP or otherwise copy all the **JAVA** and related **CLASS** files from the `c:\temp\scorecode\jscore` folder to the UNIX folder at `$HOME/example/jscore/eminer/user/Score.`
6. When you are finished, the list of files in the `$HOME/example/jscore` directory should resemble the following:

```
$ ls -l

dtj.jar
eminer
icu4j.jar
sas.analytics.eminer.jusutil.jar
sas.core.jar
sas.core.nls.jar
sas.icons.jar
sas.icons.nls.jar
tkjava.jar
tkjava.nls.jar
```

7. The contents of the `$HOME/example/jscore/eminer/user/Score` directory should resemble the following:

```
$ ls -l

DS.class
Jsbasic.java
JscoreUserFormats.java
Score.java
```

Note: Windows JAVA files will need to have the carriage-returns in the body removed. Many FTP clients automatically remove carriage-returns in text files. If not, most Solaris systems provide a **dos2unix** command to perform that task. The **dos2unix** command 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 the CLASSPATH environment variable to contain absolute paths for the JAR files that are distributed with SAS Enterprise Miner Java Scoring. Here is one way to set environment variables: At a prompt, on a single line, enter something that resembles the following:

```
export
CLASSPATH=.:$HOME/example/jscore/dtj.jar:$HOME/example/j
score/sas.analytics.eminer.jsutil.jar:$HOME/example/jscor
e/sas.core.jar:$HOME/example/jscore/sas.core.nls.jar:$H
OME/example/jscore/sas.icons.jar:$HOME/example/jscore/sa
s.icons.nls.jar:$HOME/example/jscore/sas.nls.collator.ja
r:$HOME/example/jscore/sas.icons.jar:$HOME/example/jscor
e/tkjava.nls.jar:$HOME/example/jscore/icu4j.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
```

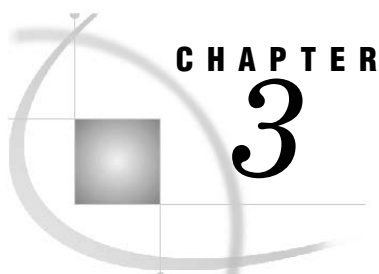
3. The result should be a set of newly-created Java class files that implement the Jscore interface.
4. After you compile the Jsbasic main program and the SAS Enterprise Miner generated source code, copy the **Jsbasic.class** file from `$HOME/example/jscore/eminer/user/Score` to the working directory that 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_ =

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



C and Java Score Code in SAS Enterprise Miner

<i>SAS Enterprise Miner Tools That Produce C and Java Score Code</i>	<i>24</i>
<i>SAS Formats Support.....</i>	<i>26</i>
<i>Generated C and Java Code</i>	<i>27</i>
<i>Generated C Code.....</i>	<i>27</i>
<i>DB2 User-Defined Functions</i>	<i>27</i>
<i>DB2 Data Types</i>	<i>29</i>
<i>C Code Usage</i>	<i>29</i>
<i>C Formats Support.....</i>	<i>30</i>
<i>C Formats Support Distribution</i>	<i>31</i>
<i>C Formats Usage</i>	<i>31</i>
<i>Generated Java Code</i>	<i>32</i>
<i>Java Package Name</i>	<i>33</i>
<i>Java Code Usage</i>	<i>33</i>
<i>Java Scoring Jars</i>	<i>33</i>
<i>Java Scoring Jars Distribution.....</i>	<i>34</i>
<i>Java Scoring Jars Usage.....</i>	<i>34</i>
<i>SAS System Formats.....</i>	<i>34</i>

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. These files are distributed with SAS 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.

It should be emphasized that creating a scoring application is a very complex and highly advanced task that requires expertise in several areas. The likelihood of successfully implementing a scoring system that incorporates C or Java code that is generated in SAS Enterprise Miner is exactly proportional to your fluency and experience with the environment in which you choose to implement your application. Testing of both the application and the generated code are critical to the success of any such project.

SAS Enterprise Miner Tools That Produce C and Java Score Code

SAS Enterprise Miner can generate C and Java score code for most analytical models that are built from nodes that produce SAS DATA step scoring code. The following list of SAS Enterprise Miner nodes by area indicates which nodes can produce C and Java score code. Any nodes that are not listed cannot produce C or Java score code.

Sample Tools	
C and Java Generated	No C or Java score code
Filter Node	Append
	Data Partition ^{nc}
	File Import ^{nc}
	Input Data ^{nc}
	Merge
	Sample ^{nc}
	Time Series ^{nc}

Explore Tools	
C and Java Generated	No C or Java score code
Cluster	Association
SOM/Kohonen	DMDB ^{nc}
Variable Clustering	Graph Explore ^{nc}
Variable Selection	Market Basket
	Multiplot ^{nc}
	Path Analysis
	Stat Explore ^{nc}
	Text Miner

Modify Tools	
C and Java Generated	No C or Java score code
Impute	Drop ^{nc}
Interactive Binning	
Principal Components	
Replacement	
Rules Builder **	
Transform Variables **	

Model Tools	
C and Java Generated	No C or Java score code
AutoNeural	MBR
Decision Tree	
Dmine Regression	
DMNeural	
Ensemble***	
Gradient Boosting	
LARS	
Model Import	
Neural Network	
Partial Least Squares	
Regression	
Rule Induction	
Two Stage	

Utility Tools	
C and Java Generated	No C or Java score code
End Groups	Control Point ^{nc}
Start Groups	Merge
	Metadata ^{nc}
	Reporter ^{nc}
	SAS Code

Credit Scoring Tools	
C and Java Generated	No C or Java score code
Credit Exchange	Reject Inference ^{nc}
Interactive Grouping	
Scorecard	

^{nc} Tool produces no score code.

** It is possible to create code that cannot be correctly generated as C or Java. When you are creating transformations or expressions in the Transformation tool or the Rules Builder, careful inspection and testing is required to make sure your C and Java score code is generated correctly.

*** Depends on members of process flow diagram.

**** Any nodes that are not listed in the above tables cannot produce C or Java score code.

The SAS Enterprise Miner Score node can produce DATA step, C, and Java score code for most modeling process flow diagrams. However, a process flow diagram does 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. SAS Enterprise Miner process flow diagrams that contain a node not listed in the above table do not generate C or Java code.

The SAS Enterprise Miner SAS Code node is a special case because it is an open-ended tool for user-entered SAS code. SAS Enterprise Miner does not translate user-entered SAS code into C and Java score code. When SAS 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 that is represented by the SAS Code node, SAS 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 might be possible for you to insert your own C code to take the place of the omitted SAS Code node content.

SAS Enterprise Miner also does not generate Java score code for SAS Code node content. When SAS 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. SAS 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.

SAS Formats Support

SAS formats are functions used in the SAS System to configure the size, form, or pattern of raw data for display and analysis. There are two basic types of SAS formats: the pre-defined formats that are supplied with all SAS Systems, and the formats that are defined by the customer. The formats that are defined by the customer are also referred to as user-defined formats. The user-defined formats that are used in the generated C and Java score code are supported by a combination of generated code and distributed functions. The SAS System formats for Java are supported through libraries that are distributed with SAS Enterprise Miner. The SAS System formats for C are supported through libraries that are distributed as the SAS Stand-alone Formats.

Generated C and Java Code

The C and Java code that SAS Enterprise Miner generates is a conversion of the algorithms and operations that the SAS DATA step code performed in the process flow diagram. You can use the tools available in SAS Enterprise Miner to generate valid SAS score code that cannot be correctly generated as C or Java score code. Any generated code should be tested thoroughly before deployment.

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 SAS Enterprise Miner generates conforms to the “ISO/IEC 9899 International Standard for Programming Languages – C.” The Java code that SAS Enterprise Miner generates conforms to the Java Language Specification, published in 1996 by Addison-Wesley.

You can use generated C or Java scoring code from a SAS Enterprise Miner analytical model as the core for a scoring system, but you should not confuse the generated C or Java scoring code with a complete scoring system. In both C or Java languages, the programs that you write to enclose the scoring code must provide a suitable environment for performing the data analysis.

After you successfully run a SAS Enterprise Miner model process flow diagram that generates C or Java score code, you can export your model as an SPK file that contains the generated C and Java code, or you can use the File menu in the results browser to save individual files.

Generated C Code

The C scoring code is generated as several output files. They include 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 Document Type Definition (DTD) is supplied.
- **DB2_Score.c** is C code for DB2 scalar User Defined Functions for each of the output variables defined in the scoring code.
- **Score.c** is the model score code that **is** generated as a C function. It is C source code and must be compiled before it can be executed.

DB2 User-Defined Functions

In addition to generating the scoring algorithms that are developed in SAS 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 UDFs that are generated by SAS Enterprise

Miner 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 SAS 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 SAS 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.

SAS Enterprise Miner can generate score functions that return values that are not useful in a scoring context. The UDFs that SAS 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 type of computed variable) with the name of the corresponding target variable (or decision data set).

SAS 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

SAS 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_ <i>targetname</i> 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 that is 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 SAS 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.

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. SAS Enterprise Miner is capable of producing UDF code that contains more than 91 arguments, but DB2 cannot use any of the additional arguments.

DB2 Data Types

The UDFs that SAS 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 SAS Enterprise Miner generates in order to convert data types for scoring.

C Code Usage

To compile, link, and run C code that is generated in SAS Enterprise Miner, you need to first gather the required tools, libraries, and files.

The generated C code conforms to the “ISO/IEC 9899 International Standard for Programming Languages – C”, so any current compiler should be able to compile the code.

Other than the standard C libraries, the generated C code will have dependencies on the SAS Stand-alone Formats libraries. See the SAS Formats section below for details.

The generated C code also depends on three C header files.

- **cscore.h**
- **csparm.h**
- **jazz.h**

The **cscore.h** and **csparm.h** files are distributed with the SAS Enterprise Miner Server Windows systems. They are located in **SASROOT\dmisc\sasmisc**. For UNIX systems, they are located in **SASROOT/misc/dmisc**. Copy them to your development environment. The **cscore.h** file has several operating system specific definitions that will, in most cases, need to be modified for your target operating system. Those modifications are documented in comments in the header file and in the following example.

The **jazz.h** header file is distributed with the SAS Stand-alone Formats product. See details below in the C Formats Support section. Copy it to your development environment.

To run the generated C code, you need to create a main program to invoke the score function. You can view the **score.c** file and inspect it to determine how the score function should be called.

The metadata file **Cscore.xml** also describes the generated function and its arguments. By default, the generated function accepts two pointers as arguments. The first argument points to an array of input data values. The second argument points to an array of output data values. The memory, which is required for each data value, must be allocated by the calling program.

Both arrays must be composed of the PARM data structure that is 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 for each character value is not allocated before 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 that are 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. Such a main program can be as simple as the code in [Appendix 3](#).

C Formats Support

The C scoring code that is generated in SAS Enterprise Miner supports the use of SAS System Formats through the SAS Stand-alone Formats product. The SAS Stand-alone Formats do not depend on a SAS System environment in any way. The SAS Stand-alone Formats product contains a header file. It also contains a set of libraries that are needed for compilation, linking, and running the SAS Enterprise Miner-generated C scoring function. The SAS

Stand-alone Formats are a set of link and run-time libraries that are compiled for each supported operating system. Those include the following:

- Windows 32-bit
- Windows Itanium 64-bit
- Windows x64-bit
- Solaris 64-bit
- AIX 64-bit
- Linux 32-bit
- Linux 64-bit
- HP-Itanium 64-bit
- HP 64-bit

C Formats Support Distribution

The SAS Stand-alone Formats are distributed as downloads from the SAS Customer Support website. Look in the Knowledge Base section for Samples and SAS Notes. Search for Note 35872, titled “SAS Stand-alone Formats for SAS Enterprise Miner C Score Code.” Follow the instructions to download the package for your target operating system.

C Formats Usage

The C scoring function or application that is generated in SAS Enterprise Miner is linked to **jazzfbrg**. **This means** that at run time the code in **jazzfbrg** can dynamically load the rest of the routines that are needed to support the SAS System formats. Although only **jazzfbrg** might need to be present when linking the function or application, all of the files must be available at run time.

For the Stand-alone 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 that are 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 directory path for the Stand-alone Formats shared libraries (**jazwf***). The value of this environment variable must be the fully qualified directory name for the directory that holds the Stand-alone Formats.

On Solaris systems, the Stand-alone Formats are dynamically loaded from shared libraries via the environment variable LD_LIBRARY_PATH. HP and UNIX systems use a slightly different environment variable, SHLIB_PATH. A thorough understanding of your target system’s procedures for compiling, linking, and running with dynamically loaded code is required to successfully exploit the Stand-alone Formats and the code that is generated by the SAS Enterprise Miner C Scoring component.

For environments where the Stand-alone Formats support is not available or not desired, it should be possible for an experienced C programmer to modify

the source code in the **cscore.h** header file that is distributed with SAS Enterprise Miner in **SASROOT/dmine/sasmisc**. The object of the modifications is to remove the dependency on the Stand-alone Formats and to support any format that they want, with their own C code.

If you write your own format functions, you can integrate those functions into the logic that handles formats in **cscore.h**. The **cscore.h** file that is distributed with SAS Enterprise Miner already contains two examples of such C formatting code—partial support for \$CHAR and BEST formats. If your situation enables you to accept the limitations of those examples (no padding for \$CHAR, and no scientific notation for BEST), you can use the example formats without any modification. You can also add any additional formats that you might need. In that case, C scoring code that is generated by SAS Enterprise Miner will contain only those formats, and will be compiled with the **cscore.h** header file that will support those formats.

In cases where Stand-alone Formats support is not desired, the dependency on the Stand-alone formats support can be removed. This can be accomplished by modifying a copy of the **cscore.h** header file that is distributed with SAS Enterprise Miner in **SASROOT/dmine/sasmisc**. In the **cscore.h** file, the preprocessor symbol **FMTLIB** is set to 0, which disables support for the SAS Stand-alone formats.

Generated Java Code

Java scoring code is generated in several output files. The primary model logic is generated as a Java class file. The other files are generated as Java source files, and the model's variable metadata is encoded as XML. The Java code that is generated by SAS Enterprise Miner is compatible with the version of Java that SAS Enterprise Miner uses. The generated Java files might include some or all of the following:

- **DS.class** is the actual DATA step code that is generated directly to Java binary code. There is no Java source code supplied.
- **DS_UEXIT.java** is generated only if code from unsupported tools was omitted from the generated Java code. This Java source code is a template that customers 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 is supplied.
- **JscoreUserFormats.java** is the Java source code that supports any user-written formats that might be used in the model. It is Java 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.

After you run a SAS 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 SAS Enterprise Miner Reference Help.

Java Package Name

The code that is generated by SAS Enterprise Miner 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) searches for the related source and class files by the package name in a path relative to the current working directory. The Java compiler uses the package name to form a hierarchical path for each related file. For example, if the package name has the default of "eminer.user.Score," the Java compiler searches for the package's files in the path **eminer\user\Score**. In order to compile the generated Java source code, all of the generated Java files (Jscore.xml is not required) must be placed in a directory or folder tree that looks like the package name "eminer.user.Score." You can change the default package name in the SAS Enterprise Miner Client before the flow is run. On the the main menu, select **Options** ➔ **Preferences**. Then fill in a package name of at least two levels.

Java Code Usage

Java scoring code that is generated by SAS Enterprise Miner depends on the classes and methods that are distributed as the SAS Enterprise Miner Java Scoring JAR files. In order to compile or run Java score code that is generated by SAS Enterprise Miner, you need to copy the supporting Java archives and configure your system to make the JAR files available to Java.

Java Scoring JAR Files

The SAS Enterprise Miner Java Scoring JAR files support the classes and methods that are used in the generated Java code, including the use of SAS formats. The SAS Enterprise Miner Java Scoring JAR files are as follows:

- **dtj.jar**
- **icu4j.jar**
- **sas.analytics.eminer.jsutil.jar**
- **sas.core.jar**
- **sas.core.nls.jar**
- **sas.icons.jar**
- **sas.icons.nls.jar**
- **sas.nls.collator.jar**
- **tkjava.nls.jar**

These JAR files include support for most, but not all of the SAS System formats. The list of supported Java formats is detailed in [Appendix 4](#).

Java Scoring JAR File Distribution

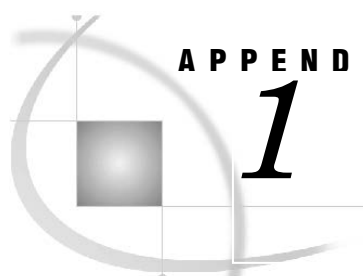
The SAS Enterprise Miner Java scoring JAR files are distributed as part of the SAS Enterprise Miner Server image. On Windows systems, they are found in the path `SASROOT\dmisc\sasmisc`. For UNIX systems, check the path `SASROOT/misc/dmisc`. It is recommended that you save copies of your Java scoring JAR files in your scoring environment.

Java Scoring JAR File Usage

Wherever you want to compile and run the Java code that is generated by SAS Enterprise Miner, you need to make the SAS Enterprise Miner Java scoring JAR files available to Java. Adding the directory path that contains your Java scoring JAR files to your CLASSPATH environment variable enables both the compile and execution steps.

SAS System Formats

Supported SAS System formats are listed in Appendix 4.



APPENDIX

1

Programming Information

<i>General Code Limitations</i>	35
<i>Supported Functions</i>	36
<i>Supported SAS Operators</i>	37
<i>Arithmetic Operators</i>	37
<i>Comparison Operators</i>	37
<i>Logical Operators</i>	37
<i>Other Operators</i>	37
<i>Conditional Statement Syntax</i>	38
<i>Variable Name Length</i>	38
<i>Character Data Length</i>	38
<i>Extended Character Sets</i>	38

General Code Limitations

The SAS DATA step language is a flexible and powerful development environment. The SAS 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 SAS Enterprise Miner produces is compatible with the restrictions that are imposed by the C and Java code generation process.

It is possible to create code in SAS Enterprise Miner that cannot be correctly translated into C or Java code. This is particularly a problem with data transformations that are performed within SAS Enterprise Miner. When you use the SAS 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. This facilitates the correct generation of score code. It is best to attempt to structure DATA step code for any transformation to make it 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 SAS 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	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	Mnemonic	C Equivalent
&	AND	&&
	OR	
^	NOT	!

Other Operators

In SAS, the concatenation operator (| |) concatenates character values. SAS 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 you are comparing a character type variable with a quoted character constant, the quoted character constant must be the second operand.

Variable Name Length

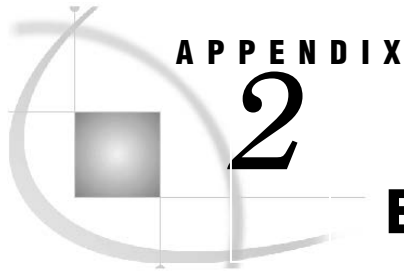
SAS Enterprise Miner truncates column names at 32 bytes. You should exercise care during the ETL process for training data and scoring data in order to make sure that the first 32 characters of all column names are unique.

Character Data Length

The maximum allowable length for character data in SAS Enterprise Miner is 32 bytes. You should exercise care during the ETL process for training data and scoring data to make 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 SAS Enterprise Miner generates depends on 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.



Example Java Main Program

```
import eminer.user.Score.*;
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("CHECKING",((Object)new Double(1.0)));
        indata.put("DURATION",((Object)new Double(6.0)));
        indata.put("HISTORY",((Object)new Double(4.0)));
        indata.put("PURPOSE",((Object)"3"));

        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("_WARN_ = " +
                (String)outdata.get("_WARN_"));
        } catch (Exception ex) {
            System.out.println("Exception caught....Scoring failed");
            return;
        }

        // load obs2 data into input Map
        indata.put("CHECKING",((Object)new Double(1.0)));
        indata.put("DURATION",((Object)new Double(42.0)));
        indata.put("HISTORY",((Object)new Double(2.0)));
        indata.put("PURPOSE",((Object)"2"));
```

```

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

    // process scoring output

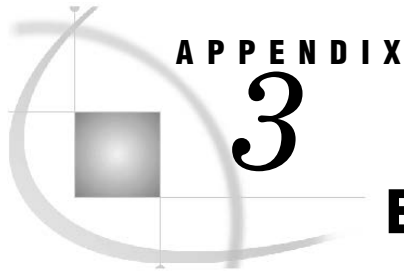
    System.out.println("\n>> Second observation...");

    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("_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
 *           SAS Enterprise Miner 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 UNIX big endian systems */
#define UNX_BE_MISSING  (*((double*)" \xff\xff\xfe\0\0\0\0"))

/* Set the system specific value of missing */
#define MISSING UNX_BE_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);

    /*-----
    * Initialize outputs to type appropriate for the 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);

```

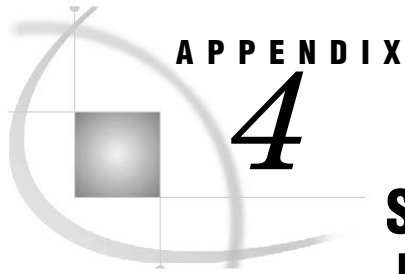
```

/*-----
 * 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 */
}

```



SAS System Formats Supported Java Scoring

\$	CATDFDT	CSYDFDT
\$ASCII	CATDFDWN	CSYDFDWN
\$BINARY	CATDFMN	CSYDFMN
\$CHAR	CATDFMY	CSYDFMY
\$F	CATDFWDX	CSYDFWDX
\$HEX	CATDFWKX	CSYDFWKX
\$OCTAL	COMMA	DANDFDD
AFRDFDD	COMMAX	DANDFDDB
AFRDFDDB	COMMAX	DANDFDDC
AFRDFDDC	CRODFDD	DANDFDDD
AFRDFDDD	CRODFDDB	DANDFDDP
AFRDFDDP	CRODFDDC	DANFDDDS
AFRDFDDS	CRODFDDD	DANFDE
AFRDFDE	CRODFDDP	DANFDN
AFRDFDN	CROFDDDS	DANFDT
AFRDFDT	CROFDE	DANFDWN
AFRDFDWN	CROFDN	DANFMN
AFRDFMN	CROFDT	DANFMY
AFRDFMY	CROFDWN	DANFWDX
AFRDFWDX	CROFMN	DANFWKX
AFRDFWKX	CROFMY	DATE
BEST	CROFWDX	DATEAMPM
BINARY	CROFWKX	DATETIME
CATDFDD	CSYDFDD	DAY
CATDFDDB	CSYDFDDB	DDMMYY
CATDFDDC	CSYDFDDC	DDMMYYB
CATDFDDD	CSYDFDDD	DDMMYYC
CATDFDDP	CSYDFDDP	DDMMYYD
CATDFDDS	CSYDFDDS	DDMMYYN
CATDFDE	CSYDFDE	DDMMYYP
CATDFDN	CSYDFDN	DDMMYYs

DESDFDD	ENGDFDE	FINDFDDS
DESDFDDB	ENGDFDN	FINDFDE
DESDFDDC	ENGDFDT	FINDFDN
DESDFDDD	ENGDFDWN	FINDFDT
DESDFDDP	ENGDFMN	FINDFDWN
DESDFDDS	ENGDFMY	FINDFMN
DESDFDE	ENGDFWDX	FINDFMY
DESDFDN	ENGDFWKX	FINDFWDX
DESDFDT	ESPDFDD	FINDFWKX
DESDFDWN	ESPDFDDB	FRADFDD
DESDFMN	ESPDFDDC	FRADFDDB
DESDFMY	ESPDFDDD	FRADFDDC
DESDFWDX	ESPDFDDP	FRADFDDD
DESDFWKX	ESPDFDDS	FRADFDDP
DEUDFDD	ESPDFDE	FRADFDDS
DEUDFDDB	ESPFDN	FRADFDE
DEUDFDDC	ESPFDT	FRAFDN
DEUDFDDD	ESPFDWN	FRAFDT
DEUDFDDP	ESPDFMN	FRAFDWN
DEUDFDDS	ESPDFMY	FRAFMN
DEUDFDE	ESPDFWDX	FRAFMY
DEUDFDN	ESPDFWKX	FRAFWDX
DEUDFDT	EURDFDD	FRAFWKX
DEUDFDWN	EURDFDDB	FRSDFDD
DEUDFMN	EURDFDDC	FRSDFDDB
DEUDFMY	EURDFDDD	FRSDFDDC
DEUDFWDX	EURDFDDP	FRSDFDDD
DEUDFWKX	EURDFDDS	FRSDFDDP
DOLLAR	EURDFDE	FRSDFDDS
DOLLARX	EURFDN	FRSDFDE
DOWNAME	EURFDT	FRSFDN
DTDATE	EURFDWN	FRSFDT
DTMONYY	EURDFMN	FRSFDWN
DTWKDATX	EURDFMY	FRSDFMN
DTYEAR	EURDFWDX	FRSDFMY
DTYYQC	EURDFWKX	FRSDFWDX
E	EURO	FRSDFWKX
ENGDFDD	F	HEX
ENGDFDDB	FINDFDD	HHMM
ENGDFDDC	FINDFDDB	HOURL
ENGDFDDD	FINDFDCC	HUNDFDD
ENGDFDDP	FINDFDDD	HUNDFDDB
ENGDFDDS	FINDFDDP	HUNDFDDC

HUNDFDDD	MMDDYY	NLMNICHF
HUNDFDDP	MMDDYYB	NLMNICNY
HUNDFDDS	MMDDYYC	NLMNIDKK
HUNDFDE	MMDDYYD	NLMNIEUR
HUNDFDN	MMDDYYN	NLMNIGBP
HUNDFDT	MMDDYYP	NLMNIHKD
HUNDFDWN	MMDDYYS	NLMNIILS
HUNDFMN	MMSS	NLMNIJPY
HUNDFMY	MMYY	NLMNIKRW
HUNDFWDX	MMYYB	NLMNIMYR
HUNDFWKX	MMYYC	NLMNINOK
ITADFDD	MMYYD	NLMNINZD
ITADFddb	MMYYN	NLMNIPLN
ITADFDDC	MMYYP	NLMNIRUR
ITADFDDD	MMYYS	NLMNISEK
ITADFDDP	MONNAME	NLMNISGD
ITADFDDS	MONTH	NLMNITWD
ITADFDE	MONYY	NLMNIUSD
ITADFDN	NEGPAREN	NLMNIZAR
ITADFDT	NLDATE	NLMNLAUD
ITADFDOWN	NLDATEMN	NLMNLCAD
ITADFMN	NLDATEW	NLMNLCHF
ITADFMY	NLDATEWN	NLMNLCNY
ITADFWDX	NLDATM	NLMNLDDK
ITADFWKX	NLDATMAP	NLMNLEUR
JULDATE	NLDATMTM	NLMNLGBP
JULDAY	NLDATMW	NLMNLHKD
JULIAN	NLDDFDD	NLMNLILS
LOGPROB	NLDDFddb	NLMNLJPY
MACDFDD	NLDDFDDC	NLMNLKRW
MACDFddb	NLDDFDDD	NLMNLMYR
MACDFDDC	NLDDFDDP	NLMNLNOK
MACDFDDD	NLDDFDDS	NLMNLNZD
MACDFDDP	NLDDFDE	NLMNLPLN
MACDFDDS	NLDDFDN	NLMNLRUR
MACDFDE	NLDDFDT	NLMNLSEK
MACDFDN	NLDDFDWN	NLMNLSGD
MACDFDT	NLDDFMN	NLMNLTWD
MACDFDOWN	NLDDFMY	NLMNLUSD
MACDFMN	NLDDFWDX	NLMNLZAR
MACDFMY	NLDDFWKX	NLMNY
MACDFWDX	NLMNIAUD	NLMNYI
MACDFWKX	NLMNICAD	NLNUM

NLNUMI	PTGDFDE	SLODFWDX
NLPCT	PTGDFDN	SLODFWKX
NLPCTI	PTGDFDT	SVEDFDD
NLTIMAP	PTGDFDWN	SVEDFDDB
NLTIME	PTGDFMN	SVEDFDDC
NORDFDD	PTGDFMY	SVEDFDDD
NORDFDDB	PTGDFWDX	SVEDFDDP
NORDFDDC	PTGDFWKX	SVEDFDDS
NORDFDDD	PVALUE	SVEDFDE
NORDFDDP	QTR	SVEDFDN
NORDFDDS	QTRR	SVEDFDT
NORDFDE	RSTDOCNY	SVEDFDWN
NORDFDN	RSTDOCYY	SVEDFMN
NORDFDT	RSTDONYN	SVEDFMY
NORDFDWN	RSTDOPNY	SVEDFWDX
NORDFMN	RSTDOPYN	SVEDFWKX
NORDFMY	RSTDOPYY	TIME
NORDFWDX	RUSDFDD	TIMEAMPM
NORDFWKX	RUSDFDDB	TOD
NUMX	RUSDFDDC	WEEKDATE
OCTAL	RUSDFDDD	WEEKDATX
PERCENT	RUSDFDDP	WEEKDAY
PERCENTN	RUSDFDDS	WEEKU
POLDFDD	RUSDFDE	WEEKV
POLDFDDB	RUSFDN	WEEKW
POLDFDDC	RUSFDT	WORDDATE
POLDFDDD	RUSFDWN	WORDDATX
POLDFDDP	RUSDFMN	YEAR
POLDFDDS	RUSDFMY	YEN
POLDFDE	RUSDFWDX	YEN
POLDFDN	RUSDFWKX	YYMM
POLDFDT	SLODFDD	YYMMB
POLDFDWN	SLODFDDB	YYMMC
POLDFMN	SLODFDDC	YYMMD
POLDFMY	SLODFDDD	YYMMDD
POLDFWDX	SLODFDDP	YYMMDDB
POLDFWKX	SLODFDDS	YYMMDDC
PTGDFDD	SLODFDE	YYMMDDD
PTGDFDDB	SLOFDN	YYMMDDN
PTGDFDDC	SLOFDT	YYMMDDP
PTGDFDDD	SLOFDWN	YYMMDDS
PTGDFDDP	SLODFMN	YYMMN
PTGDFDDS	SLODFMY	YYMMP

YYMMS
YYMON
YYQ
YYQB
YYQC
YYQD

YYQN
YYQP
YYQR
YYQRB
YYQRC
YYQRD

YYQRN
YYQRP
YYQRS
YYQS

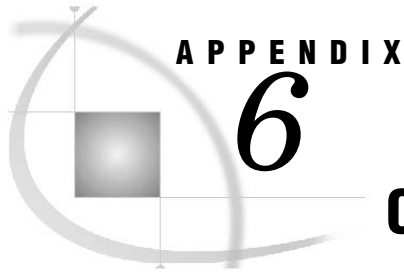


APPENDIX 5

SAS System Formats Supported for C Scoring

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

MACDFMN	NLMNICNY	NLTIME	RUSDFDE	TOD
MACDFMY	NLMNIDKK	NORDFDD	RUSDFDN	VAXRB
MACDFWDX	NLMNIEUR	NORDFDE	RUSDFDT	WEEKDATE
MACDFWKX	NLMNIGBP	NORDFDN	RUSDFDWN	WEEKDATX
MDYAMP	NLMNIHKD	NORDFDT	RUSDFMN	WEEKDAY
MINGUO	NLMNIILS	NORDFDWN	RUSDFMY	WORDDATE
MMDDYY	NLMNIJPY	NORDFMN	RUSDFWDX	WORDDATX
MMDDYYB	NLMNIKRW	NORDFMY	RUSDFWKX	WORDF
MMDDYYC	NLMNIMYR	NORDFWDX	S370FF	WORDS
MMDDYYD	NLMNINOK	NORDFWKX	S370FHEX	XPORTFLT
MMDDYYN	NLMNINZD	NUMX	S370FIB	XPORTINT
MMDDYYP	NLMNIPLN	OCTAL	S370FIBU	XYMMDD
MMDDYYSS	NLMNIRUR	ODDSR	S370FPD	YEAR
MMSS	NLMNISEK	PCPIB	S370FPDU	YEN
MMYY	NLMNISGD	PD	S370FPIB	YEN
MMYYC	NLMNITWD	PDJULG	S370FRB	YYMM
MMYYD	NLMNIUSD	PDJULI	S370FZD	YYMMC
MMYYN	NLMNIZAR	PERCENT	S370FZDL	YYMMD
MMYYP	NLMNLAUD	PERCENTN	S370FZDS	YYMMDD
MMYYSS	NLMNLCAD	PIB	S370FZDT	YYMMDDDB
MONNAME	NLMNLCNF	PIBR	S370FZDU	YYMMDDC
MONTH	NLMNLCNY	PK	SETLOCALE	YYMMDDD
MONYY	NLMNLDKK	POLDFDD	SIZEK	YYMMDDN
MRB	NLMNLEUR	POLDFDE	SIZEKB	YYMMDDP
NEGPAREN	NLMNLGBP	POLDFDN	SIZEKMG	YYMMDDS
NENGO	NLMNLHKD	POLDFDT	SLODFDD	YYMMN
NLDATE	NLMNLILS	POLDFDWN	SLODFDE	YYMMP
NLDATEMN	NLMNLJPY	POLDFMN	SLODFDN	YYMMS
NLDATEW	NLMNLKRW	POLDFMY	SLODFDT	YYMON
NLDATEWN	NLMNLMYR	POLDFWDX	SLODFDWN	YYQ
NLDATM	NLMNLNOK	POLDFWKX	SLODFMN	YYQC
NLDATMAP	NLMNLNZD	PTGDFDD	SLODFMY	YYQD
NLDATMTM	NLMNLPLN	PTGDFDE	SLODFWDX	YYQN
NLDATMW	NLMNLRUR	PTGDFDN	SLODFWKX	YYQP
NLDDFDD	NLMNLSEK	PTGDFDT	SSN	YYQR
NLDDFDE	NLMNLSGD	PTGDFDWN	SVEDFDD	YYQRC
NLDDFDN	NLMNLTWD	PTGDFMN	SVEDFDE	YYQRD
NLDDFDT	NLMNLUSD	PTGDFMY	SVEDFDN	YYQRN
NLDDFDWN	NLMNLZAR	PTGDFWDX	SVEDFDT	YYQRP
NLDDFMN	NLMNY	PTGDFWKX	SVEDFDWN	YYQRS
NLDDFMY	NLMNYI	PVALUE	SVEDFMN	YYQS
NLDDFWDX	NLNUM	QTR	SVEDFMY	YYQZ
NLDDFWKX	NLNUMI	QTRR	SVEDFWDX	Z
NLMNIAUD	NLPCT	RB	SVEDFWKX	ZD
NLMNICAD	NLPCTI	ROMAN	TIME	
NLMNICHF	NLTIMAP	RUSDFDD	TIMEAMP	



C Compiler Command Examples

<i>C Compiler Command Examples</i>	53
<i>C Compiler Command Examples</i>	53
<i>W32 – Windows 32-bit (x86)</i>	53
<i>LAX- Linux for x64 (x86-64)</i>	55
<i>LNX- Linux 32-bit (x86)</i>	56
<i>H64- HP-UX on PA-RISC</i>	57
<i>H6I- HP-UX on Itanium</i>	58
<i>R64 – AIX on Power</i>	59
<i>S64 - Solaris on SPARC</i>	60
<i>SAX – Solaris 10 x64 (x64-86)</i>	61

C Compiler Command Examples

The compiler options in the following examples are provided only as a possible starting point. These examples are for producing a Windows dynamic link library (DLL) or a UNIX Shared library. There are in most cases many options that could be applied to the compilation of the SAS Enterprise Miner C score code that are not mentioned here. Experience, careful research, and experimentation are required to optimize the run-time performance of any code using compiler options.

W32 – Windows 32-bit (x86)

OS Name	Microsoft Windows XP Professional
Version	5.1.2600 Service Pack 2 Build 2600
System	X86-based PC
Compiler	Microsoft Visual C++ version 6.0
Compiler Documentation	http://msdn.microsoft.com/en-us/library/9s7c9wdw(VS.80).aspx

Compile command	<code>cl /c /nologo /Zp4 /I"C:\Temp\Ccode\headers" "C:\Temp\Ccode\Score.c" & link /nologo /dll Score.obj jazzfbrg.lib /libpath:"C:\Temp\SAFMTS" /out:"C:\Temp\Ccode\testscore.dll" & exit</code>
cl	The Microsoft Visual C/C++ command line compiler
/c	Specifies that the compiler should compile only (linking is not performed)
/nologo	Suppresses display of sign-on banner
/Zp4	Packs structures on 4-byte boundaries.
/I "C:\Temp\Ccode\headers"	Adds a directory to the path list that is searched for include files.
"C:\Temp\Ccode\Score.c"	The name of the C source file
&	Separates DOS command-line commands
Link	The name of the Windows linker
/dll	The link option to build a DLL
Score.obj	The name of the object file that is produced by the compile command
jazzfbrg.lib	The link library for the Stand-alone format
/libpath:"C:\Temp\SAFMTS"	Specifies a path that the linker will search first to resolve references
/out: "C:\Temp\Ccode\testscore.dll"	Specifies the output filename
exit	DOS command that is used to close the window when running as batch or when opened for external commands

Note: If you specify the Microsoft Visual C/C++ compiler (**-cl**) and if the pathname in your **/out:** specification contains embedded spaces, you must use the short (DOS 8.3 specification) versions of the directory names, instead of the (DOS 9 and later) long names with spaces. When the compiler passes incorrect pathnames to the linker, the linker will flag error code LNK1181. You use the DOS command **dir /x** from the parent directory to display the short and long name DOS names for a child directory. Alternatively, you can use separate compile and link commands to avoid pathname problems.

LAX- Linux for x64 (x86-64)

OS Name	SUSE LINUX Enterprise Server
Version	SUSE LINUX Enterprise Server 9 (x86_64)
Compiler	GNU C version 3.3.3 (SuSE Linux)
Compiler Documentation	http://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/index.html#toc_Invoking-GCC
Compile command	<code>cc -std=iso9899:1999 -shared -fPIC -I/tstest1/headers -o/tstest1/libtstest1 /tstest1/Score.c /sas920/safmts/jazxfbrg</code>
-std=iso9899:1999	Specifies the revised ISO C standard, published in December 1999
-shared	A shared object that can then be linked with other objects to form an executable
-fPIC	Generate position-independent code (PIC) suitable for use in a shared library
-I<i>dir</i>	Add the directory <i>dir</i> to the head of the list of directories to be searched for header files
-o <i>filename</i>	Place output in file <i>filename</i>
Score.c	The SAS Enterprise Miner C scoring code
jazxfbrg	Contains the required Stand-alone formats functions that need to be linked in

LNx- Linux 32-bit (x86)

OS Name	Novell SuSE SLES 9 x86-
Version	SUSE LINUX Enterprise Server 9 (i586)
Compiler	Intel® C++ Compiler for Linux version 9.0
Compiler Documentation	http://software.intel.com/en-us/articles/intel-c-compiler-for-linux-9x-manuals/
Compile command	<code>icc -shared -i_static -strict-ansi -I/users/sasled/tstest1/headers -o/users/sasled/tstest1/libtstest1 /users/tstest1/Score.c/sas920/ safmts/jazxfbrg</code>
-shared	Tells both the compiler and linker to produce a dynamic shared object
-i_static	Statically links libraries that are provided by Intel.
-strict-ansi	Used for strict ANSI conformance. This avoids conflicts with file extensions.
-I<i>dir</i>	Add the directory <i>dir</i> to the head of the list of directories to be searched for header files
-o <i>filename</i>	Places the output in file <i>filename</i>
Score.c	SAS Enterprise Miner C scoring code
jazxfbrg	Contains the required Stand-alone formats functions that need to be linked in

H64- HP-UX on PA-RISC

OS Name	HP-UX 11.23
Version	HP-UX B.11.23 U 9000/800
Compiler	-01 B.11.X.32509-32512.GP HP C Compiler
Compiler Documentation	http://h21007.www2.hp.com/portal/download/files/unprot/hpux/HP%20C%20HPUX%20Reference%20Manual.pdf
Compile command	/usr/bin/cc -v -V -b +Z +DD64 -q -I/ tstest1/headers -o/ tstest1/libtstest1 / tstest1/Score.c /sas920/safmts/jazxfbrg >> / tstest1/cclist.txt 2>&1
-v	Causes sub processes to print version information to stderr
-v	Enables verbose mode
-b	Creates a shared library rather than an executable file. The object files must have been created with the +z or +Z option to generate position-independent code (PIC).
+Z	Generates shared library object code with a large data linkage table (long-form PIC). +DD64 generates 64-bit object code for PA2.0 architecture.
+DD64	Generates 64-bit object code for PA2.0 architecture
-q	Causes the output file from the linker to be marked as demand loadable. For details and system defaults, see the ld(1) description in the HP-UX Reference Manual.
-Idir	Add the directory <i>dir</i> to the head of the list of directories to be searched for include files by the preprocessor
-o filename	Places the output in file <i>filename</i>
/tstest1/Score.c	The absolute name of the SAS Enterprise Miner generated C scoring source file
/sas920/safmts/jazxfbrg	The Stand-alone formats link library

H6I- HP-UX on Itanium

OS Name	HP-UX 11.23
Version	HP-UX B.11.23 U ia64
Compiler	HP aC++/ANSI C B3910B A.06.06 [Nov 7 2005]
Compiler Documentation	http://h21007.www2.hp.com/portal/download/files/unprot/hpux/HP%20C%20HPUX%20Reference%20Manual.pdf
Compile command	<pre>/usr/bin/cc -v -V -b +Z +DD64 -q -I/ tstest1/headers -o/ tstest1/libtstest1 / tstest1/Score.c /sas920/safmts/jazxfbrg >> / tstest1/cclist.txt 2>&1</pre>
-v	Causes sub processes to print version information to stderr
-v	Enables verbose mode
-b	Creates a shared library rather than an executable file. The object files must have been created with the +z or +Z option to generate position-independent code (PIC).
+Z	Generates shared library object code with a large data linkage table (long-form PIC). +DD64 generates 64-bit object code for PA2.0 architecture.
+DD64	Generates 64-bit object code for PA2.0 architecture
-q	Causes the output file from the linker to be marked as demand loadable. For details and system defaults, see the ld(1) description in the HP-UX Reference Manual.
-I<i>dir</i>	Add the directory <i>dir</i> to the head of the list of directories to be searched for include files by the preprocessor
-o <i>filename</i>	Places the output in file <i>filename</i>
/tstest1/Score.c	The absolute name of the SAS Enterprise Miner generated C scoring source file
/sas920/safmts/jazxfbrg	The Stand-alone formats link library

R64 – AIX on Power

OS Name	AIX
Version	AIX Version 3.5
Compiler	IBM XL C Enterprise Edition for AIX, Version 7.0.0.4
Compiler Documentation	http://publib.boulder.ibm.com/infocenter/c omphelp/v8v101/index.jsp
Compile command	<code>c99 -q64 -G -qlibansi -qarch=com -I/tstest1/headers /tstest1/ Score.c /sas920/safmts/jazxfbrg -lm -o /tstest1/libtstest1</code>
-qarch=com	produces object code that will run on all the 64-bit PowerPC(R) hardware platforms but not 32-bit-only platforms
-q64	Generates 64-bit code
-qlibansi	Configures the optimizer to generate better code because it will know about the behavior of a standard function
-G	Specifies the linker that is to create a shared object enabled for run-time linking
-lm	Specifies the standard math library for linking. Some configurations might not require this.
-I <i>dir</i>	Specifies an additional search path for #include filenames
-o <i>filename</i>	Specifies an output location and name for the shared library
/tstest1/Score.c	The absolute name of the SAS Enterprise Miner generated C scoring source file
/sas920/safmts/jazxfbrg	the Stand-alone formats link library

S64 - Solaris on SPARC

OS Name	Solaris 9
Version	SunOS 5.8 Generic February 2000 (also known as Solaris 8)
Compiler	Sun C 5.7 Patch 117836-02 2005/03/23
Compiler Documentation	http://www.oracle.com/pls/topic/lookup?ctx=dsc&id=/app/docs/doc/819-3688
Compile command	<code>cc -v -G -xtarget=ultra3 -xarch=v9a -xcode=pic32 -I/tstest1/headers -o /tstest1/libtstest1/tstest1/ Score.c/sas920/safmts/jazxfbrg</code>
-G	Specifies that the linker is to create a shared object enabled for run-time linking
-xcode=pic32	Generates position-independent code for use in shared libraries (large models). Equivalent to -KPIC.
-xtarget=ultra3	Specifies the target system for instruction set and optimization
-xarch=v9a	Specifies the instruction set architecture (ISA). If you use this option with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice results in a binary program that is not executable on the intended target platform.
-I/tstest1/headers	Adds the specified directory to the list of directories that are searched for #include files
-o/tstest1/libtstest1	Specifies the output file filename instead of using the default filename of a.out . The specified filename cannot be the same as the source file. This option and its arguments are passed to ld(1).
/tstest1/Score.c	The absolute name of the SAS Enterprise Miner generated C scoring source file
/sas920/safmts/jazxfbrg	The Stand-alone formats link library

SAX – Solaris 10 x64 (x64-86)

OS Name	SunOS
Version	SunOS 5.10 Generic January 2005 i86pc
Compiler	C 5.9 SunOS_i386 Patch 124868-01 2007/07/12
Compiler Documentation	http://www.oracle.com/pls/topic/lookup?ctx=dsc&id=/app/docs/doc/819-3688
Compile command	<code>cc -V -v -G -xtarget=opteron -xarch=amd64a -KPIC -I/tstest1/headers -o /tstest1/libtstest1/tstest1/ Score.c/sas920/safmts/jazzfbrg</code>
-G	Specifies that the linker is to create a shared object enabled for run-time linking
-xtarget=opteron	Specifies the target system for instruction set and optimization
-xarch=amd64a	Specifies the instruction set architecture (ISA). If you use this option with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice results in a binary program that is not executable on the intended target platform.
-KPIC	Generates position-independent code for use in shared libraries
-I/tstest1/headers	Adds the specified directory to the list of directories that are searched for #include files
-o/tstest1/libtstest1	Specifies the output file filename instead of using the default filename of a.out . The specified filename cannot be the same as the source file. This option and its arguments are passed to ld(1).
/tstest1/Score.c	The absolute name of the SAS Enterprise Miner generated C scoring source file
/sas920/safmts/jazzfbrg	The Stand-alone formats link library

